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

RENATA DE MATOS GALANTE

Modelo Temporal de Versionamento com Suporte à Evolução de Esquemas

Tese apresentada como requisito parcial
para a obtenção do grau de
Doutor em Ciência da Computação

Prof. Dr. Clesio Saraiva dos Santos
Orientador

Prof. Dra. Nina Edelweiss
Co-orientadora

Porto Alegre, dezembro de 2003

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Galante, Renata de Matos

Modelo Temporal de Versionamento com Suporte à Evolução de Esquemas / Renata de Matos Galante. – Porto Alegre: PPGC da UFRGS, 2003.

150 f.: il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2003. Orientador: Clesio Saraiva dos Santos; Co-orientadora: Nina Edelweiss.

1. Bancos de Dados Temporais Orientados a Objetos. 2. Evolução de Esquemas. 3. Versionamento de Esquemas. 4. Semântica Operacional. I. Santos, Clesio Saraiva dos. II. Edelweiss, Nina. III. Título.

“O real não está na saída e nem na chegada, está na travessia.”
— JOÃO GUIMARÃES ROSA

AGRADECIMENTOS

A todas as pessoas que auxiliaram e se interessaram por esse trabalho, os meus sinceros agradecimentos. Algumas pessoas, no entanto, tiveram uma contribuição mais direta. A elas eu deixo uma lembrança especial.

Iniciando os agradecimentos pela *"dupla dinâmica"* (termo bem definido pela Mirella), os professores Nina e Clesio, que acompanharam o desenvolvimento deste trabalho desde a sua concepção, estando sempre abertos ao diálogo, motivando nas tarefas e no convívio humano. Ao professor Clesio, que me acompanha de pertinho desde o início do mestrado, pela condução serena do trabalho e pela excelente orientação prestada durante o longo período de elaboração desta tese. À professora Nina, pela orientação precisa, pela confiança que sempre imprimiu às nossas discussões, por ter-me viabilizado escolher os meus próprios caminhos e neles ter trilhado comigo.

Meus agradecimentos aos funcionários e professores do Instituto de Informática. Aos professores Heuser e Mara Abel, por terem me "incentivado" a colocar um ponto final nesta tese. A Álvaro Moreira, por me introduzir no mundo da semântica operacional e por ser uma pessoa de referência para discussões sobre a continuidade deste trabalho. Uma lembrança especial à professora Lia Golendziner (*in memorian*) que, de modo brilhante, introduziu o tema central desta tese, no qual eu tenho trabalhado desde o mestrado.

Agradecimentos ao Instituto de Informática e ao Programa de Pós-Graduação em Ciência da Computação da UFRGS, pela oportunidade oferecida para realização deste trabalho e à CAPES, pelo suporte financeiro.

Nenhum trabalho é absolutamente solitário, em especial o intelectual. Agradeço aos que trabalharam no projeto "Tempo-Versões" desde sua origem, em especial à Mirella por seu trabalho pioneiro, pelo seu rigor e crítica científica. A Fábio pelas sugestões valiosas que motivaram especialmente a argumentação da tese. A Carlos Eduardo, que contribuiu significativamente para que os resultados fossem alcançados, em particular, com a implementação da ferramenta. Não poderia deixar de registrar um enorme agradecimento aos amigos Carina, Daniela, Eduardo, Fábio, Raquel e Vanessa por terem tido paciência de revisar todo o texto com um "*deadline*" de dois dias.

Agradeço também a todos os amigos que compartilharam esses anos de estudos e expectativas, que sempre auxiliaram tanto no ambiente de trabalho quanto em momentos de descontração, contribuindo com conhecimentos e experiências para a ciência e para a vida particular. Às colegas Carina, Daniela, Mirella e Vanessa, que, com o passar do tempo e depois de inúmeros cafezinhos e bate-papos, foram "promovidas" à categoria de grandes e queridas amigas.

Por último e com certeza não menos importante, à minha família, Ailton, Izabel, Fernanda e Claudia, que têm sido suporte e estímulo sempre presentes, sem os quais este trabalho nunca teria sido possível.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

BDOO	Bancos de Dados Orientados a Objetos
BDT	Bancos de Dados Temporais
BDTOO	Bancos de Dados Temporais Orientados a Objetos
MSQL	*Multi-Schema Query Language*
MTV	*Multi Table Version*
ODL	*Object Definition Language*
ODMG	*Object Data Management Group*
OID	*Object Identifier*
OQL	*Object Query Language*
PMTV	*Partial Multiple Table Versioning*
SGBD	Sistema Gerenciador de Banco de Dados
SIL	Sistema de Informação Legado
STV	*Single Table Version*
TQL	TIGUKAT *Query Language*
TRM	*Temporal Relational Model*
TT	Tempo de Transação
TV	Tempo de Validade
TVM	*Temporal Versions Model*
TVQL	*Temporal Versioned Query Language*
UC	*until change*
UFRGS	Universidade Federal do Rio Grande do Sul
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A utilização de versões tem sido essencial em diversas aplicações de banco dados, por permitir o armazenamento e a manipulação de diferentes estados da base de dados. Durante a evolução de um esquema, o versionamento preserva todas as versões de esquemas e de seus dados associados. Por outro lado, os conceitos de bancos de dados bitemporais, que incorporam tanto tempo de transação quanto tempo de validade, provêm flexibilidade ao mecanismo de evolução de esquemas, não somente por permitir acesso a informações presentes, passadas e futuras, mas também por permitir atualizações e consultas entre as diversas versões de esquemas existentes.

O objetivo principal desta tese é definir um modelo que utilize os conceitos de tempo e de versão para permitir o gerenciamento da evolução dinâmica de esquemas em bancos de dados orientados a objetos. O resultado, o Modelo Temporal de Versionamento com suporte à Evolução de Esquemas (TVSE - *Temporal and Versioning Model to Schema Evolution*), é capaz de gerenciar o processo de evolução de esquemas em todos os seus aspectos: versionamento e modificação de esquemas, propagação de mudanças e manipulação de dados. Esse modelo difere de outros modelos de evolução de esquemas por permitir o gerenciamento homogêneo e simultâneo do histórico da evolução do banco de dados intencional e extensional. Com o objetivo de complementar a definição deste modelo é apresentado um ambiente para gerenciar o versionamento temporal da evolução de esquemas. Desse ambiente foi implementado um protótipo da ferramenta de apoio ao gerenciamento de evolução de esquemas. Por fim, enriquecendo o universo da tese e com o intuito de prover uma maior fundamentação teórica e matemática para descrever as políticas de evolução de esquemas, é especificada uma semântica operacional para um subconjunto do modelo proposto.

Palavras-chave: Bancos de Dados Temporais Orientados a Objetos, Evolução de Esquemas, Versionamento de Esquemas, Semântica Operacional.

Temporal and Versioning Model to Schema Evolution

ABSTRACT

The use of versions has been very important in several database applications, due to its power of allowing the storage and handling of different database states. Versioning is used to preserve all schema versions as well as their instances during schema evolution. On the other hand, bitemporal database concepts, which incorporate transaction and valid timestamps, provide flexibility to schema evolution mechanisms. They do not only allow access to present, past and future information, but also enable updates and queries among the different schema versions.

The main purpose of this thesis is to define a model that uses the time and version concepts to manage the dynamic schema evolution in object-oriented databases. The Temporal and Versioning Model to Schema Evolution - TVSE, which is the result of the research, is able to manage the schema evolution process considering all involved aspects: versioning and schema modification, change propagation and data manipulation. TVSE differs from other schema evolution models by enabling the homogeneous and simultaneous management of the evolution history concerning both intentional and extensional databases. To complement the definition of TVSE, an environment to manage the temporal versioning of schema evolution is also presented. One of the tools of this environment tool was specified and implemented, aiming to support a user during the schema evolution management. Finally, an operational semantics is specified for a subgroup of the proposed model, to provide mathematical and theoretical basis for the description of the schema evolution politics.

Keywords: Temporal Object-Oriented Database,Schema Evolution, Schema Versioning, Operational Semantics.

1 INTRODUÇÃO

1.1 Motivação

Aplicações não convencionais, como por exemplo, CAD (*Computer Aided Design*) e CASE (*Computer-Aided Software Engineering*), freqüentemente exigem a manutenção de diversos estados da base de dados, retendo o histórico das modificações realizadas. Como resposta a tal requisito, é empregado o conceito de versão. Embora com a utilização de versões sejam armazenadas diversas alternativas de projeto, a evolução histórica dos dados não é completamente capturada.

Bancos de dados temporais (ÖZSU et al., 1995; SNODGRASS, 2000; TANSEL et al., 1993) têm por objetivo capturar os aspectos dinâmicos dos objetos do mundo real. Entretanto, durante o ciclo de vida de um banco de dados, não somente os objetos evoluem, mas também o esquema sofre significativas mudanças, tanto nos requisitos funcionais, como, por exemplo, mudanças no domínio das aplicações e/ou especificações de projeto, como também nos requisitos não funcionais, isto é, no desempenho da aplicação.

Evolução de esquemas (BANERJEE et al., 1987; RODDICK, 1995) e versionamento de esquemas (CASTRO; GRANDI; SCALAS, 1995a; RODDICK, 1991) são duas técnicas utilizadas para realizar modificações na estrutura do banco de dados, mantendo a consistência entre o esquema e a base de dados. Enquanto a evolução mantém apenas a versão corrente do esquema e sua respectiva base de dados, o versionamento preserva todas as versões de esquemas, e seus dados associados, durante a evolução.

Bancos de dados temporais, especialmente os bitemporais, que incorporam tanto o tempo de transação quanto o tempo de validade, provêem flexibilidade ao mecanismo de evolução de esquemas, não somente por permitir aos usuários e programas aplicativos acesso a informações temporais (passadas e futuras), mas também, e principalmente, por permitir atualizações e consultas pró-ativas e retroativas entre as diversas versões de esquemas existentes.

O suporte ao versionamento de esquemas com características temporais tem sido estudado extensivamente no âmbito de bancos de dados relacionais (ANGONESE, 2000; CASTRO; GRANDI; SCALAS, 1997; MOREIRA; EDELWEISS, 1999a; WEI; ELMASRI, 2000a). Em ambientes orientados a objetos, a necessidade de alteração do esquema conceitual tem sido identificada em muitos domínios de aplicação, emergindo em sistemas de aplicações não convencionais, como, por exemplo, CAD, sistemas de automação de escritórios, projetos de engenharia e inteligência artificial.

Alguns trabalhos mais recentes têm enfocado a questão do suporte a versões. Um deles, o Modelo de Versões (GOLENDZINER, 1995), propõe uma extensão aplicável a modelos de dados orientados a objetos, pela incorporação de conceitos e mecanismos que

suportam a definição e manipulação de objetos, versões e configurações. Com o objetivo de permitir a representação de todo o histórico das modificações, em (MORO, 2001) foi proposta uma extensão ao Modelo de Versões, com o acréscimo do conceito de tempo: TVM - o Modelo Temporal de Versões (*Temporal Versions Model*). Este modelo permite armazenar as versões de um objeto e, para cada versão desse, o histórico das alterações feitas no valores de seus atributos e relacionamentos dinâmicos, que são temporalizados.

1.2 Objetivos Gerais da Tese

O objetivo central desta tese é definir um modelo que utilize os conceitos de tempo e de versão para permitir o gerenciamento da evolução dinâmica de esquemas em bancos de dados orientados a objetos. Esse modelo é denominado Modelo Temporal de Versionamento com suporte a Evolução de Esquemas (TVSE - *Temporal and Versioning Model to Schema Evolution*).

Primeiramente, a tese introduz as áreas relacionadas ao tema "evolução de esquemas conceituais", identificando o escopo de atuação da mesma. A seguir, são revistas as principais características do TVM, por ser o modelo de dados adotado como base para o gerenciamento da evolução de esquemas. Com intuito de identificar funcionalidades reais e práticas no contexto de evolução de esquemas, é apresentado sucintamente um estudo de caso considerando a evolução do Sistema Discente da UFRGS. Dentro desse contexto, é definido um modelo para evolução dinâmica de esquemas em bancos de dados temporais orientados a objetos que busca identificar e tratar os diversos aspectos relacionados à evolução de esquemas com tempo e versões. Especificamente, o modelo pode ser dividido em três tópicos distintos: gerenciamento do versionamento e modificação de esquemas, aliado as técnicas de propagação das mudanças na base de dados extensional; gerenciamento de dados, que permite a execução de operações de atualização de objetos durante a vigência de versionamento de esquemas; e uma solução para o armazenamento físico dos esquemas e de seus dados associados. Com objetivo de complementar, é apresentado um ambiente para gerenciar o versionamento temporal para evolução de esquemas. Especificamente, é detalhada toda arquitetura proposta para o ambiente, sendo proposto um mecanismo híbrido para implementação da base de dados extensional. Desse ambiente foi implementado um protótipo da ferramenta de apoio ao gerenciamento de evolução de esquemas. Por fim, enriquecendo o universo da tese e com o intuito de prover uma maior fundamentação teórica e matemática para descrever as políticas de evolução de esquemas, é especificada uma semântica operacional para um subconjunto do modelo proposto.

O modelo difere de outros modelos de evolução de esquemas por permitir o gerenciamento homogêneo e simultâneo do histórico da evolução do banco de dados intencional e extensional. O modelo de dados adotado é o TVM. O TVM foi selecionado porque traz consigo características de um modelo de versões e de um modelo temporal, permitindo resolver apropriadamente as necessidades requeridas em ambientes com suporte a evolução de esquemas. Por exemplo, o armazenamento do histórico temporal de versões de esquemas. Cabe justificar também a escolha em propor mecanismos de evolução de esquemas considerando modelos orientados a objetos em detrimento a outras abordagens existentes. Por serem semanticamente ricos, esses modelos permitem uma modelagem mais natural e concisa da realidade. Considerando a variedade de tipos existentes, a complexidade sintática e semântica relacionadas as transformações que podem ser aplicadas em bancos de dados, nem todas essas características podem

ser naturalmente modeladas considerando, por exemplo, um modelo relacional. Além disso, conforme tratado em detalhes nas conclusões, a posterior aplicação do modelo proposto nesta tese em outra áreas relacionadas torna-se mais facilitada, permitindo um mapeamento mais direto, sem que exaustivas extensões precisem ser feitas. Por exemplo, utilização do modelo de evolução de esquemas no contexto de evolução de documentos XML ou ainda a aplicação em ambientes de integração de esquemas que exigem o tratamentos da evolução e repercussão de mudanças tanto nos esquemas locais quanto no global.

1.3 Principais Contribuições

Resumidamente, as principais contribuições desta tese são:

- identificação do problema - limitações de bancos de dados orientados a objetos em termos de evolução de esquemas. Em geral, o suporte a evolução de esquemas está limitado a um conjunto pré-definido de operações de modificação, sem que o histórico da evolução seja mantido. Entretanto, essa necessidade é comumente requerida no contexto de banco de dados legados que sofrem alterações freqüentes, requerendo, conseqüentemente, a habilidade de manter diversas versões do esquema juntamente com seus dados associados (mais especificamente destaca-se a necessidade de resolver consultas históricas bem como atualizações envolvendo informações);
- solução - como solução esta tese propõe um modelo que define as estratégias que regem o processo de evolução de esquemas em todos os seus aspectos, tanto na modificação de esquemas como na propagação de mudanças e na manipulação de dados. O diferencial do modelo consiste em tratar homogeneamente os conceitos de tempo e de versão, aplicados na manutenção do histórico da evolução da base de dados intencional e extensional;
- consistência - a consistência do modelo é garantida através de restrições de integridade temporais, estruturais e comportamentais. Essas restrições asseguram que as modificações de esquema preservam a integridade tanto do esquema conceitual quanto da base de dados. Além disso, uma semântica operacional é definida para um subconjunto do modelo, no qual os aspectos essenciais são formalmente tratados, fornecendo uma base teórica e matemática que permite uma comparação segura com outros modelos que abordam o mesmo problema tratado nesta tese;
- implementação - um subconjunto relativo àas principais funcionalidade do TVSE foi implementado em um banco de dados convencional (objeto-relacional). O protótipo serviu de indicativo para comprovar que a solução proposta é possível de ser implementada, gerando os resultados esperados quanto à integração dos conceitos de tempo e versão no banco de dados intencional e extensional.

1.4 Organização do Texto

O restante da tese está organizado em mais 8 capítulos, como segue:

- capítulo 2, Áreas Relacionadas, apresenta o contexto no qual a tese está inserida, identificando as principais necessidades e desafios quanto ao gerenciamento de

evolução de esquemas, com ênfase para a adoção dos conceitos de tempo e versão, finalizando com um comparativo entre as propostas apresentadas;

- capítulo 3, Modelo Temporal de Versões, apresenta o Modelo Temporal de Versões (TVM) e sua linguagem de consulta, bem como uma análise do TVM frente ao gerenciamento de evolução de esquemas, estabelecendo as premissas para a especificação da tese;
- capítulo 4, Arquitetura para o Versionamento Temporal de Esquemas, introduz a abordagem proposta nesta tese e apresenta uma visão geral do processo de evolução de esquemas e atualização de dados em bancos de dados temporais orientados a objetos (BDTOO);
- capítulo 5, Modelo Temporal de Versionamento com Suporte à Evolução de Esquemas, apresenta detalhadamente o modelo de evolução de esquemas proposto. São apresentados a representação temporal e de versões, o gerenciamento da modificação de esquemas, o mecanismo de propagação de mudanças e o armazenamento da extensão;
- capítulo 6, Atualização de Objetos, apresenta a linguagem de atualização de objetos proposta para o modelo de evolução de esquemas, bem como considerações sobre a linguagem de consulta;
- capítulo 7, Semântica Operacional, apresenta a semântica operacional para a linguagem de versionamento e modificação de esquemas e atualização de objetos proposta nesta tese;
- capítulo 8, Ambiente Temporal de Evolução de Esquemas, apresenta um ambiente que presta suporte ao desenvolvimento de aplicações com gerência de versionamento temporal de esquemas para o modelo proposto nesta tese, detalha o mecanismo para o armazenamento físico da base de dados extensional e, por fim, ilustra a ferramenta implementada;
- capítulo 9, Conclusões, apresenta uma revisão da tese, as principais contribuições e aspectos inovadores, e os trabalhos futuros.

A tese também contém 4 anexos, que apresentam respectivamente os seguintes conteúdos:

- Anexo A, BNF da Linguagem de Versionamento de Esquemas;
- Anexo B, BNF da Linguagem de Modificação de Esquemas;
- Anexo C, BNF da Linguagem de Atualização de Objetos;
- Anexo D, Especificação das Funções de Adaptação.

2 ÁREAS RELACIONADAS

Este capítulo apresenta o contexto no qual a tese está inserida: evolução de esquemas conceituais. As áreas de pesquisa são apresentadas, identificando as principais necessidades e desafios quanto ao gerenciamento de evolução de esquemas com características de tempo e de versões, visando a delimitar o escopo do problema tratado pela tese. Dentro desse contexto, são descritos alguns trabalhos relacionados, que são comparados entre si e posteriormente com a solução proposta pela tese.

Este capítulo apresenta de forma resumida os resultados obtidos no exame de qualificação de doutorado (GALANTE, 2001).

2.1 Modificação, Evolução e Versionamento de Esquemas

Em (JENSEN; et al., 1998) foi definido um glossário que apresenta um consenso das definições dos conceitos específicos de bancos de dados temporais. Os conceitos de modificação, evolução e versionamento de esquemas são comumente confundidos, sendo no glossário feita a seguinte distinção:

- modificação de esquema - o SGBD (Sistema de Gerenciamento de Banco de Dados) permite modificações no esquema de uma base de dados populada;
- evolução de esquemas - o SGBD permite modificações no esquema de uma base de dados populada, garantindo a consistência e a integridade dos dados armazenados. Entretanto, a evolução de esquemas não implica um suporte completo à história evolutiva do esquema, mas somente a habilidade para alterar as definições do esquema conceitual sem a perda de informações;
- versionamento de esquema - o SGBD permite a visualização de todos os dados, tanto retrospectiva quanto prospectivamente, por meio de várias versões de esquemas definidas pelo usuário.

O versionamento de esquemas pode ser classificado em duas categorias (RODDICK, 1995):

- versionamento parcial - as requisições de modificação de esquemas são permitidas somente no esquema corrente;
- versionamento total - as requisições de modificação de esquemas são permitidas em qualquer versão de esquema.

2.2 Requisitos para Evolução de Esquemas

No contexto de evolução de esquemas de dados, em (CAMOLESI; TRAINA, 1996) é apresentada uma classificação que pode ser útil para aqueles que se propõem a desenvolver um modelo de dados e/ou um SGBD com capacidade de manutenção da base de dados intencional. Nesse sentido, são consideradas três áreas de abrangência para evolução de esquemas:

- nível de projeto - envolve o conceito de modelo de dados, ou seja, os conceitos sobre esquemas de bancos de dados e a metodologia utilizada para produção de tais esquemas;
- nível físico - relacionado ao problema de evolução de esquemas físico e de suas conseqüências na base de dados, reestruturação e reformatação;
- nível lógico - relacionado ao problema de evolução física e conceitual do esquema externo (as visões dos usuários). Refere-se, especificamente, às operações de alteração do esquema lógico, às técnicas de evolução empregadas e às estratégias de propagação das atualizações na base de dados intencional.

Dentro desse contexto, a complexidade de evolução de esquemas pode ser classificada em cinco níveis, a fim de preservar a consistência do banco de dados:

- representação de esquema - identificação dos aspectos do esquema que devem sofrer modificações;
- operações de modificação de esquema - identificação das operações de modificação permitidas para o esquema;
- consistência na modificação de esquema - maneira como as modificações do esquema são qualificadas e propagadas no banco de dados;
- integridade do banco de dados - identificação da forma como as instâncias vigentes no banco de dados são afetadas frente à modificação do esquema;
- disponibilidade do banco de dados - extensão do banco de dados para permitir a execução das aplicações após a modificação do esquema.

Cada modelo de dados define elementos existentes em seu esquema, um conjunto de operações de evolução de esquemas, e restrições de integridade para a execução das ações evolutivas que geram o novo esquema. Toda ação de alteração de esquema deve passar por filtros de integridade nos quais a consistência do novo esquema deve ser verificada através de um conjunto de restrições para cada uma das alterações permitidas.

O tratamento adequado das alterações realizadas em um esquema durante o seu tempo de vida tem sido alvo de muitas pesquisas e, conseqüentemente, surgiram diversas propostas que visam a tratar adequadamente o assunto. Considerando a temporalidade e o versionamento de esquemas, alguns trabalhos têm explorado esses conceitos no âmbito de sistemas de bancos de dados relacionais (ANGONESE, 2000; MOREIRA; EDELWEISS, 1999a; CASTRO; GRANDI; SCALAS, 1995a, 1997), e outros têm incorporado esses conceitos em bancos de dados orientados a objetos (RODRÍGUEZ; OGATA; YANO, 1999a; GORALWALLA et al., 1997, 1998; LAUTEMANN, 1997a,b; GRANDI; MANDREOLI, 2003; RASHID; SAWYER; PULVERMUELLER, 2000; RASHID, 2001; WEI; ELMASRI, 2000a). Entretanto, a maioria das propostas trata o versionamento de esquemas ou através de características temporais (CASTRO; GRANDI;

SCALAS, 1995a, 1997; GRANDI; MANDREOLI, 2003; MOREIRA; EDELWEISS, 1999a; RODRÍGUEZ; OGATA; YANO, 1999a; GORALWALLA et al., 1997, 1998; WEI; ELMASRI, 2000a) ou pelo controle de versões (LAUTEMANN, 1997a,b; RASHID; SAWYER; PULVERMUELLER, 2000; RASHID, 2001). A principal contribuição desta tese é a especificação de um modelo de evolução de esquemas que integra esses dois conceitos: versão e tempo.

Nas próximas seções, estas propostas são brevemente descritas para cada problema mencionado.

2.3 Evolução de Esquemas e Propagação das Mudanças

O gerenciamento da evolução de esquemas durante a fase operacional é um problema complexo, pois cada mudança deve levar em consideração as instâncias previamente armazenadas. Tais transformações requerem o armazenamento de esquemas antigos para garantir o acesso aos dados associados, levando ao problema de administrar as diferentes versões de esquemas e as relações entre estas versões com seus dados correspondentes.

A inclusão de facilidades de evolução de esquemas em sistemas de bancos de dados envolve a solução de dois problemas fundamentais: (*i*) a semântica de modificação de esquemas, que se refere aos efeitos das modificações no próprio esquema; e (*ii*) a semântica de propagação de mudanças, que descreve a forma como essas modificações nos esquemas afetam as instâncias da base de dados. O primeiro problema envolve a verificação e manutenção da consistência após as modificações no esquema, ao passo que o segundo envolve a consistência dos dados vigentes de acordo com as mudanças realizadas.

A alteração da base de dados em decorrência de uma evolução de esquema pode ser realizada de duas formas distintas (CAMOLESI; TRAINA, 1996): (*i*) *on-line*, no qual o SGBD é mantido em operação enquanto o processo de evolução de esquemas é realizado; ou (*ii*) *off-line*, no qual todos os processos em andamento são interrompidos enquanto a base de dados está sendo reorganizada, devendo ser executada em um período de pouca requisição de dados.

A seguir, os trabalhos relacionados ao processo de modificação de esquemas e propagação de mudanças são apresentados e comparados entre si.

2.3.1 Trabalhos Relacionados

O suporte à evolução e versionamento de esquemas tem sido amplamente estudado tanto no campo de bancos de dados relacionais quanto em bancos de dados orientados a objetos. Em (RODDICK, 1995) é apresentada uma avaliação dos principais tópicos envolvidos no gerenciamento de evolução e versionamento de esquemas.

Historicamente, existem três técnicas fundamentais empregadas para modificar a estrutura do esquema em bancos de dados orientados a objetos. Primeira, evolução de esquemas (BANERJEE et al., 1987), na qual o banco de dados possui um esquema lógico e operações de modificação são aplicadas nas definições das classes e na hierarquia de especialização/generalização. Segunda, versionamento de esquemas (LAUTEMANN, 1996, 1997b), no qual várias versões do esquema lógico podem ser criadas e manipuladas independentemente. Terceira, versionamento de classes (MONK; SOMMERVILLE, 1993; CLAMEN, 1991; SKARRA; ZDONIK, 1986), que corresponde à criação de novas versões de classe a cada modificação realizada. Nesse último, as versões antigas são sempre mantidas, sendo que diversas versões de uma mesma classe podem existir

simultaneamente no banco de dados.

O sistema de banco de dados COAST (*Complex Object And Schema Transformation*) (LAUTEMANN et al., 1999) define um mecanismo para gerenciamento da evolução de esquemas em bancos de dados orientados a objetos (LAUTEMANN, 1997b). O principal objetivo é manter as aplicações sempre em execução, garantindo a integridade do esquema, através da consistência estrutural (integridade das classes e relacionamentos de agregação/generalização) e comportamental (validade do conjunto de métodos). Versões de esquemas são utilizadas para manter o histórico das modificações; entretanto, nem todas as modificações no esquema ocasionam a derivação de versões. Em (BUDDRUS; GÄRTNER; LAUTEMANN, 1997) é apresentada a base formal para as primitivas de modificação de esquemas. As primitivas que não afetam as aplicações, denominadas primitivas de extensão, não geram novas versões, pois somente introduzem novas informações no esquema. Por outro lado, as primitivas que têm impacto sobre as aplicações, denominadas primitivas de modificação, acarretam a derivação de novas versões. A vantagem está em manter as aplicações sempre ativas e controlar a derivação excessiva de versões. Entretanto, esse controle no número de versões geradas limita a manutenção do histórico completo das atualizações, já que nem todas as modificações são retidas.

DBEvolution (BENATALLAH, 1999) é um *framework* implementado através de um método híbrido para gerenciamento da evolução de esquemas, combinando as técnicas de modificação e de versionamento. As operações de atualização são aplicadas sobre a versão corrente do esquema. Sendo aceita, a operação é traduzida em modificação, podendo também provocar a derivação de uma nova versão para o esquema. A modificação física do esquema, ou seja, quando a alteração substitui o antigo esquema, é realizada somente quando solicitada pelo usuário; caso contrário, o histórico da evolução é mantido através da derivação de versões de esquemas e classes.

Farandole 2 (AL-JADIR; LÉONARD, 1998) define um mecanismo baseado em versões para gerenciar a evolução de esquemas. Baseia-se na idéia de contexto, podendo ser considerada como uma extensão ao conceito de visões. Um conjunto de autorizações de mudança é definido, como também regras que garantem a coerência do esquema frente a essas alterações, associando cada versão aos dados armazenados, integrando assim o modelo. O conceito de contexto, as alterações elementares do banco de dados e uma taxonomia de mudanças de esquema estão implementados em (AL-JADIR et al., 1995). Além da realização de modificações de esquema e representação coerente da realidade trabalhada, esse mecanismo apresenta um relativo aumento de desempenho, à medida que os objetos não são copiados, nem propagados na presença de modificações de esquemas, apenas multi-objetos são definidos, evitando a necessidade de definição de regras complexas de propagação.

Sades Evolution (*Semi-Autonomous Database Evolution System*) (RASHID, 2001) apresenta um estudo sobre evolução de esquemas do ponto de vista de linguagens de programação. O mecanismo de evolução de esquemas está centrado nas modificações realizadas nas definições das classes, para as quais a técnica de versionamento de classes é utilizada, gerando diferentes visões para o esquema conceitual.

O trabalho desenvolvido pelo grupo conduzido por Elke Rundensteiner merece uma atenção diferenciada, pois ela tem sido, nos últimos anos, uma das pesquisadoras mais ativas no contexto de evolução e versionamento de esquemas. O objetivo principal do trabalho desenvolvido pelo grupo é investigar a tecnologia de evolução transparente de esquemas, permitindo modificações na estrutura do banco de dados

sem interromper as aplicações existentes (RA; RUNDENSTEINER, 1997). Dentre os trabalhos desenvolvidos, destacam-se:

- (i) SERF (CLAYPOOL; RUNDENSTEINER, 1999; CLAYPOOL; JIN; RUNDENSTEINER, 1998) - *Schema Evolution through an Extensible, Re-usable and Flexible Framework* - é um *framework* baseado na premissa de que diferentes aplicações possuem requisitos peculiares e diferenciados. O mecanismo de gerência de evolução de esquemas deve evitar a construção de código *ad-hoc* para cada aplicação em particular. A principal estratégia é integrar o conjunto de primitivas de modificação de esquemas pré-definidas pelo grupo ODMG com a linguagem OQL para a realização do processo migração de objetos. SERF provê flexibilidade para o usuário definir a semântica de mudança, extensibilidade para definir operações de modificação compostas com base na taxonomia pré-definida, bem como reusabilidade dessas operações através de *templates*. O núcleo do SERF foi demonstrado em (RUNDENSTEINER et al., 2000);
- OQL_SERF (RUNDENSTEINER et al., 1999) - é a implementação do *framework* SERF, que utiliza como base os conceitos definidos pelo padrão ODMG. O protótipo está baseado no modelo de objetos (ODMG *Object Model*). A linguagem de consulta OQL é utilizada como linguagem de transformação do banco de dados (OQLas) (SU; CLAYPOOL; RUNDENSTEINER, 2000) e os metadados (ODMG *Metadata Repository*) mantêm as informações necessárias ao gerenciamento da evolução de esquemas;
- ROVER (CLAYPOOL; RUNDENSTEINER; HEINEMAN, 2000, 2001) - é um *framework* para gerência de evolução de relacionamentos que especifica um conjunto pré-definido de operações. Através do SERF o usuário pode construir novas operações compostas. A abordagem incorpora também a noção de contrato como alternativa para a especificação de restrições de integridade para as primitivas de evolução de esquemas, com o intuito de compensar os custos de reengenharia.

Tem sido desenvolvido ainda um método de otimização de operações de evolução de esquemas em um ambiente integrado. O método é denominado CHOP (CLAYPOOL; NATARAJAN; RUNDENSTEINER, 1999, 2000), e está baseado na taxonomia de operações de modificação de esquemas proposta pelo grupo ODMG. As principais contribuições são: (*i*) especificação de um otimizador baseado em heurísticas que reduz a seqüência de operações de modificação de esquemas; (*ii*) provas de correção que garantem que o otimizador produza os mesmos resultados da execução original de cada modificação, garantindo também que cada seqüência gerada seja mínima e única; (*iii*) implementação de um protótipo baseado no padrão ODMG; e (*iv*) experimentação e validação que comprovam o desempenho eficiente do otimizador.

2.3.1.1 Evolução de Esquemas

No contexto de bancos de dados temporais, três modelos formais especificam mecanismos para gerenciar o versionamento de esquemas em bancos de dados orientados a objetos.

No primeiro modelo (GORALWALLA et al., 1997, 1998), é proposto um método para gerenciamento de evolução de esquemas baseado no modelo TIGUKAT (PETERS, 1994; ÖZSU et al., 1995). TIGUKAT é um modelo de objetos, com características temporais, que inclui um modelo de consulta, políticas para evolução de esquemas e

controle de versões. O tempo é utilizado como base para gerenciar as mudanças do esquema e manter o histórico das versões dos objetos. A granularidade do versionamento contempla esquemas, tipos e objetos. Além disso, em (PETERS; ÖZSU, 1997, 1995) é proposto um modelo axiomático, que formaliza a evolução dinâmica de esquemas, permitindo o gerenciamento de mudanças em sistemas em operação. As primitivas de modificação de esquemas, aliadas aos axiomas, garantem a consistência do esquema, sem que haja necessidade de executar uma verificação explícita a cada modificação, pois versões inconsistentes nunca são geradas.

TVOO (*Temporal Versioned Object Oriented*) (RODRÍGUEZ; OGATA; YANO, 1999a) é o segundo modelo formal para evolução de esquemas que combina os conceitos de tempo e versão com a tecnologia de orientação a objetos. A principal característica é que novas definições para os esquemas não geram novas versões da base de dados. As definições antigas do esquema são consideradas alternativas de projeto e, conseqüentemente, o acesso aos dados existentes é feito de forma consistente sem perda de informação.

O terceiro modelo, OODM$|_{SV}$ (GRANDI; MANDREOLI, 2003), apresenta uma descrição formal para o versionamento temporal de esquemas, incluindo um conjunto completo de primitivas de modificação de esquemas compatíveis com aquelas definidas no padrão ODMG (GRANDI; MANDREOLI, 1999). A formalização do conceito de versão de esquema e do inter-relacionamento entre essas versões é realizada através de lógica de descrição (FRANCONI; GRANDI; MANDREOLI, 2000a,b), que permite, através de mecanismos de inferência, executar dois tipos de verificação: (i) consistência local, que se refere a uma versão de esquema; e (ii) consistência global, que se refere ao banco de dados como um todo.

2.3.1.2 Propagação de Mudanças

Para o problema de propagação de mudanças, várias soluções têm sido propostas e implementadas em sistemas reais. Quatro abordagens são identificadas:

- conversão imediata (coerção) - a propagação é realizada exatamente após a modificação de esquema. Essa abordagem está implementada, por exemplo, no sistema GemStone (PENNEY; STEIN, 1987);
- conversão adiada - a propagação é feita logicamente de acordo com o novo esquema. Entretanto, a reestruturação física é realizada somente no instante da efetiva utilização dos objetos. Essa abordagem está implementada, por exemplo, no sistema ORION (BANERJEE et al., 1987);
- emulação - a propagação é prorrogada indefinidamente até o surgimento de um novo esquema. Nenhuma alteração física é realizada nas instâncias, ou seja, o sistema realiza uma adaptação lógica da informação de acordo com o novo esquema, agindo como um filtro para criar a ilusão de instâncias modificadas. Essa abordagem está implementada, por exemplo, no sistema CLOSQL (MONK; SOMMERVILLE, 1993);
- método híbrido - a propagação combina duas ou mais das abordagens apresentadas nos itens anteriores. Essa abordagem está implementada, por exemplo, nos sistemas Sherpa (NGUYEN; RIEU, 1989) e O_2 (FERRANDINA et al., 1995).

Em qualquer uma das abordagens, o mecanismo de evolução de esquemas deve prover, por *default*, regras automáticas para a transformação da base de dados. Na

inexistência de tais mecanismos, o projetista tem a possibilidade de implementar aplicações, denominadas funções de conversão, que permitem a reestruturação da base de dados em conformidade com as alterações realizadas no esquema.

O sistema COAST utiliza funções de conversão para atualizar a base de dados em decorrência de uma modificação de esquema (LAUTEMANN, 1997a, 1999). As funções de conversão estabelecem como parâmetros de entrada as classes definidas no antigo esquema, transformando os objetos da base de dados de acordo com a nova estrutura definida para o esquema. Quanto às estratégias de conversão, o usuário pode optar entre a conversão imediata ou adiada. No primeiro caso, a execução das aplicações é suspendida até o término da adaptação. No segundo caso, as aplicações nunca são interrompidas completamente. Durante o processo de conversão, o estado antigo do esquema é mantido pela derivação de versões de esquemas. Cada aplicação trabalha sempre com uma versão de esquema que, por sua vez, está associada a um conjunto de objetos, denominado escopo de acesso às instâncias. Somente um escopo de acesso às instâncias é visível para cada versão de esquema.

No *DBEvolution*, o processo de propagação de mudanças (BENATALLAH; TARI, 1998) é realizado imediatamente após a modificação de esquemas, podendo ser realizado por modificação, versionamento e/ou emulação. A característica singular dessa proposta é a introdução do nível de pertinência de classe. O nível de pertinência caracteriza-se pela importância da disponibilidade das instâncias da classe com relação aos programas aplicativos. Dois estados são definidos: (*i*) pertinente - quando a disponibilidade da instância é considerada alta; e (*ii*) obsoleto - em situação inversa. Objetos considerados pertinentes são convertidos de acordo com as novas definições do esquema e armazenados na base de dados. Objetos classificados como obsoletos permanecem vigentes (por emulação) somente durante a execução das aplicações, não sendo armazenados fisicamente na base de dados.

Sades Evolution (RASHID; SAWYER; PULVERMUELLER, 2000) propõe um método flexível para adaptação das instâncias durante o versionamento de classes. O diferencial é que o código para adaptação das instâncias permanece separado das versões de classe, ao contrário das implementações realizadas pelas linguagens de programação. As rotinas para adaptação das instâncias são implementadas para cada operação de modificação, e não para cada versão de classe. Isso possibilita a utilização de estratégias alternativas de adaptação co-existindo em um mesmo sistema.

No contexto do sistema TIGUKAT, o trabalho (PETERS; BARKER, 2000) introduz um modelo axiomático para propagação de mudanças que é capaz de identificar, de maneira declarativa, o conjunto de objetos afetado pelas modificações de esquemas. Este mecanismo permite, por exemplo, utilizar o conjunto de objetos como entrada para qualquer uma das abordagens de propagação de mudanças previamente citadas. A propagação das alterações é realizada através da técnica de emulação. Assim, os objetos podem estar relacionados com mais de uma versão de esquema.

Em (RODRÍGUEZ; OGATA; YANO, 1999b) o modelo TVOO apresenta um mecanismo para acessar objetos temporais versionados em bancos de dados com suporte ao versionamento de esquemas. Os objetos e classes não são fisicamente descartados do banco de dados após serem modificados, sendo assim retido o histórico das modificações. O principal diferencial é que os objetos permanecem dependentes das mudanças formando uma hierarquia de versões. Entretanto, a única dimensão temporal suportada é o tempo de transação.

O modelo OODM$|_{SV}$ (GRANDI; MANDREOLI, 2003) provê um mecanismo de tradução dos dados existentes de uma versão de esquema para outra, seguindo o método de gerenciamento síncrono. Esse mecanismo é utilizado também para propagar as mudanças de esquemas nos dados existentes, preservando sempre a integridade global do banco de dados. O mecanismo de propagação utiliza a técnica de coerção e está baseado na inclusão de valores nulos nos objetos para os novos atributos incluídos.

Ao invés de confiar em um mecanismo automático de reorganização da base de dados após as mudanças de esquema, em (LAGORCE; STOCKUS; WALLER, 1997) é apresentada uma linguagem de programação declarativa para a atualização das instâncias. Essa linguagem permite a escrita de programas que, simultaneamente, atualizam tanto o esquema quanto as instâncias, garantindo um estado consistente ao banco de dados, devido à realização de verificação estática de consistência perante a execução desses programas.

Uma abordagem completamente diferente é apresentada em (LERNER, 2000), no qual algoritmos são especificados para analisar modificações realizadas em tipos complexos. A análise é sempre realizada através da comparação entre duas versões de esquemas. Regras de derivação são especificadas para a propagação das mudanças nos dados existentes.

Considerando somente o banco de dados extensional na presença de um esquema fixo, o mecanismo de multiversões (MEDEIROS; BELLOSTA; JOMIER, 2000) permite manipular múltiplas versões de objetos através de visões, combinando propriedades de versões e mecanismos de visões. A principal contribuição é tratar de forma integrada esses dois conceitos, uma vez que são tratados separadamente pela comunidade de banco de dados. Além disso, em (GANÇARSKI; JOMIER, 2001) é proposto um *framework* para o desenvolvimento de aplicações de projeto e gerenciamento de configurações, considerando, em especial, o tratamento de versões de objetos complexos.

2.3.1.3 Comparação entre os trabalhos relacionados

Sintetizando, as Tabelas 2.1 e 2.2 comparam os trabalhos relacionados, considerando o processo de modificação de esquemas e propagação de mudanças.

Tabela 2.1: Comparativo de trabalhos relacionados: modificação de esquemas e propagação de mudanças

Itens / Propostas	**COAST**	***DBEvolution***	**Farandole 2**	**Sades**	**TIGUKAT**	**TVOO**
Modelo de Dados	OO	OO	OO	OO	temporal OO	temporal OO
Granularidade do versionamento	esquemas	esquemas, classes e objetos	esquemas	classes	esquemas, tipos e objetos	esquemas, classes e objetos
Taxonomia de mudanças	$\surd$	$\surd$	$\surd$	$--$	$\surd$	$\surd$
Restrições para modificação	$\surd$	$\surd$	$\surd$	$--$	$\surd$	$\surd$
Propagação de Mudanças	implícita	implícita	implícita, explícita	implícita, explícita	implícita	implícita, explícita
Método de Propagação	conversão imediata e adiada	híbrido	multi-objetos	conversão adiada	conversão adiada	conversão imediata e adiada
Compatibilidade com ODMG	$--$	$--$	$--$	$--$	$\surd$	$--$

Legenda: $\surd$ Possui $\times$ Não possui $--$ Não conhecido

Com base nessas tabelas, dois pontos importantes devem ser salientados. O primeiro é que grande parte dos trabalhos aborda alguns aspectos específicos de evolução de esquemas e não o processo como um todo. Nesse sentido, são trabalhos mais robustos,

Tabela 2.2: Comparativo de trabalhos relacionados: modificação de esquemas e propagação de mudanças (cont.)

Itens / Propostas	**OODM**$\vert_{SV}$	**Lagorce**	**Lerner**	***Multiview***	**SERF**
Modelo de Dados	temporal OO	OO	OO	OO	OO
Granularidade do versionamento	esquemas	×	tipos	objetos	×
Taxonomia de mudanças	√	−−	√	×	√
Restrições para modificação	√	√	√	×	√
Propagação de Mudanças	implícita, explícita	implícita	implícita	×	implícita
Método de Propagação	conversão imediata	conversão imediata	conversão imediata e adiada	×	conversão imediata
Compatibilidade com ODMG	√	−−	−−	−−	√

Legenda: √ Possui × Não possui −− Não conhecido

pois propõem soluções para as semânticas de mudança e de propagação. O segundo é que todas as abordagens que utilizam o gerenciamento de versões para o controle da evolução de esquemas adotam modelos independentes para esquemas (intenção) e dados (extensão), mas nunca em conjunto. O trabalho apresentado nesta tese tem seu diferencial na incorporação de um modelo temporal de versões tanto no nível de esquemas quanto no nível de objetos. Mecanismo de versões e tempo são combinados a fim de controlar não somente o versionamento de esquemas, mas também o armazenamento das instâncias e a propagação de mudanças. O suporte à evolução de esquemas, com a utilização dos conceitos de versão e tempo, oferece uma solução inteligente para o problema de evolução de esquemas, facilitando o manuseio de qualquer alteração entre dados e estrutura, pois a definição do esquema é retida após ter sido alterada ou corrigida.

Por fim, bancos de dados comerciais também tratam a questão de evolução de esquemas e versionamento de esquemas, classes e objetos, porém de forma menos detalhada que as abordagens propostas. A Tabela 2.3 apresenta um resumo comparativo dos principais bancos de dados comerciais, estando limitados aos orientados a objetos, que é o foco de estudo desta tese. Os bancos de dados comparados são: Jasmine (COMPUTER ASSOCIATES INTERNATIONAL, Inc., 1998); POET (POET SOFTWARE, 1997); Versant (VERSANT OBJECT TECNOLOGY, 1997); O2 (ARDENT SOFTWARE, 1998); GemStone (GEMSTONE SYSTEMS, INC., 2003); Itasca (ITASCA DISTRIBUTED OBJECT DATABASE MANAGEMENT SYSTEM, 1995); ObjectStore (OBJECT DESIGN INC., 2003); e ObjectivityDB (OBJECTIVITY INC., 2001a,b).

Tabela 2.3: Comparativo de trabalhos relacionados: SGBDs comerciais

Itens / SGBDs	**Jasmine**	**POET**	**Versant**	**O2**	**GemStone**	**Itasca**	**ObjectStore**	**Objectivity/DB**
Compatibilidade com ODMG	√	√	√	√	√	−−	√	√
Alterações em classes	√	√	√	√	√	√	√	√
Alterações em métodos	√	×	×	√	√	√	√	−−
Alterações em atributos	√	√	√	√	√	√	√	√
Versionamento de esquemas	×	×	×	×	×	×	√	×
Versionamento de classes	×	√	×	×	√	×	×	√
Versionamento de objetos	×	×	√	√	−−	√	×	√

Legenda: √ Possui × Não possui −− Não conhecido

2.4 Armazenamento e Gerenciamento da Extensão e Intenção

Na presença de versionamento de esquemas, a base de dados extensional deve estar armazenada de forma a garantir flexibilidade no acesso e na atualização. Em (CASTRO; GRANDI; SCALAS, 1995b,a, 1997) são propostas duas soluções para o armazenamento da base de dados extensional:

- repositório único - os dados correspondentes a todas as versões de esquema são mantidos em um único repositório, com um esquema global, incluindo todas as propriedades introduzidas pelas sucessivas mudanças de esquema;
- múltiplos repositórios - distintos repositórios são mantidos para cada versão de esquema. Quando uma nova versão é criada, os dados são copiados para um novo repositório e adaptados de acordo com o esquema correspondente.

Em (CASTRO; GRANDI; SCALAS, 1997) são definidas também duas estratégias para gerenciar o sincronismo entre as versões de esquemas e os dados armazenados:

- gerenciamento síncrono - a pertinência temporal dos dados deve estar contida na pertinência temporal da versão de esquema correspondente, tornando-os dependentes. Nesse caso, os dados devem ser armazenados, recuperados e atualizados sempre com a versão de esquema que tenha mesma pertinência temporal;
- gerenciamento assíncrono - a pertinência temporal entre as versões de esquema e os dados armazenados é completamente independente. Nesse caso, os dados podem ser recuperados e atualizados através de qualquer versão de esquema.

A seguir, os trabalhos relacionados ao processo de armazenamento e gerenciamento da intenção e extensão são apresentados e comparados entre si.

2.4.1 Trabalhos Relacionados

Duas propostas foram precursoras nas pesquisas considerando conversão e armazenamento da base de dados frente às mudanças de esquemas em bancos de dados temporais relacionais: STV (*Single Table Version*) e MTV (*Multi Table Version*). Na abordagem STV há somente uma versão da relação em todo o tempo de vida do banco de dados. A proposta utiliza o conceito de esquema completado (RODDICK, 1991), o qual inclui todos os atributos introduzidos pelas sucessivas mudanças do esquema. Nos casos em que a mudança é destrutiva, como, por exemplo, na exclusão de um atributo ou restrição de um domínio, ela não pode ser executada fisicamente, devendo apenas ser gravada em um catálogo. Se a mudança, ao contrário, propuser a adição de um atributo ou a extensão de um domínio, o atributo deve ser duplicado e convertido para o novo formato. Por sua vez, a abordagem MTV (CASTRO; GRANDI; SCALAS, 1997) requer a criação de tantos repositórios quantas forem as mudanças nas relações (versões de esquema). Cada repositório é formatado de acordo com a versão do esquema correspondente. Quando um novo repositório é inicializado, as tuplas são copiadas do repositório antigo de acordo com as mudanças aplicadas no esquema.

Por outro lado, o PMTV (*Partial Multiple Table Versioning*) (WEI; ELMASRI, 1999, 2000a,b) propõe um método híbrido para o armazenamento da atualização de esquemas em bancos de dados bitemporais relacionais. Para uma modificação de esquema, como, por exemplo, a inclusão de um atributo, é criada uma relação bitemporal, contendo

somente o novo atributo e o identificador da relação que está sendo modificada. Esta técnica é similar à fragmentação vertical (ÖZSU; VALDURIEZ, 1999) em sistemas de bancos de dados distribuídos. A relação completa pode ser reconstruída com a operação *Entity-Join* (TANSEL et al., 1993). As principais vantagens são a diminuição do espaço requerido para as novas versões de objetos e agilidade na recuperação dos objetos antes da conversão da base de dados. O ponto crítico é o processamento de consulta, pois envolve operações complexas de *join* durante a otimização. Entretanto, esta questão tem sido resolvida através da proposta de uma estrutura de indexação.

No grupo de pesquisa *Versions and Time in Databases* (MORO et al., 2003), no qual esta tese está inserida, três trabalhos de implementação foram desenvolvidos com o intuito de investigar os tópicos envolvendo evolução de esquemas e simular a viabilidade de implementação das propostas. O primeiro trabalho, proposto por (ANGONESE, 2000), apresenta um mecanismo que integra gerenciamento de versões do esquema com o controle temporal dos dados da aplicação. O modelo de dados utilizado é o TempER (ANTUNES, 1997), no qual são apresentados os mapeamentos necessários para a implementação das características temporais. Como resultado, é implementado um Gerenciador Temporal de Versões de Esquemas, que mapeia, sobre um banco de dados convencional, os principais conceitos relacionados ao versionamento de esquemas: tempo de transação para a intenção, bitemporal para extensão, gerenciamento síncrono e armazenamento da base de dados em múltiplos repositórios. O segundo trabalho, proposto por (SANTOS, 2003), apresenta um estudo comparativo sobre versionamento de esquemas envolvendo duas alternativas para o armazenamento da extensão: repositório único e múltiplos repositórios. O objetivo principal é avaliar os resultados obtidos em cada abordagem, considerando o espaço de armazenamento, o tempo para atualização dos dados e o tempo de resposta às consultas. Com relação à implementação, o modelo temporal TRM (*Temporal Relational Model*) (NAVATHE; AHMED, 1989) é mapeado para o DB2, sendo especificados os requisitos para a criação de um ambiente temporal usando versões de esquemas. O versionamento da intenção é mantido através de tempo de transação, os dados da extensão são armazenados em um banco de dados de tempo de validade e o gerenciamento entre a intenção e extensão é síncrono. Como resultado, verifica-se que a utilização da abordagem repositório único é vantajosa, considerando o desempenho na manipulação de dados (por exemplo, operações de inserção, exclusão e atualização) e o espaço de armazenamento da intenção e extensão; porém, a base de dados permanece indisponível durante a propagação das mudanças. O terceiro trabalho, TVMSE (*Temporal Versions Model Schema Evolution*) (JANTSCH, 2003), é um protótipo que simula um ambiente de versionamento de esquemas, no qual a intenção é implementada através de tempo de transação, a extensão é bitemporal (agregando as principais características do TVM, utilizado nessa tese) e o gerenciamento é síncrono. Como contribuição, o TVMSE agrega estratégias de consultas temporais tanto na intenção quando na extensão. Para as consultas à intenção é utilizada a linguagem MSQL (GRANDI, 2002) e para extensão é utilizada a linguagem TVQL (MORO et al., 2002), com o acréscimo de uma cláusula que permite selecionar o repositório desejado.

Sintetizando, a Tabela 2.4 compara os trabalhos relacionados com relação ao processo de armazenamento e gerenciamento da intenção e extensão.

Com base nessa tabela, dois pontos importantes devem ser salientados. O primeiro é com relação ao gerenciamento temporal da intenção. A maioria das propostas utiliza o tempo de transação para o gerenciamento das versões de esquema. Esse gerenciamento limita a utilização das versões de esquema à medida que apenas a última versão

Tabela 2.4: Comparativo de trabalhos relacionados: gerenciamento intenção e extensão

Itens / Propostas	**STV**	**MTV**	**PMTV**	**Angonese**	**Santos**	**TVMSE**
Modelo de Dados	relacional	relacional	relacional	TempER	TRM	TVM
Gerenciamento da Intenção	TT, TV ou bitemporal	TT, TV ou bitemporal	bitemporal	TT	TT	TT
Gerenciamento da Extensão	TT, TV ou bitemporal	TT, TV e bitemporal	bitemporal	bitemporal	TV	bitemporal
Armazenamento da Extensão	mono	multi	mono e multi	multi	mono e multi	multi
Sincronismo entre Intenção e Extensão	síncrono e assíncrono	síncrono e assíncrono	síncrono e assíncrono	síncrono	síncrono	síncrono

criada pode ser manipulada. Embora STV e MTV permitam todas as representações temporais (TT, TV e bitemporal), eles não tratam os problemas relacionados à criação de versões e à conversão da base de dados quando o gerenciamento da intenção é bitemporal. Nesse sentido, o PMTV é superior porque utiliza representação bitemporal, permitindo modificações retro e pró-ativas entre as versões de esquemas. Além disso, o gerenciamento das mudanças através da criação de uma nova relação bitemporal elimina o problema de inserção de valores nulos, não havendo necessidade de reestruturação do banco de dados como no STV, nem duplicação de dados como no MTV. Outro ponto a salientar é que todas as propostas estão voltadas para bancos de dados relacionais, ao passo que o gerenciamento da modificação de esquemas é uma necessidade evidente em modelos orientados a objetos, pois freqüentemente representam projetos em evolução, que requerem a manutenção dos diversos estágios do banco de dados e/ou armazenamento de diversas alternativas.

2.5 Gerenciamento de Dados

O gerenciamento da modificação do esquema conceitual e físico envolve operações sobre aspectos estruturais e comportamentais dos elementos do esquema. Essas atualizações devem ser tratadas como transações normais, realizadas enquanto o banco de dados permanece em execução. Além disso, tais transações podem envolver operações de manipulação de dados, como, por exemplo, o mecanismo de propagação de mudanças nos dados. Assim, há um relacionamento direto entre evolução dinâmica de esquema e gerenciamento de transações.

Como transações são construtores que alteram o estado do banco de dados, seus efeitos precisam ser claramente especificados. Assim, um modelo de transações deve permitir a execução concorrente de operações de modificação de esquemas e manipulação de dados. Como as operações podem estar intercaladas, o mecanismo deve prover estratégias para garantir a integridade da pertinência temporal entre intenção e extensão.

A seguir, os trabalhos relacionados ao mecanismo de manipulação de dados durante o processo de evolução de esquemas são apresentados e comparados entre si.

2.5.1 Trabalhos Relacionados

Considerando bancos de dados relacionais, TSQL2 (SNODGRASS; et al., 1994a,b; SNODGRASS, 1995) é a linguagem de consulta temporal que atingiu maior popularidade. TSQL2 é uma extensão da linguagem SQL, tendo sido desenvolvida por um grupo composto por acadêmicos e especialistas em bancos de dados temporais. Dentre suas funcionalidades está o suporte ao versionamento de esquemas de tempo de

transação (única dimensão temporal suportada), de acordo com o modelo apresentado em (RODDICK, 1994). TSQL2 utiliza o conceito de esquema completado, no qual o esquema é formado por relações contendo a união de todos os atributos já definidos para elas. TSQL2 oferece ainda comandos para efetuar transações sobre o esquema com dois objetivos: (*i*) separar as transações sobre dados das transações sobre esquemas, permitindo que as operações sobre os dados da intenção possam ser testadas antes de serem efetivadas; e (*ii*) possibilitar operações compostas de modificação, evitando a proliferação de versões e fazendo com que várias operações sejam realizadas de forma atômica, gerando novas versões somente a partir de mudanças significativas.

A linguagem TSQL2 é estendida em (CASTRO; GRANDI; SCALAS, 1995a, 1997) a fim de suportar o versionamento de tempo de validade e bitemporal. A principal extensão diz respeito ao comando `SET SCHEMA`, que altera o estado do banco de dados. A cláusula `DATE` é acompanhada por `VALID` e/ou `TRANSACTION`, como, por exemplo: (i) `SET SCHEMA VALID <data>` - para o versionamento de tempo de validade; (ii) `SET SCHEMA TRANSACTION <data>` - para o versionamento de tempo de transação; e (iii) `SET SCHEMA VALID <data> AND TRANSACTION <data>` - para o versionamento bitemporal. Os valores temporais selecionados como correntes são usados por todos os comandos em TSQL2 subseqüentes até que um novo comando `SET SCHEMA` seja executado ou seja alcançado o fim da transação. Porém, somente uma única versão de esquema pode ser selecionada por vez.

Outras abordagens tratam exclusivamente do processamento de consulta com suporte ao versionamento de esquemas. Um mecanismo para a resolução de consultas que envolvem múltiplas versões do esquema conceitual é proposto em (MOREIRA; EDELWEISS, 1999b,a; MOREIRA, 1999). O mecanismo proposto suporta qualquer tipo de banco de dados temporal e está de acordo com os princípios da linguagem TSQL2. Além disso, uma segunda extensão à linguagem TSQL2 é proposta nesse trabalho para dar suporte a consultas sobre os dados da intenção. A extensão inclui os seguintes comandos: (*i*) `DESCRIBE SCHEMA <complemento>` - que mostra a estrutura do esquema; e (*ii*) `DESCRIBE SCHEMA CHANGES <complemento>` - que mostra as modificações sofridas pelo esquema. Com relação às consultas aos dados da extensão, a cláusula `using all versions` permite ao usuário informar quando todas as versões aplicáveis devem aparecer no resultado.

MSQL (GRANDI, 2002) - *Multi-Schema Query Language* - é uma linguagem de consulta que explora as potencialidades do versionamento de esquemas em modelos de dados relacionais. A linguagem propõe um modelo lógico de armazenamento, que tem como base a abordagem de múltiplos repositórios (CASTRO; GRANDI; SCALAS, 1997). A principal contribuição da linguagem é permitir aos usuários finais e aos desenvolvedores de aplicação expressar consultas envolvendo diferentes versões de esquemas, cujo resultado pode conter a combinação de dados provenientes de repositórios distintos. MSQL está implementada em um banco de dados relacional, no qual todas as consultas são mapeadas para um conjunto de comandos SQL padrão, mantendo a compatibilidade com as atuais aplicações de banco de dados.

No campo de bancos de dados orientados a objetos, o modelo TIGUKAT (PETERS, 1994; ÖZSU et al., 1995) trata todo sistema como entidades, incluindo o esquema (meta-objetos), as consultas e as instâncias do banco de dados. As transações são tratadas como construtores que mudam o estado do banco de dados quando aplicados a estas entidades. Ao invés da utilização de técnicas de bloqueio (*lock*), algoritmos são especificados para gerenciar o controle de concorrência (para as operações de modificação

no esquema e atualização dos dados), garantindo a correção das operações e a integridade do banco de dados. TQL (*TIGUKAT Query Language*) (PETERS et al., 1993) é a linguagem de consulta proposta para o modelo TIGUKAT. TQL é baseada no paradigma SQL cuja semântica é definida em termos de cálculo de objetos. A principal característica da linguagem TQL é o acesso temporal reflexivo que permite tratar uniformemente esquemas e objetos, recuperando todo o histórico da evolução ao longo do tempo.

O modelo OODM$|_{SV}$ (GRANDI; MANDREOLI, 2003) apresenta uma extensão em (GRANDI; MANDREOLI, 1999) para a linguagem de definição (ODL) e manipulação de objetos (OQL), propostas pelo grupo ODMG. As operações de modificação de esquemas e manipulação de dados são intercaladas entre as cláusulas `BEGIN TRANSACTION` e `COMMIT` ou `ROLLBACK`. A semântica do *now* é especificada, de forma que uma modificação, aplicada na versão corrente do esquema, leva à derivação de uma nova versão, na qual o tempo de finalização da transação é definido pela execução da operação `COMMIT`. Neste caso, o diferencial da proposta está no fato que o tempo de transação da operação pode ser o tempo da nova versão criada e não o tempo da versão corrente. Quando uma operação de modificação de esquema é intercalada com uma operação de manipulação de dados, a execução da transação é realizada em um banco de dados intermediário. Ao final da transação, o sistema verifica o resultado obtido. No caso de inconsistência, a transação é abortada, garantindo assim a integridade do banco de dados durante o processamento das transações.

Farandole 2 (AL-JADIR; LÉONARD, 1998), através do mecanismo de multi-objetos, suporta múltipla instanciação, migração e classificação automática de objetos, considerando as modificações realizadas no esquema. Versões dos objetos (denominadas *Multiobjects*) podem ser criadas, modificadas e excluídas de acordo com as modificações executadas no esquema, reduzindo o tempo de propagação de mudanças e agilizando o processamento de consultas.

Em (RIEDEL, 1999) é proposta uma estratégia que estende o padrão ODMG com versões temporais de classes e de esquemas. As consultas são formuladas utilizando plenamente as características da linguagem OQL proposta pelo grupo ODMG. O acesso aos dados da extensão pode ser realizado especificando uma determinada versão (de esquema ou de classe) ou através da utilização do operador `NE` (*null-extend*). O operador `NE` transforma um objeto ou literal em um subtipo dentre aqueles especificados para os valores nulos. A principal contribuição dessa estratégia é o tratamento da manipulação de valores nulos no resultado das consultas que envolvem mais de uma versão de esquema.

Sintetizando, a Tabela 2.5 compara os trabalhos relacionados, considerando a manipulação de dados durante o processo de evolução de esquemas.

Conforme ilustrado nessa tabela, a maioria das propostas realiza a manipulação de dados em uma versão de esquema previamente selecionada, negligenciando o fato de que o processo de evolução de esquemas é contínuo durante o ciclo de vida de um banco de dados. TSQL2, Castro e OODM$|_{SV}$ permitem o sincronismo entre as operações de modificação de esquemas e manipulação de dados, intercalando as operações em transações. Tais abordagens têm a desvantagem de abortar a transação na presença de inconsistências. A integridade do banco de dados é sempre mantida, porém inúmeras operações podem ser freqüentemente desfeitas. A proposta mais completa é a apresentada por TIGUKAT, no qual algoritmos são propostos para gerenciar o processamento concorrente das transações. Entretanto, o foco do trabalho está voltado para o processamento concorrente de transações de dados e não, especificamente, o gerenciamento da evolução dos esquemas.

Tabela 2.5: Comparativo de trabalhos relacionados: manipulação de dados

Itens / Propostas	**TSQL2**	**Castro**	**Moreira**	**MSQL**	**TQL**	**OODM$\|_{SV}$**	***Farandole 2***	**Riedel**
Modelo de dados	relacional	relacional	relacional	relacional	OO	OO	OO	OO
Dimensão temporal	TT	TT, TV, bitemporal	TT, TV, bitemporal	−−	−−	bitemporal	×	×
Consulta a intenção	×	×	√	×	×	×	×	×
Consultas a extensão	√	√	√	√	√	√	√	√
Consultas às modificações de esquema	×	×	√	×	×	×	×	×
Consultas multi-esquemas	×	×	√	√	−−	√	×	√
Transações esquemas	√	√	×	×	√	√	√	×
Transações dados	√	√	×	×	√	√	√	×
Sincronismo: transações esquemas e dados	√	√	×	×	√	√	×	×

Legenda: √ Possui × Não possui −− Não conhecido

A manipulação de dados (consulta e atualização) é um tópico recente de pesquisa e apresenta um campo em aberto no contexto de gerenciamento de evolução de esquemas. Dentro desse contexto, é evidente a necessidade de um mecanismo que permita a sincronização das operações de manipulação de dados e as operações de modificação de esquemas durante a gerência da evolução de esquemas, garantindo a consistência do banco de dados enquanto este permanece em operação.

2.6 Semântica para Evolução de Esquemas

Conforme mencionado na seção 2.3, a inclusão de facilidades de evolução de esquemas em sistemas de bancos de dados envolve a solução de dois problemas fundamentais: a semântica de modificação de esquemas e a semântica de propagação de mudanças. Nesse sentido, a especificação formal da semântica de modificação de esquemas e da semântica de propagação de mudanças provê uma maior fundamentação para descrever as políticas de evolução de esquemas, fornecendo também uma base teórica e matemática que permite uma comparação segura com os demais modelos e abordagens propostos.

A seguir, os trabalhos relacionados à semântica de modificação de esquemas e à propagação de mudanças são apresentados e comparados entre si.

2.6.1 Trabalhos Relacionados

No campo de orientação a objetos, duas abordagens garantem a consistência com relação às modificações de esquema. A primeira é baseada na adoção de invariantes e regras (BANERJEE et al., 1987), sendo utilizada em sistemas como, por exemplo, ORION (BANERJEE et al., 1987) e O2 (FERRANDINA et al., 1995). As invariantes definem a consistência de um esquema e as regras especificam o comportamento a ser seguido para garantir a consistência do esquema após cada modificação realizada. A segunda apresenta um modelo axiomático, que formaliza a evolução dinâmica de esquemas (PETERS; ÖZSU, 1997) e a propagação de mudanças (PETERS; BARKER, 2000), no contexto do modelo temporal de objetos TIGUKAT (PETERS, 1994; ÖZSU et al., 1995).

OODM$|_{SV}$ (GRANDI; MANDREOLI, 2003) é um modelo formal para o suporte de versões temporais de esquemas em bancos de dados orientados a objetos. As definições

do modelo são compatíveis com os padrões estabelecidos pelo grupo ODMG (CATTELL et al., 2000). As operações de modificação de esquemas são formalizadas através de semântica operacional, permitindo uma análise formal do comportamento de cada mudança. Outro modelo formal é o TVOO (RODRÍGUEZ; OGATA; YANO, 1999a) que combina a tecnologia de orientação a objetos com os conceitos de versionamento temporal de esquemas. Porém, o foco do trabalho está restrito à semântica de modificação de esquemas.

Sintetizando, a Tabela 2.6 compara os trabalhos relacionados à semântica adotada para o mecanismo de evolução de esquemas nos dois níveis: a semântica de mudança e a propagação de mudança.

Tabela 2.6: Comparativo de trabalhos relacionados: semântica para evolução de esquemas

Propostas Itens de Comparação	**ORION**	**TIGUKAT**	**TVOO**	**OODM$\|_{SV}$**
Formalização	invariantes e regras	modelo axiomático	modelo axiomático	semântica operacional
Semântica para modificação de esquemas	$\surd$	$\surd$	$\surd$	$\surd$
Semântica para propagação de mudanças	$\times$	$\surd$	$\times$	$\surd$
Legenda:	$\surd$ Possui	$\times$ Não possui		

Com base nesta tabela, TIGUKAT e OODM$|_{SV}$ são trabalhos mais completos, pois especificam formalmente uma semântica para a modificação de esquemas e para a propagação de mudanças. Enquanto o primeiro define um modelo axiomático, o segundo utiliza semântica operacional. O modelo proposto nesta tese adota o mesmo estilo da segunda abordagem, utilizando a semântica operacional para especificar precisamente uma linguagem de modificação de esquemas e um mecanismo de atualização de dados, formalizando, desse modo, o comportamento dinâmico da evolução de esquemas. Sem tal precisão matemática, é difícil, por exemplo, assegurar a correção do modelo proposto. A necessidade de uma definição formal torna-se mais evidente quando a abordagem deve satisfazer restrições complexas, como, por exemplo, o modelo proposto nesta tese, que agrega os conceitos de versão e de tempo. Dentro desse contexto, podem ser citados vários exemplos de pesquisas recentes envolvendo semântica operacional e sistemas de tipos para uma rigorosa formalização de linguagens de bancos de dados. Bierman (BIERMAN, 2003) propõe uma semântica operacional e um sistema de tipos para uma linguagem de consulta para bancos de dados orientados a objetos. O trabalho apresentado em (COLAZZO et al., 2002) propõe um sistema de tipos para uma linguagem de consulta, utilizando esse sistema de tipos para verificar a coerência das consultas com a estrutura definida para o esquema. Em (SIMÉON; WADLER, 2003) é apresentada uma semântica formal para *XML Schema*. Outro exemplo bem conhecido é a padronização da linguagem XQuery (DRAPER et al., 2003) pela W3C, que também utiliza semântica operacional e um sistema de tipos.

2.7 Considerações Finais

Este capítulo apresentou uma revisão bibliográfica a respeito das características relevantes quanto ao gerenciamento de evolução de esquemas em bancos de dados orientados a objetos com a utilização dos conceitos de versão e/ou tempo. Os itens selecionados como parâmetros de comparação foram identificados perante o tratamento adotado pela maioria das propostas estudadas.

As abordagens examinadas utilizam os conceitos de versão e/ou tempo como forma

de tratar evolução de esquemas. Sendo assim, esses conceitos diferem de uma proposta para outra, visto que cada uma procura suprir suas carências próprias. Entre essas, podem ser encontradas:

- abordagens relacionais - STV (RODDICK, 1991), MTV (CASTRO; GRANDI; SCALAS, 1995a, 1997), PMTV (WEI; ELMASRI, 2000a,b);
- abordagens orientadas a objetos - ORION (BANERJEE et al., 1987), COAST (LAUTEMANN, 1997a, 1999), DBEvolution (BENATALLAH, 1999), Farandole 2 (AL-JADIR et al., 1995; AL-JADIR; LÉONARD, 1998), Sades (RASHID; SAWYER; PULVERMUELLER, 2000; RASHID, 2001), TIGUKAT (PETERS, 1994; ÖZSU et al., 1995; PETERS; ÖZSU, 1995, 1997; PETERS; BARKER, 2000), TVOO (RODRÍGUEZ; OGATA; YANO, 1999a,b), OODM$|_{SV}$ (GRANDI; MANDREOLI, 2003), SERF (CLAYPOOL; JIN; RUNDENSTEINER, 1998; CLAYPOOL; RUNDENSTEINER, 1999; RUNDENSTEINER et al., 1999; SU; CLAYPOOL; RUNDENSTEINER, 2000; CLAYPOOL; RUNDENSTEINER; HEINEMAN, 2000, 2001).

De modo geral, todas essas abordagens possuem basicamente os seguintes requisitos:

- modelo temporal e/ou de versões como base para evolução de esquemas;
- definição de uma álgebra para realização das modificações de esquemas;
- definição de uma estrutura para conservar o histórico das modificações realizadas;
- definição de mecanismos para garantir a consistência de esquema frente a qualquer modificação;
- definição de mecanismos para propagação das mudanças de esquema na base de dados vigente;
- definição de mecanismos para refletir o novo estado do banco de dados para as aplicações existentes; e
- mecanismo para manipulação de dados durante o processo de evolução de esquemas.

As Tabelas 2.7 e 2.8, no final do capítulo, apresentam um resumo comparativo, abordando, de forma geral, as características envolvidas no processo de evolução de esquemas conceituais. Foram considerados somente modelos e sistemas de evolução de esquemas, apesar de terem sido analisadas diversas outras propostas, comparadas ao longo do texto, que abordam alguns dos aspectos relacionados.

Apesar das vantagens de alguns trabalhos com relação a outros, o que se constata é que, apesar de reconhecida a importância dos mecanismos de evolução de esquemas em sistemas de bancos de dados e, mais recentemente, de várias linhas de pesquisa terem expandido e explorado tais funcionalidades, ainda não existe um consenso entre as propostas apresentadas nem um modelo completamente definido ou totalmente implementado. Verifica-se que dois importantes requisitos não são plenamente atendidos nesse contexto:

- um mecanismo completo de evolução de esquema, contemplando modificação e propagação de mudanças. Na maioria das propostas, estas duas etapas são tratadas separada e independentemente. Os seguintes requisitos são identificados:

  - definição completa das operações de modificação de esquema e sua repercussão no esquema do banco de dados;
  - especificação da semântica da propagação das modificações nas instâncias do banco de dados;
  - formalização de todo processo de evolução (modificação de esquemas e propagação de mudanças), de forma a garantir a integridade do esquema e dos inter-relacionamentos entre esquemas e instâncias;

- um processo de manipulação de dados durante a evolução de esquemas. A maioria das propostas negligencia este requisito. Porém, se o sistema de banco de dados presta suporte à evolução de esquemas, naturalmente deve gerenciar e manter a atualização e a consulta dos dados durante todo processo de evolução de esquemas.

Essas limitações, encontradas nos trabalhos relacionados, motivaram o desenvolvimento desta tese. A solução proposta atende aos requisitos mencionados, sendo apresentada a partir do capítulo 4.

Tabela 2.7: Comparativo de trabalhos relacionados: modelos e sistemas

Itens / Propostas	**STV**	**MTV**	**PMTV**	**ORION**	**COAST**	**DBEvolution**
Modelo de Dados	relacional	relacional	relacional	OO	OO	OO
Dimensão Temporal	TT, TV, bitemporal	TT, TV, bitemporal	bitemporal	×	×	×
Gerenciamento da Intenção	TT, TV, bitemporal	TT, TV, bitemporal	TT, TV, bitemporal	×	×	×
Gerenciamento da Extensão	TT, TV, bitemporal	TT, TV, bitemporal	TT, TV, bitemporal	×	×	×
Armazenamento da Extensão	mono	multi	mono, multi	−−	−−	−−
Sincronismo entre Intenção e Extensão	síncrono, assíncrono	síncrono, assíncrono	síncrono, assíncrono	−−	−−	−−
Granularidade do Versionamento	tabelas	tabelas	tabelas	objetos	esquemas	esquemas, classes, objetos
Método para evolução de esquemas	versões	versões	versões	conversão	versões	versões, conversão
Taxonomia de mudanças	√	√	×	√	√	√
Restrições para modificação	×	×	×	√	√	√
Propagação de mudanças	implícita	implícita	implícita	implícita	implícita	implícita
Métodos de Propagação	conversão imediata	conversão imediata	conversão imediata	conversão imediata	híbrido	híbrido (versões)
Atualização de dados	√	√	√	×	×	×
Processamento de Consultas	√	√	√	×	×	×
Compatibilidade com ODMG	⋆	⋆	⋆	−−	−−	−−
Semântica para Evolução de Esquemas	×	×	×	invariantes	×	×

Legenda: √ Possui × Não possui −− Não conhecido ⋆ Não se aplica

Tabela 2.8: Comparativo de trabalhos relacionados: modelos e sistemas (cont.)

Itens / Propostas	**Farandole 2**	**Sades**	**TIGUKAT**	**TVOO**	**OODM$\|_{SV}$**	**SERF**
Modelo de Dados	OO	OO	Temporal OO	OO	temporal OO	OO
Dimensão Temporal	×	×	−−	TT	bitemporal	×
Gerenciamento da Intenção	×	×	−−	−−	bitemporal	×
Gerenciamento da Extensão	×	×	−−	−−	bitemporal	×
Armazenamento da Extensão	−−	−−	−−	−−	bitemporal	−−
Sincronismo entre Intenção e Extensão	×	×	−−	assíncrono	síncrono	×
Granularidade do Versionamento	esquemas	classes	esquemas, tipos, objetos	esquemas, classes, objetos	esquemas	×
Método para evolução de esquemas	versões	versões	versões	versões	versões	conversão
Taxonomia de mudanças	√	−−	√	√	√	√
Restrições para modificação	√	−−	√	√	√	√
Propagação de mudanças	implícita, explícita	implícita, explícita	implícita	implícita, explícita	implícita, explícita	implícita
Métodos de Propagação	multi-objetos	conversão adiada	conversão adiada	conversão adiada, imediata (versões)	conversão imediata	conversão imediata
Atualização de dados	×	×	√	×	√	×
Processamento de Consultas	×	×	√	×	√	×
Compatibilidade com ODMG	−−	−−	√	−−	√	√
Semântica para Evolução de Esquemas	×	×	modelo axiomático	modelo axiomático	semântica operacional	×

Legenda: √ Possui × Não possui −− Não conhecido ⋆ Não se aplica

3 MODELO TEMPORAL DE VERSÕES

O Modelo Temporal de Versões (TVM) (MORO, 2001) é uma extensão do Modelo de Versões (GOLENDZINER, 1995), possibilitando representar toda história dos dados de uma aplicação. O conceito de versão permite a manutenção de alternativas de projeto, enquanto a dimensão temporal permite armazenar o histórico e a evolução dos dados das instâncias dos objetos.

Dois níveis de aplicação para o conceito de versão são identificados:

- versionamento de objetos - versões são utilizadas para representar a história dos dados de uma aplicação; e
- versionamento de esquemas - versões são utilizadas para a manutenção do histórico das modificações realizadas no esquema do banco de dados.

O TVM trata essencialmente da evolução histórica dos dados de uma aplicação, enquanto o foco da tese é o gerenciamento da evolução de esquemas conceituais em bancos de dados. Dentro desse contexto, o TVM é utilizado como base para definição das estratégias que regem o processo de evolução de esquemas em todos os seus aspectos: modificação de esquemas, propagação de mudanças e manipulação de dados. O TVM foi escolhido por apresentar características peculiares que auxiliam o processo de evolução de esquemas, permitindo uma modelagem mais natural e concisa da realidade. Sua utilização mostra-se adequada à resolução deste problema, conduzindo a uma maior completude o modelo de evolução de esquemas proposto.

Este capítulo apresenta uma revisão das principais características do TVM. O objetivo consiste em estabelecer as premissas nas quais se baseou o modelo proposto nesta tese. Primeiramente, o Modelo de Versões proposto por Golendziner (GOLENDZINER, 1995) é apresentado, uma vez que o TVM é a união desse modelo com características temporais. A seguir, são revistas as características do TVM e sua linguagem de consulta (TVQL -*Temporal Versioned Query Language*), considerando apenas os requisitos posteriormente utilizados na especificação do modelo de evolução de esquemas. Por fim, a linguagem TVQL é estendida a fim de incorporar expressões de caminho, característica relevante para a recuperação de versões de objetos durante o versionamento de esquemas.

3.1 Modelo de Versões

O Modelo de Versões (GOLENDZINER, 1995) estende um modelo de dados orientado a objetos, acrescentando conceitos e mecanismos que suportam a definição e a manipulação de objetos, versões e configurações. Uma versão é a descrição de um objeto em um determinado período de tempo, ou sob certo ponto de vista, cujo registro é

importante para a aplicação considerada. Sendo um objeto de primeira classe, possui um identificador único (`OID`), podendo ser diretamente manipulada e consultada. As versões de uma entidade do mundo real ficam agrupadas em um objeto versionado, que é também um objeto de primeira classe que mantém informações a respeito de suas versões.

As versões de um objeto versionado são organizadas formando um grafo acíclico dirigido que reflete seus relacionamentos, no qual apenas a primeira versão não possui uma antecessora. O mecanismo de versão corrente permite determinar automaticamente a versão de um objeto versionado a ser utilizada. As versões possuem um estado, para seu estágio de desenvolvimento, podendo ser em trabalho, estável ou consolidada.

O Modelo de Versões utiliza herança por extensão, a qual permite o versionamento em todos os níveis da hierarquia. A grande maioria dos modelos utiliza a herança por refinamento com o versionamento dos objetos apenas nas classes mais especializadas (folhas) da hierarquia de herança. Essa existência de versões nos vários níveis de uma hierarquia de herança introduz a idéia de mapeamentos entre versões, aos quais podem ser associadas restrições de cardinalidade (*n:m*, *1:1*, *1:n* ou *n:1*) que disciplinam as correspondências entre versões situadas em diferentes níveis.

Diferentes escolhas de versões para componentes e/ou ascendentes geram diferentes configurações para um mesmo objeto. Assim, é possível considerar que uma configuração é uma versão especial do objeto, chamada versão configurada. Versões configuradas podem fazer referência somente a outras versões configuradas, podendo ser compartilhadas por outras versões configuradas ou não. Além disso, uma versão configurada pode ser copiada, podendo ser novamente trabalhada.

A correspondência dos objetos com seus ascendentes e descendentes é feita através do identificador de objetos (`OID`). O identificador possui um componente comum a todos os objetos que representa a entidade modelada. A estrutura do identificador do Modelo de Versões é a seguinte: `OID ::= <id entidade, classe, número da versão>`.

Resumidamente, o Modelo de Versões apresenta as seguintes extensões ao modelo orientado a objetos:

- a classe raiz é chamada *Object* e é estendida com novas operações para suportar as características de versionamento do modelo;
- duas novas classes são definidas como subclasse da classe *Object*: (*i*) *Versioned Object*, que permite a modelagem de objetos versionados; (*ii*) *Version*, que permite a modelagem de versões;
- uma extensão na definição de classe também foi proposta para especificar as cardinalidades associadas às correspondências entre objetos e seus ascendentes e descendentes.

A Figura 3.1 ilustra a hierarquia de classes do Modelo de Versões.

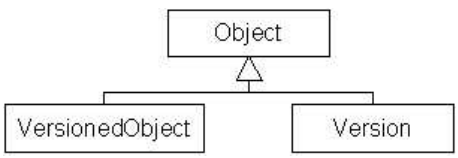


Figura 3.1: Hierarquia de classes do Modelo de Versões

3.2 Modelo Temporal de Versões

Apesar de a literatura apresentar uma vasta gama de modelos de versões (GOLENDZINER, 1995; WUU; DAYAL, 1993; AGRAWAL et al., 1991; BEECH; MAHBOD, 1988; BJÖRNERSTEDT; HULTÉN, 1989; CHOU; KIM, 1986; TALENS; OUSSALAH; COLINAS, 1993; KIM; BERTINO; GARZA, 1989) e de modelos temporais (EDELWEISS; OLIVEIRA, 1994; WUU; DAYAL, 1993; KÄFER; SCHÖNING, 1992; SU; CHEN, 1991) para bancos de dados orientados a objetos, a grande parte das pesquisas concentra-se na utilização desses modelos separadamente. Dentre as alternativas de modelos de versionamento, este trabalho baseia-se particularmente no TVM, que propõe uma extensão temporal (tendo como base o modelo TF-ORM (EDELWEISS, 1994; EDELWEISS; OLIVEIRA, 1994)) para o Modelo de Versões (GOLENDZINER, 1995), tratando os conceitos de tempo e de versão homogeneamente. O Modelo Temporal de Versões (TVM) (MORO, 2001) foi proposto como uma extensão definida sobre o Modelo de Versões com o acréscimo do conceito de tempo, permitindo armazenar as versões de um objeto e, para cada versão desse, o histórico das alterações feitas nos valores de seus atributos e relacionamentos dinâmicos.

Esta seção apresenta um resumo do TVM. Cabe salientar que somente as características relevantes ao gerenciamento de evolução de esquemas são abordadas. Maiores detalhes podem ser encontrados em (MORO et al., 2001a,b,c). Os artigos citados servem como referência, porém a sua leitura não é indispensável para compreensão da tese.

3.2.1 Representação Temporal

O tempo é associado a objetos, versões, atributos e relacionamentos. Um mesmo objeto apresenta uma linha de tempo para cada versão. Desse modo, duas diferentes ordens temporais são utilizadas: (*i*) tempo ramificado para um objeto, devido às diferentes linhas de tempo de cada versão que são originadas do objeto; e (*ii*) tempo linear para cada versão. O tempo varia de forma discreta, e a temporalidade é representada no modelo através do rótulo de elemento temporal (conjunto de intervalos temporais), bitemporal (tempos de validade e transação) e implícito.

Cada objeto possui o atributo pré-definido `alive` associado ao seu estado de vida. O atributo `alive` recebe o valor *true* no momento da criação do objeto, armazenando também seu tempo de validade inicial. Quando termina o tempo de vida do objeto, o valor do atributo `alive` é atualizado para *false*, e o tempo de validade final recebe o tempo de transação do momento da exclusão lógica.

Os atributos e relacionamentos das classes podem ser definidos como estáticos (quando não têm a variação de seus valores armazenada) ou temporalizados (todas as alterações de seus valores são armazenadas, formando seu histórico). A classificação de atributos e relacionamentos como temporalizados ou não fica sob responsabilidade do usuário, durante a especificação da aplicação, sendo permitido que uma mesma classe tenha atributos e relacionamentos de ambos os tipos (temporais ou não). Os valores dos atributos e relacionamentos temporais são associados com os rótulos pré-definidos `tTimei`, `tTimef`, `vTimei` e `vTimef`, que representam, respectivamente, os tempos de transação inicial e final, e os tempos de validade inicial e final.

3.2.1.1 Regras de Integridade Temporal

Regras de integridade temporal são estabelecidas a fim de garantir que o tempo de vida de uma versão esteja contido no tempo de vida do respectivo objeto. O mesmo deve acontecer para os valores armazenados para atributos e relacionamentos temporais, que apresentam valores contidos no tempo de vida da versão correspondente. Assim, definem-se as seguintes regras para o tempo de vida de um objeto:

1. o tempo de validade inicial de um atributo ou relacionamento temporalizado (`vTimei`) deve ser maior ou igual ao tempo de vida inicial (`alive.vTimei`) da versão à qual pertence;

2. o tempo de transação inicial de um atributo ou relacionamento temporalizado (`tTimei`) deve ser maior ou igual ao tempo de vida inicial (`alive.vTimei`) da versão à qual pertence;

3. o tempo de vida final (`alive.vTimef`) de uma versão deve ser maior que o inicial (`alive.vTimei`) e maior ou igual ao maior tempo de transação e validade final das variações de seus atributos e relacionamentos temporalizados (`tTimef`), (`vTimef`);

4. o tempo de vida inicial (`alive.vTimei`) de uma versão deve ser menor que o final (`alive.vTimef`) e maior ou igual ao tempo de vida inicial (`alive.vTimei`) do objeto versionado;

5. o tempo de vida final (`alive.vTimef`) de um objeto versionado deve ser maior que o inicial (`alive.vTimei`) e maior ou igual ao maior tempo de vida final de suas versões (`alive.vTimef`).

Por exemplo, considerando uma versão **`V`** com um atributo (ou relacionamento) temporalizado **`a`** sob uma linha de tempo **`t`**. As Figuras 3.2-a e 3.2-b apresentam as possíveis combinações para os valores de `vTimei`, `vTimef`, `tTimei` e `tTimef` em relação ao atributo `alive` da versão, segundo as regras 1, 2 e 3. A Figura 3.2-c mostra os possíveis valores de validade inicial e final do atributo `alive` para essa versão de um objeto versionado **`OV`**, segundo as regras 4 e 5.

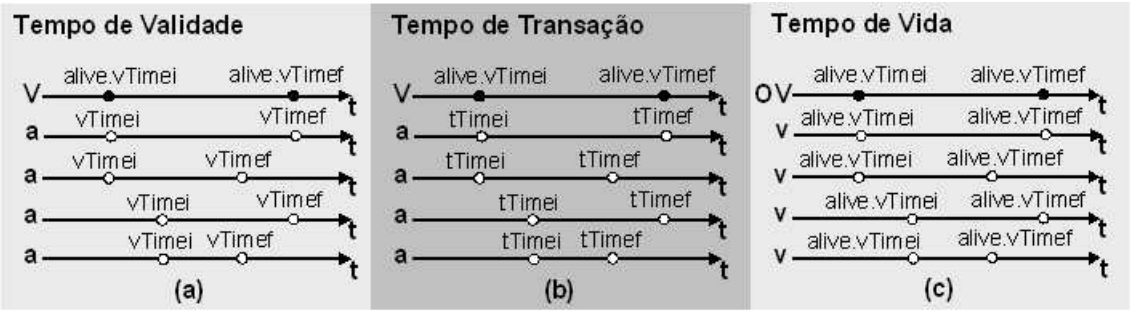


Figura 3.2: Regras de Integridade Temporal

Inserção e Atualização

A inserção é a primeira definição de um valor para um atributo. Qualquer informação inserida deve receber o tempo de transação e o tempo de validade. O usuário deve fornecer o tempo de validade inicial e, opcionalmente, o final da informação que está

sendo inserida. O tempo de transação final é inserido automaticamente quando um novo valor, gerado por uma atualização, for definido, ou quando terminar sua validade. Quando o usuário não fornecer o tempo de validade final, ele será igual a *null* (valendo até que outra informação seja definida ou o objeto seja excluído).

Em bancos de dados bitemporais nenhum atributo é fisicamente modificado, exceto aqueles cujo tempo de transação final está em aberto (ELMASRI; NAVATHE, 2000). As informações não são perdidas ou excluídas em atualizações, uma vez que o tempo de transação possibilita a distinção entre duas ou mais informações que possuam o tempo de validade igual. Sendo assim, qualquer definição de novo valor é realizada com o tempo de transação definindo a semântica das informações.

Exclusões Lógica e Física

A exclusão de versões pode ser lógica ou física. Quando uma versão é excluída logicamente, ela passa para o estado desativada e tem seu tempo de vida finalizado. No momento da exclusão lógica, se houver atributos e relacionamentos temporalizados, os tempos finais de validade e transação em aberto recebem o mesmo valor de tempo final definido para o objeto ou versão. A exclusão física somente é utilizada quando se deseja remover fisicamente uma informação que não é mais relevante. Essa operação, conhecida como *vacuuming*[1], sendo realizada raramente, somente quando se quer diminuir o volume de dados armazenados. Entretanto, deve-se estar ciente de que essas informações serão perdidas (HÜBLER; EDELWEISS, 2000). O TVM não define as regras para este tipo de exclusão, assumindo que qualquer remoção corresponde a uma exclusão lógica.

3.2.1.2 Hierarquia de Tipos Temporais

O TVM define um conjunto de tipos que modelam o histórico e os rótulos temporais para os atributos e os relacionamentos, conforme Figura 3.3[2].

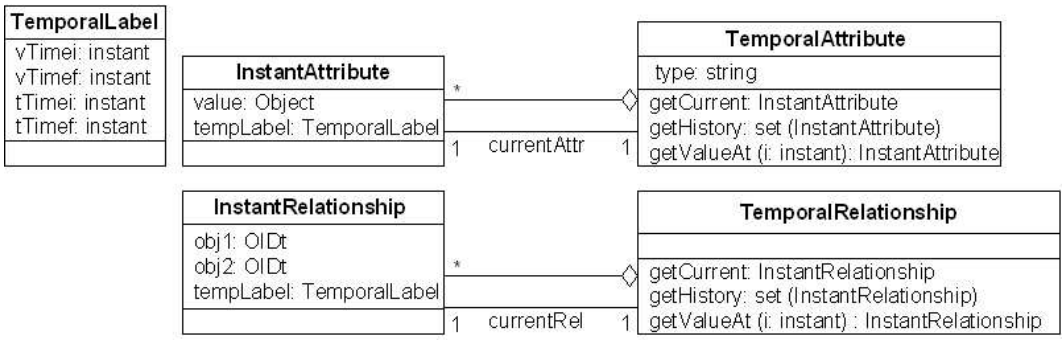


Figura 3.3: Hierarquia de Tipos Temporais

A classe `TemporalLabel` armazena o rótulo temporal com os tempos de validade inicial e final, e transação inicial e final. A classe `InstantAttribute` armazena o valor do atributo com seu rótulo temporal e a classe `InstantRelationship` armazena os identificadores dos objetos (ou versões) que pertencem ao relacionamento e seu rótulo temporal. As classes `TemporalAttribute` e `TemporalRelationship` são agregados

[1] O impacto da utilização desse tipo de exclusão na base de dados e na linguagem de consulta é descrito detalhadamente por Jensen em (JENSEN, 1999).

[2] Todos os diagramas deste trabalho são apresentados em UML (OMG, Object Management Group, 2003).

dos conjuntos de valores que os atributos e relacionamentos temporais assumem durante as vidas dos objetos. Essas classes possuem um relacionamento com o instante corrente (`currentAttr` e `currentRel`, respectivamente). A operação `getHistory` retorna todos os valores com os rótulos de tempo, `getHistoryOfValue` retorna o histórico do valor passado por parâmetro e `getValueAt` retorna os valores em um instante de tempo específico, passado como parâmetro.

3.2.2 Estados de uma Versão

As versões passam por diferentes estados que refletem seu estágio de desenvolvimento e/ou consistência, e que definem as operações que podem ser executadas sobre elas. O diagrama de estados da Figura 3.4 ilustra as transições entre um estado e outro, bem como as operações que ocasionam tais transições.

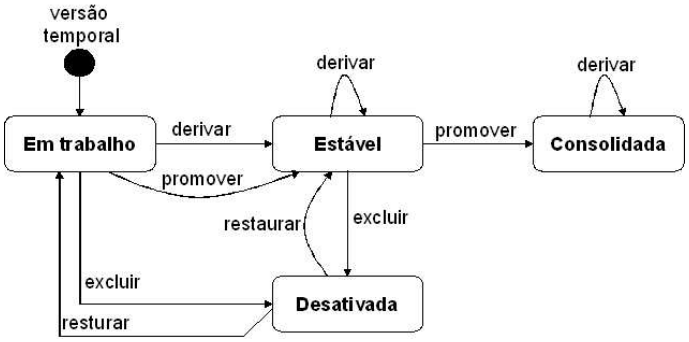


Figura 3.4: Diagrama de estados de uma versão de objeto

Cada estado define um conjunto de operações que podem ser aplicadas sobre uma versão, conforme a Tabela 3.1. Desse modo, quando uma versão está no estado:

- em trabalho - pode servir como base para derivação de uma nova versão, ser promovida para estável, ser alterada, consultada ou ainda excluída (logicamente);
- estável - pode servir como base para derivação de uma nova versão, ser promovida para consolidada, compartilhada por outros usuários, excluída (logicamente), desde que não possua nenhuma versão sucessora, e não pode mais ser alterada;
- consolidada - pode servir como base para derivação de uma nova versão, ser consultada, compartilhada por outros usuários, porém não pode ser alterada nem excluída;
- desativada - pode ser apenas consultada e restaurada.

3.2.3 Hierarquia de Classes

A Figura 3.5 ilustra a hierarquia de classes do TVM. A classe *Temporal Object* permite a representação de aspectos temporais. As linhas tracejadas informam a parte da hierarquia que fica transparente ao usuário, ou seja, as operações de controle do objeto versionado não podem ser executadas diretamente pelo mesmo.

A hierarquia do TVM permite especificar dois tipos de classes de aplicação:

- classe de aplicação não temporal e não versionável - definida como subclasse de *Object*. Sua modelagem pode ser usada para contemplar classes que representam objetos de um modelo já existente ou classes auxiliares nas quais os conceitos de tempo e versão não são necessários;

Tabela 3.1: Estados das versões de objetos e respectivas operações

Estados Operações	**Em Trabalho**	**Estável**	**Consolidada**	**Desativada**
Derivar	√	√	√	−−
Promover	√	√	−−	−−
Alterar	√	×	×	×
Excluir	√	√	×	×
Consultar	√	√	√	√
Compartilhar	×	√	√	×
Restaurar	−−	−−	−−	√

Legenda: √ Pode ser realizada × Não pode ser realizada −− Não definida

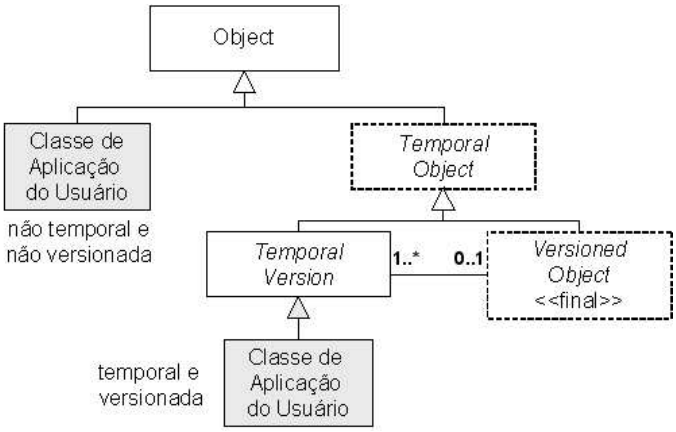


Figura 3.5: Hierarquia de classes do TVM e classes de aplicação

- classe de aplicação temporal versionável - definida como subclasse de *Temporal Version*. Seus atributos e relacionamentos podem ser definidos como estáticos ou temporalizados, e suas instâncias são versões com rótulo de tempo associado (atributos *start* e *end* herdados de *Temporal Object*).

A Figura 3.6 apresenta graficamente os objetos instanciados de uma classe de aplicação, definida como temporal versionada. Isso significa que a classe definida pelo usuário é subclasse da classe *TemporalVersion*. São apresentados um objeto sem versões (`Objeto 1`) e um objeto versionado com suas respectivas versões (`V0, V1, V2, V3`).

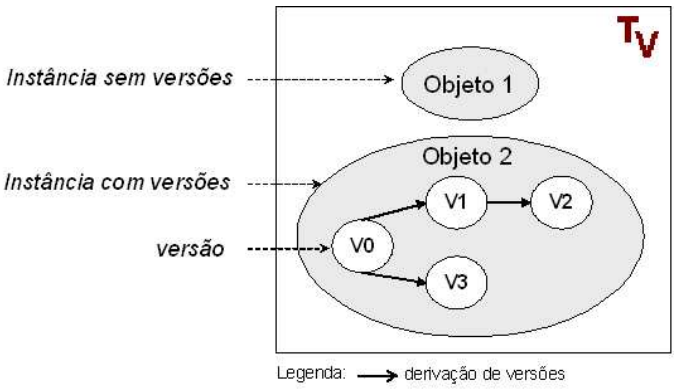


Figura 3.6: Representação de instâncias de uma classe

3.2.4 Relacionamento de Herança por Extensão

Uma das características mais importantes do Modelo de Versões é a definição de herança por extensão, no qual as versões são admitidas nos vários níveis da hierarquia de herança (conforme citado na seção 3.1). 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.

O TVM implementa dois tipos de herança: herança por refinamento e herança por extensão. 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 (por exemplo, em tempo de execução).

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 ilustrado na Figura 3.7. Esse conceito é apresentado por Biliris, em (BILIRIS, 1990).

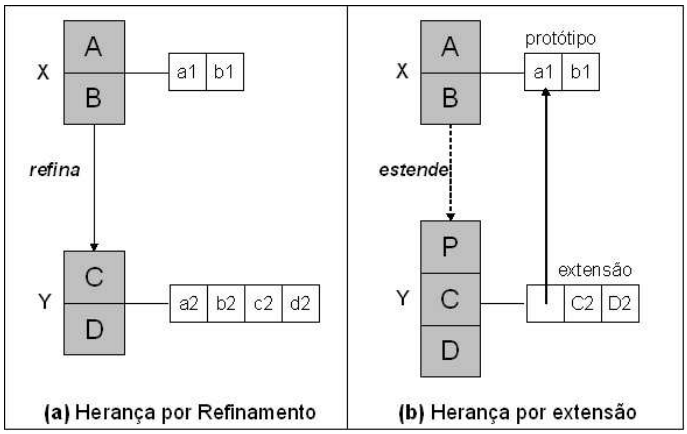


Figura 3.7: Implementação de herança no TVM

A fim de evitar confusões com a herança adotada nas linguagens e modelos orientados a objetos, denominada herança por refinamento, o TVM propõe que o mecanismo de herança por extensão seja modelado como uma especialização do conceito de relacionamento. Esse mecanismo é chamado de relacionamento de herança por extensão. A hierarquia estabelecida entre as classes participantes desse relacionamento é denominada hierarquia por extensão.

3.2.4.1 Correspondências entre Objetos e Versões

A representação de versões nos diversos níveis de hierarquia permite a existência de múltiplos ascendentes para um objeto (versionado, sem versões ou versão) em

uma subclasse, caso o objeto ascendente possua versões. É permitido ao usuário estabelecer restrições de cardinalidade (mapeamentos) entre versões de um objeto em uma classe e versões de seu ascendente na superclasse. Essas restrições são chamadas de correspondências e são modeladas como a cardinalidade de um relacionamento de herança por extensão (Figura 3.8), podendo ser:

- 1:1 - cada versão na subclasse corresponde a exatamente uma versão na superclasse;
- 1:n - uma versão na superclasse pode corresponder a somente uma versão na subclasse, mas uma versão na subclasse pode corresponder a várias versões na superclasse;
- n:1 - uma versão na subclasse pode corresponder a somente uma versão na superclasse, mas uma versão na superclasse pode corresponder a várias versões na subclasse;
- n:m - várias versões na subclasse podem estar relacionadas com uma versão na superclasse, e cada versão na subclasse pode corresponder a várias versões na superclasse.

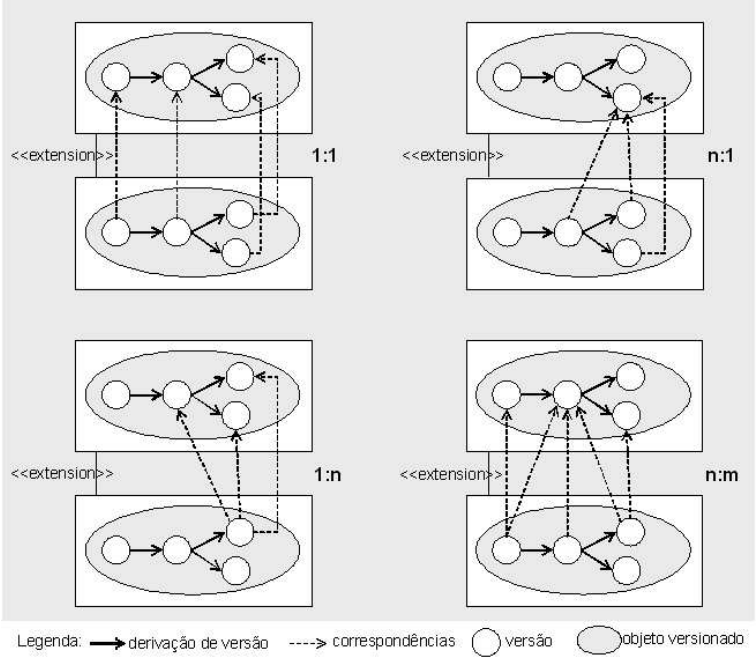


Figura 3.8: Tipos de correspondências entre versões nos níveis da hierarquia de herança

Objetos não versionados e objetos versionados sem versão, são considerados como uma versão, para efeitos de verificação de restrições de cardinalidade imposta pela correspondência.

3.2.5 Configuração

O sistema de configurações continua basicamente o mesmo estabelecido no Modelo de Versões. A única diferença é que, assim como as versões e os objetos versionados, a configuração possui o comportamento temporal estabelecido.

Considerando um objeto composto, em que os componentes podem ser versionados, uma configuração associa exatamente uma versão para cada um desses componentes.

Como o modelo apresenta herança por extensão, esse processo deve incluir a definição de uma versão para cada um dos níveis onde o objeto está representado. Assim, diferentes escolhas de versões para componentes e/ou ascendentes geram diferentes configurações para o mesmo objeto, o que permite considerar que uma configuração é uma versão especial de um objeto, chamada versão configurada.

3.2.6 Relacionamento entre as Classes Normais e Temporais Versionadas

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

- associação - pode ser definida entre as classes normais, desde que o relacionamento não seja temporal. Caso contrário, é possível entre quaisquer tipos de classe;
- herança por refinamento - pode ser definida somente entre classes normais;
- herança por extensão - pode ser definida 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 - pode ser definida entre duas classes normais, duas temporais versionadas e entre uma temporal versionada e normal (sendo a classe normal componente).

3.2.7 Linguagem de Definição de Classe

A Figura 3.9 apresenta a sintaxe geral e simplificada da linguagem para definição de uma classe proposta para o TVM (A BNF completa está especificada em (MORO, 2001)). As classes não temporais e não versionadas são representadas na BNF por `normalClass`. As classes temporais versionadas são denominadas como `tempVersionClass`. A cláusula `correspondence` permite estabelecer o tipo de cardinalidade entre a classe descendente e sua ascendente. A cardinalidade entre as classes normais é omitida (`1:1`), pois em cada classe só pode haver um objeto ascendente ou descendente. As demais cardinalidades só podem ser utilizadas em subclasses da raiz temporal versionada (subclasse de *TemporalVersion*), sendo que, se o usuário não especifica um valor, a cardinalidade assumida é de um para um. A cláusula `temporal` indica que um atributo ou relacionamento terá sua evolução armazenada.

```
class ::= [public] [ abstract | final]
   class className [hasVersions] [ inherit
      [byExtension] className [correspondence (1:1 | 1:n | n:1 | n:m)]]
      [[temporal] aggregate_of[n] className (by value | by reference)
      {,[temporal]aggregate_of[n] className (by value | by reference)}]
   ( [ Properties:
        { [ public | private | protected] [static]
        [temporal] attributeName: attributeDomain;}+]
     [ Relationships:
        { [temporal] relationshipName (0:1 | 0:n | 1:1 | 1:n | n:m)
        [inverse inverseRelationshipName ] relatedClass;}+ ]
     [ Operations:
        { [ public | private | protected] [static]
        [abstract | final] operationDefinitions }+ ] )
```

Figura 3.9: Sintaxe simplificada da Linguagem de Definição de classes do TVM

3.2.8 Representação Gráfica

Foi definida uma estrutura de representação gráfica para o TVM, estendendo o diagrama de classes da UML. A notação é apresentada na Tabela 3.2, sendo identificadas as características temporais e de versões.

Tabela 3.2: Representação gráfica para os elementos do TVM

Notação	Significado
TV	classe temporal e versionável
«final»	classe final, não pode ser especializada
itálico	nome da classe, indica classe abstrata
«temporal»	relacionamento temporal
«T» ou t	atributo temporal
«extension»	relacionamento de herança por extensão
negrito	operação final (*final*), não pode ser redefinida
$	atributo ou operação de escopo de classe
+	atributo ou operação pública (*public*)
–	atributo ou operação particular (*private*)
#	atributo ou operação protegida (*protected*)

3.2.9 Exemplo da utilização do TVM

Esta seção apresenta um breve estudo de caso para ilustrar a modelagem de uma aplicação real com o uso do TVM. A base de dados faz parte do Sistema Discente, um Sistema de Informação Legado (SIL), com características temporais, que controla as atividades de graduação na Universidade Federal do Rio Grande do Sul (UFRGS).

3.2.9.1 Modelagem da Aplicação

O Sistema Discente foi desenvolvido pelo Centro de Processamento de Dados da UFRGS, pelo qual é mantido até os dias atuais, sendo composto por três subsistemas: Alunos, Currículos e Turmas. A base de dados associa tempo de validade a grande parte das informações que representa, introduzindo aspectos de base temporal ao sistema. O subsistema Currículo armazena todas as versões de currículo definidas para os cursos oferecidos pela UFRGS, inclusive os já extintos. O subsistema Alunos mantém o histórico completo (como, por exemplo, admissão, trancamento, matrícula, entre outros) de todos os alunos e ex-alunos da UFRGS. O subsistema Turmas armazena informações de turmas, como alocação de vagas e salas para estas. O foco do estudo concentra-se no processo de Revisão Curricular, no qual as universidades atualizam seus cursos trocando currículos com conteúdo defasado por currículos mais novos. O exemplo utiliza dois subsistemas: (*i*) Currículo - por representar possíveis revisões curriculares; e (*ii*) Alunos - por serem diretamente afetados por tais revisões.

O domínio modelado apresenta as seguintes regras de negócio:

- um aluno ingressa na universidade pela primeira vez normalmente através de um concurso denominado vestibular, recebendo um número de matrícula que o identifica durante toda sua vida acadêmica;
- alunos têm sempre um vínculo com algum curso, caracterizado pelo seu ingresso e afastamento. Em situações de transferência de curso, novo concurso ou reingresso, o número de matrícula do aluno permanece o mesmo do primeiro ingresso;

- o histórico escolar de um aluno é composto pelas disciplinas nas quais ele se matriculou, dos aproveitamentos de créditos obtidos (em geral, quando cursados em outras instituições) e das liberações que obteve em função de equivalência entre disciplinas em currículos distintos;
- cada curso possui um único currículo vigente. Cada currículo é composto por um conjunto de disciplinas distribuídas em semestres;
- uma turma é composta pela junção entre uma disciplina e uma sala, compondo os horários das turmas.

A Figura 3.10 apresenta o esquema atual do Sistema Discente, incluindo os três módulos: Controle de Alunos, Programação de Currículos e Programação de Turmas. Para melhor compreensão, apenas os aspectos mais relevantes de cada subsistema são mostrados. Exceto a operação que calcula a prioridade de matrícula (`calcOrdemMatricula()`), todas as demais são omitidas, pois não são necessárias nesse contexto.

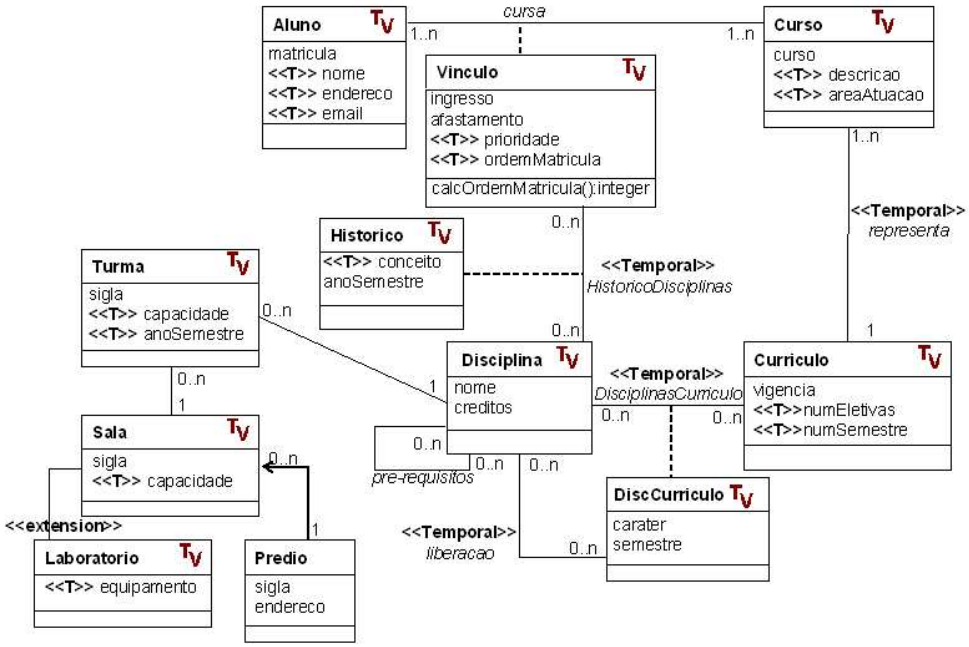


Figura 3.10: Sistema Discente da UFRGS

As características de versões e tempo são utilizadas em:

- classe `Aluno` - armazena o histórico completo do aluno na universidade;
- classe `Vínculo` - em cada vínculo o aluno possui uma prioridade de matrícula, composta por índices que determinam seu desempenho no curso (aprovações, reprovações e cancelamentos) e seu posicionamento entre os alunos deste curso para cálculo de prioridade de matrícula (método `calcOrdemMatricula()`);
- classe `Disciplina` - armazena o histórico de todas as disciplinas oferecidas pela universidade. Possuem propriedades que são independentes do currículo no qual participam, como número de créditos e pré-requisitos;
- classe `Histórico` - armazena o histórico das disciplinas nas quais o aluno se matriculou;

- classe `Curso` - armazena o histórico de todos os cursos oferecidos pela universidade;
- classe `Currículo` - possui um tempo de validade que estabelece a vigência de cada versão de currículo. Há diversos currículos vigentes no mesmo momento, um para cada curso;
- classe `DiscCurrículo` - armazena propriedades das disciplinas definidas pelo seu vínculo a um currículo específico, como caráter (opcional ou obrigatória) e o semestre recomendado;
- classe `Prédio` - armazena os prédios com suas respectivas salas de aula;
- classe `Turma` - armazena o histórico dos horários das turmas nas salas;
- relacionamento `históricoDisciplinas` - armazena o histórico de todas as disciplinas cursadas pelo aluno;
- relacionamento `representa` - cada curso tem sempre um único currículo vigente em cada instante de tempo, porém pode possuir diversos currículos se analisado em momentos distintos;
- relacionamento `disciplinasCurrículo` - armazena o histórico de todas as disciplinas que compõem (ou compuseram) um currículo;
- relacionamento `liberação` - define regras de aproveitamento de créditos obtidos pelos alunos em currículos anteriores, facilitando a transição entre currículos. Uma disciplina de um currículo pode ser liberada por uma ou mais disciplinas que compuseram currículos anteriores do mesmo curso.

3.2.9.2 Representação Gráfica das Instâncias

A Figura 3.11 apresenta uma representação gráfica da evolução do relacionamento `representa`, entre as classes `Curso` e `Currículo`. A versão de curso é sempre a mesma (por exemplo, `'engenharia da computação'`), entretanto a associação com o currículo muda de acordo com as vigências dos currículos.

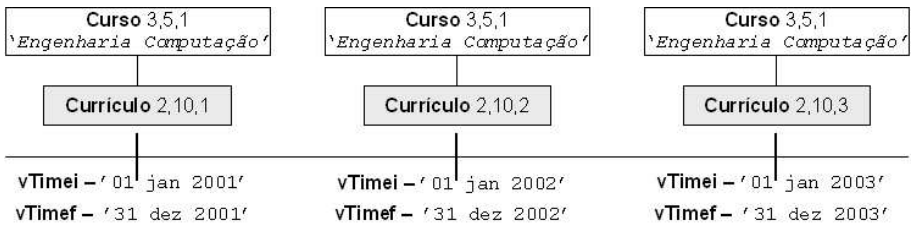


Figura 3.11: Relacionamento temporal entre curso e currículo - `representa`

A Figura 3.12 ilustra as instâncias do objeto versionado `Curriculo`, que participam do relacionamento com o curso `'engenharia da computação'`. O objeto versionado currículo possui três versões, no qual o ano de vigência é `2001` (versão `2,10,1`), `2002` (versão `2,10,2`) e `2003` (versão `2,10,3`). As versões `2002` e `2003` são alternativas da primeira versão criada. Observa-se que a mudança do valor do atributo número de eletivas (`numEletivas`) não acarreta a derivação de uma nova versão, porque o atributo é temporal, ao invés disso, apenas mais um valor é armazenado no histórico do atributo.

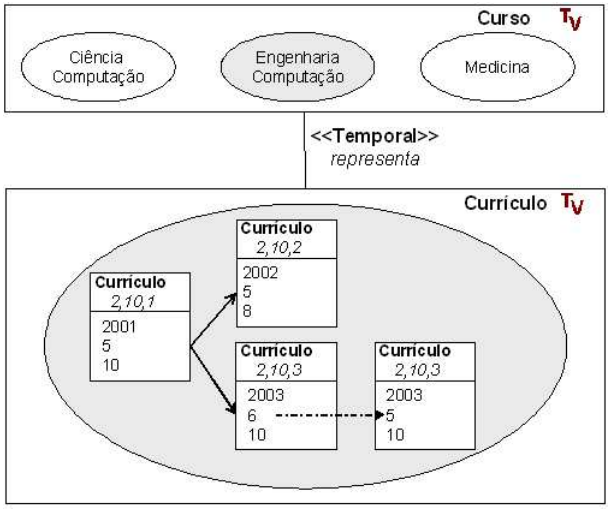


Figura 3.12: Representação gráfica das instâncias da classe Currículo

3.2.9.3 Considerações sobre a modelagem do Sistema Discente

Todas as informações dos subsistemas Alunos e Currículo armazenam o tempo de validade, exceto para os pré-requisitos. Contudo, esta informação não pode ser considerada perdida, pois existe na forma textual na documentação da universidade. A modelagem do Sistema Discente ilustra como as facilidades do TVM podem ser usadas em um sistema do mundo real. O processo de revisão curricular é facilitado à medida que todos os períodos de cada currículo alternativo são armazenados, bem como todo o histórico escolar de cada aluno. Assim, os impactos das revisões curriculares podem ser analisados em dois aspectos: (i) quando ocorre uma revisão curricular, a universidade tenta evitar que existam currículos simultâneos para um mesmo curso; (ii) um aluno pode se formar caso cumpra todos os requisitos do currículo vigente na época de sua formatura, o qual não necessariamente é o mesmo vigente na época de seu ingresso no curso. Todas essas informações estão armazenadas através do histórico temporal de Alunos e Currículo.

3.3 Linguagem de Consulta para o Modelo Temporal de Versões

Após apresentar as principais características do TVM, esta seção apresenta brevemente a especificação da sua linguagem de consulta, chamada TVQL (*Temporal Versioned Query Language*) (MORO et al., 2002). A TVQL permite, além de consultas básicas realizadas pela linguagem padrão SQL, novas consultas que retornam valores específicos dos aspectos de tempo e versão. A Figura 3.13 apresenta a sintaxe geral da TVQL.

A TVQL é baseada em SQL, mantendo os seguintes aspectos: estrutura-base com `SELECT-FROM-WHERE`; alias na cláusula `FROM`; eliminação de duplicatas com `SELECT DISTINCT`; operadores relacionais (`<, >, =, >=, <=, <>`), lógicos (`OR, AND, NOT`), de conjuntos (`UNION, INTERSECTION, DIFFERENCE`), de condições compostas (`IN, BETWEEN AND`) e de combinação (`LIKE`); funções de agregação (`COUNT, SUM, AVG, MIN, MAX`); e cláusulas para grupos de dados (`GROUP BY, HAVING, ORDER BY`).

Por *default*, as consultas que não apresentam nenhuma condição temporal específica nas cláusulas `SELECT` e `WHERE` retornam apenas os valores atuais dos dados dos objetos

```
query ::= SELECT [EVER][DISTINCT] targetC {,targetC}*
          FROM identificC {,identificC}* [WHERE[EVER]searchC]
          [GROUP BY groupC {,groupC}* [HAVING logicalExpr]]
          [ORDER BY orderC {,orderC}* [setOp query]];
targetC         ::= (*|propertyName|aggregationFunctions|preDefInterval
                    |preDefInstant) [AS identifier]
identificC      ::= className [.VERSIONS] [aliasName]
searchC         ::= logicalExpr|tempExpr
groupC          ::= propertyName|preDefInterval|preDefInstant
logicalExpr     ::= AnyLogicalExpression
tempExpr        ::= logicalExpr|PRESENT (logicalExpr)
orderC          ::= groupC [ASC|DESC]
setOp           ::= UNION | INTERSECTION | DIFFERENCE
preDefInterval ::= [propertyName.](tInterval|vInterval)
preDefInstant  ::= [propertyName.](tiInstant|tfinstant|viInstant|vfInstant)
                    |[className.](iLifeTime)|(fLifeTime)
```

Figura 3.13: Sintaxe geral da TVQL

ativos (não excluídos). Considerando as características temporais do TVM, a TVQL propõe uma série de propriedades para tempo e intervalos pré-definidos com um conjunto de operadores de comparação.

Diferentes combinações de valores correntes e históricos podem ser declarados em uma consulta, dependendo do uso das palavras reservadas `EVER` e `PRESENT`.

A palavra `EVER` pode ser utilizada nas cláusulas `SELECT` e `WHERE`, podendo retornar o histórico da vida de objetos e versões. Na cláusula `SELECT`, a consulta retorna o histórico das propriedades temporais selecionadas, considerando o histórico das propriedades temporais mencionadas na cláusula `WHERE`. Na cláusula `WHERE`, a consulta considera o histórico dos valores das propriedades temporais na cláusula. Por exemplo, considerando a classe `Aluno`, do diagrama de classes da Figura 3.10:

1.
```
SELECT nome
FROM Aluno
WHERE endereco LIKE 'Protásio Alves,%';
```

2.
```
SELECT EVER nome
FROM Aluno
WHERE endereco LIKE 'Protásio Alves,%';
```

3.
```
SELECT nome
FROM Aluno
WHERE EVER endereco LIKE 'Protásio Alves,%';
```

A primeira consulta retorna os nomes dos alunos que moram atualmente no endereço `Protásio Alves`. A segunda retorna todos os nomes dos alunos que moraram alguma vez (durante toda história) no endereço, incluindo os que moram atualmente. A terceira retorna os valores dos nomes atuais dos alunos que alguma vez moraram no endereço `Protásio Alves`. Porém, o resultado da consulta com `EVER` depende dos atributos serem temporais.

A palavra `PRESENT` considera apenas dados ativos, válidos hoje, eliminando do resultado dados excluídos. A função de `PRESENT` na cláusula `WHERE` anula a semântica temporal da palavra reservada `EVER`. Tanto a cláusula `EVER` quanto a cláusula `PRESENT`

só interferem no resultado da avaliação de propriedades temporais, não interferindo em consultas que possuem apenas atributos e relacionamentos não temporais. Por exemplo, considerando ainda a classe `Aluno`, do diagrama de classes da Figura 3.10:

1.
```
SELECT nome
FROM Aluno
WHERE EVER endereco LIKE 'Protásio Alves,%'
      PRESENT (email LIKE '%@inf.ufrgs.br');
```

2.
```
SELECT EVER nome
FROM Aluno
WHERE endereco LIKE 'Protásio Alves,%'
      PRESENT (email LIKE '%@inf.ufrgs.br');
```

As duas consultas retornam os nomes dos alunos que moraram alguma vez no endereço `Protásio Alves` e que atualmente possuem email com domínio `@inf.ufrgs.br`. A primeira retorna os nomes atuais dos alunos e a segunda, o histórico dos nomes.

A TVQL define propriedades temporais para a recuperação específica de rótulos temporais que podem ser utilizados nas cláusulas `SELECT` e `WHERE`: `tInterval` (intervalo de tempo de transação); `vInterval` (intervalo de tempo de validade); `tiInstant` (tempo de transação inicial); `tfInstant` (tempo de transação final); `viInstant` (tempo de validade inicial); e `vfInstant` (tempo de validade final). Para recuperação dos tempos de vida inicial e final dos objetos e versões são definidas as propriedades `iLifetime` (tempo de vida inicial do objeto ou versão), e `fLifeTime` (tempo de vida final do objeto ou versão).

A TVQL propõe também um conjunto de operadores de comparação específicos para condições temporais, sendo determinados pelo usuário na cláusula `WHERE`. Cada um dos operadores tem uma finalidade específica: `INTERSECT` - verifica se um intervalo intersecciona qualquer ponto de outro intervalo; `OVERLAP` - verifica se um intervalo intersecciona todo o outro intervalo; `EQUAL` - verifica se os intervalos são iguais; `BEFORE` - verifica se um intervalo ou instante antecede totalmente um intervalo; `INTO` - verifica se um intervalo ou instante está contido em um intervalo; e `AFTER` - verifica se um intervalo ou instante está após um intervalo.

Considerando as características de versionamento, o usuário define na cláusula `FROM` os objetos que devem ser consultados. Os objetos e versões correntes são definidos pelo nome da classe à qual pertencem. Todas as versões dos objetos são consideradas, acrescentando na cláusula `FROM` o nome da classe seguido da palavra reservada `versions`.

Três grupos de funções para a recuperação específica das informações sobre versões e objetos versionados são definidos na TVQL: funções que recuperam o estado (`isWorking, isStable, isConsolidated` e `isDeactivated`); funções da navegação na hierarquia de herança (`isAscendantOf` e `isDescendantOf`); e funções da navegação na hierarquia de derivação (`isFirst, isLast, isSuccessorOf, isPredecessorOf, isCurrent, isUserCurrent, isConfiguration`).

3.4 Expressões de Caminho

Para que a Linguagem de Atualização de Objetos (definida no capítulo 6) possa recuperar versões de objetos temporais foi necessário incorporar expressões de caminho

à TVQL. Sendo o TVM um modelo orientado a objetos, foi adotada uma estrutura similar à OQL (*Object Query Language*) (CATTELL et al., 2000). Cabe salientar que somente as funcionalidades requeridas pela linguagem proposta neste trabalho são especificadas nesta seção.

O acesso aos componentes dos objetos e estruturas é feito usando a notação ponto[3] (`.`). Por exemplo, se `a` é um objeto da classe `C` e `p` é uma propriedade desta classe, então para `a.p` são estabelecidas as seguintes condições:

- se `p` é um atributo, então `a.p` é o valor do atributo para o objeto `a`;
- se `p` é um relacionamento, então `a.p` é um objeto ou uma coleção de objetos que estão relacionados com `a` através de `p`;
- se `p` é um método (com parâmetros ou não), então `a.p()` é o resultado da aplicação do método `p` sobre o objeto `a`.

Para exemplificar, considera-se as seguintes consultas, tendo como base o diagrama de classes da Figura 3.10:

1.
```
SELECT a.nome
FROM Aluno a;
```

2.
```
SELECT v.calcOrdemMatricula()
FROM Vinculo v;
```

3.
```
SELECT pre.nome
FROM Disciplina.preRequisito as pre;
```

4.
```
SELECT p.creditos
FROM Disciplina as d,
     d.preRequisito as p;
```

A primeira consulta retorna todos os nomes de todos os alunos. A segunda consulta retorna a ordem de matrícula, apresentando um exemplo de consulta a partir de operações. O acesso às operações é realizado na mesma posição em que os atributos são utilizados. A terceira e quarta consultas ilustram a navegação em relacionamentos. A terceira consulta retorna um conjunto com os nomes dos pré-requisitos de uma determinada disciplina. Isto significa que é possível alcançar um objeto a partir de outro já obtido inicialmente pela consulta. A consulta inicia por disciplina, retornando seus pré-requisitos, a seguir obtém-se disciplina novamente, finalizando com o atributo nome. A quarta consulta retorna um conjunto com os créditos dos pré-requisitos de cada disciplina do banco de dados. Isso significa que para cada disciplina serão percorridos todos os seus pré-requisitos.

O acesso às informações temporais e de versões segue as mesmas definições da TVQL. Por exemplo, a primeira consulta, apresentada a seguir, retorna o tempo de vida inicial e final do objeto `Aluno` referente a `'Maria da Graça'`. A segunda consulta retorna os identificadores, os nomes e o intervalo de validade dos alunos que alguma vez moraram no endereço `'Protásio Alves, 256'` (`endereço` é um atributo temporalizado).

[3]OQL permite a utilização do símbolo "→"como sinônimo para ponto ".". Como convenção, este trabalho adota a notação ".", pois em linguagens de programação esses dois operadores apresentam significados diferentes.

1.
```
SELECT a.iLifetime, a.fLifetime
FROM Aluno a
WHERE nome = 'Maria da Graça';
```

2.
```
SELECT EVER a.tvOID, a.nome, a.endereco.vInterval
FROM Aluno.versions a
WHERE endereco = 'Protásio Alves, 256';
```

3.5 Considerações Finais

Este capítulo apresentou o Modelo Temporal de Versões (TVM) e sua linguagem de consulta (TVQL). A linguagem de consulta foi estendida a fim de permitir a representação de expressões de caminho, característica essencial para recuperação de objetos e versões pela linguagem de atualização de objetos, proposta no capítulo 6. Um artigo apresentando um estudo sobre a viabilidade de gerenciar a evolução de esquemas conceituais tendo como base o TVM foi publicado nos anais do *XVI Simpósio Brasileiro de Banco de Dados (SBBD 2001)* (ROMA et al., 2001). Tal análise inicial considera o versionamento linear que permite a existência de uma única versão de esquemas em cada instante de tempo. Essa análise mostrou a possibilidade de combinar evolução de esquemas com o modelo temporal de versões, representando a principal motivação para a realização deste trabalho.

Diversos trabalhos, como, por exemplo (GRANDI; MANDREOLI, 2003; RODDICK et al., 2001; RODRÍGUEZ; OGATA; YANO, 1999a), têm buscado propor soluções para a questão de evolução de esquemas em bancos de dados orientados a objetos, com a utilização de um modelo temporal de objetos. Nesse caso, o mecanismo de evolução de esquemas e o modelo de objetos são considerados ortogonais, ou seja, são manipulados independentemente. Considerando que modificações sempre acontecem ao longo do tempo, de certa forma o tratamento independente não permite a utilização plena das características temporais, uma vez que o mecanismo de gerenciamento de mudanças de esquemas poderia usufruir das funcionalidades do modelo temporal. Em outra abordagem, (GORALWALLA et al., 1997, 1998), é proposto um método para gerenciamento de modificação de esquemas que explora as funcionalidades de um modelo temporal, chamado TIGUKAT (PETERS, 1994; ÖZSU et al., 1995). Entretanto, nessa abordagem o gerenciamento é realizado através de evolução, na qual somente a modificação mais recente permanece acessível.

O modelo proposto nesta tese é mais genérico, uma vez que trata as modificações de esquemas como um processo de versionamento, ao invés de evolução, aproveitando todas as potencialidades do TVM, que engloba, além das características temporais, o conceito de versão. Os próximos capítulos apresentam a especificação detalhada do modelo de evolução de esquema proposto nesta tese. A característica ímpar do modelo é a incorporação do TVM tanto no nível de esquemas quanto no nível de objetos. O TVM é utilizado para controlar não somente o versionamento de esquemas, mas também o armazenamento da extensão e a propagação das mudanças nas instâncias.

4 ARQUITETURA PARA O VERSIONAMENTO TEMPORAL DE ESQUEMAS

O suporte à evolução de esquemas tem sido uma característica essencial em sistemas de bancos de dados por permitir a execução dinâmica das aplicações, enquanto atualizações de esquemas são realizadas em bancos de dados populados, tendo repercussão imediata ou tardia na base de dados. Entretanto, não existe uma norma-padrão que regularize os procedimentos que devem ser seguidos durante a evolução de esquemas.

Este capítulo apresenta uma visão geral da solução proposta para a evolução dinâmica de esquemas em bancos de dados temporais orientados a objetos. A principal contribuição é a definição das estratégias que regem o processo de evolução de esquemas no ambiente TVM em todos os seus aspectos: evolução de esquemas, propagação de mudanças nos objetos e atualização de dados. A solução proposta apresenta seu diferencial na incorporação do TVM tanto no nível de esquemas quanto no nível de objetos.

A escolha de uma abordagem temporal é justificada pela flexibilidade que provê ao versionamento de esquemas, não somente por permitir aos usuários e programas aplicativos acesso a informações temporais (passado, presente e futuro), mas também, e principalmente, por permitir atualizações pró-ativas e retroativas entre as diversas versões de esquemas existentes. Além disso, a união dos conceitos de versão e tempo permite o armazenamento de diversas versões de esquemas, como também, para cada versão de esquema, o histórico de suas modificações.

O modelo proposto é chamado de **TVSE**, uma sigla para **T***emporal and* **V***ersioning Model to* **S***chema* **E***volution* (Modelo Temporal de Versionamento com Suporte à Evolução de Esquemas).

Os capítulos 5 e 6 especificam detalhadamente as etapas da solução proposta nesta tese e o capítulo 7 apresenta a semântica operacional que formaliza as linguagens do Modelo TVSE. Antes disto, este capítulo apresenta uma visão geral do processo de evolução de esquemas, através de um estudo de caso, com o objetivo de facilitar a compreensão do TVSE. Essa visão geral não tem o intuito de ilustrar todos os detalhes do TVSE, para não tornar exaustivo o número de exemplos a serem discutidos. Ao invés disso, são apresentados os casos mais freqüentes para que o funcionamento geral possa ser entendido. Primeiramente, é apresentado um estudo de caso real, que servirá para ilustrar o TVSE ao longo do texto. A seguir, a arquitetura proposta para o modelo é ilustrada. Por fim, cada módulo da arquitetura é discutido e exemplificado separadamente.

Um artigo apresentando a visão geral do TVSE foi publicado nos anais do *Workshop de Teses e Dissertações em Bancos de Dados*, no escopo do *XVIII Simpósio Brasileiro de Banco de Dados* (SBBD2002) (GALANTE; SANTOS; EDELWEISS, 2002).

4.1 Estudo de Caso: Sistema Acadêmico da UFRGS

Esta seção descreve parte da experiência do estudo da evolução do Sistema de Controle Acadêmico da UFRGS (denominado Sistema Discente), previamente apresentado na seção 3.2.9. O estudo de um sistema em execução tem o intuito de analisar problemas significativos, baseado em dimensões reais, visando a garantir a aplicabilidade e a adequação do modelo a ser definido. O estudo de caso está publicado em (GALANTE; SANTOS; RUIZ, 1998) e a especificação completa pode ser encontrada em (GALANTE, 1998).

4.1.1 Fases de Evolução

A implementação do Sistema Discente teve início em 1972 e, desde então, mantém o registro eletrônico de todos os alunos que tiveram vínculo com a universidade, seus cursos e respectivos currículos. O histórico das turmas oferecidas semestralmente teve início somente em 1994.

A evolução do Sistema Acadêmico pode ser dividida em quatro fases, caracterizando sua evolução ao longo do tempo: (*i*) 1973 - o sistema teve início com um esquema de banco de dados hierárquico, dividindo-se em três subsistemas: Controle de Alunos, Currículos e Turmas; (*ii*) 1987 - o módulo matrícula foi desenvolvido e incorporado ao sistema; (*iii*) 1995 - alterações foram realizadas procurando atender às demandas de um modelo relacional, porém diversas características hierárquicas foram mantidas; e (*iv*) 1997 - todo sistema foi reimplementado, caracterizando-se como um banco de dados expressamente relacional.

Cabe salientar que cada fase de evolução caracterizou-se pela definição de um novo esquema para o banco de dados e pela troca de um sistema implementado por outro. Observa-se uma evolução gradativa na passagem de um sistema de banco de dados hierárquico para uma modelagem relacional.

A passagem de uma fase para outra foi realizada em partes. Em um primeiro momento, houve apenas a vigência do sistema antigo. Logo a seguir, as duas estruturas funcionaram em paralelo. Finalmente, foi realizada a migração completa para o novo sistema. A estrutura antiga foi abandonada, passando a serem utilizadas apenas as definições do novo esquema.

Durante a fase intermediária, na qual as duas estruturas vigoraram em paralelo, apenas uma interface permaneceu visível aos usuários. Primeiro, a interface se apresentou através do antigo sistema. Alguns módulos pertenciam à nova estrutura, mas o controle era realizado internamente pelo sistema antigo. No segundo passo, a interface foi transferida ao novo sistema. O controle interno passou a ser realizado pelo sistema novo, sendo alguns módulos do sistema antigo ainda executados.

4.1.2 Aplicação do TVM no processo de evolução de esquemas do Sistema Discente

A seção 3.2.9 apresentou um estudo de caso ilustrando a modelagem do Sistema Discente com o uso do TVM (a Figura 3.10 ilustra o esquema atual do sistema). A partir desse estudo de caso, algumas comparações podem ser feitas a respeito da utilização do TVM no gerenciamento de evolução de esquemas.

Segundo o estudo realizado, constatou-se que para cada curso existe um currículo vigente com validade ano/semestre, o qual, após encerrado, não pode mais ser aberto para alterações. Assim, a programação de currículos pode ser comparada a objetos versionados, pois a cada ano uma nova versão do currículo é gerada para cada curso.

A seleção do currículo vigente é semelhante à escolha de uma versão corrente, sendo a versão corrente utilizada quando o objeto é solicitado sem a especificação de uma de suas versões. No Sistema Discente, a cada ano/semestre apenas um currículo permanece atual para cada curso. E, ainda, o encerramento do currículo pode ser comparado à passagem da versão para o estado consolidado ou desativado, pois, após encerrado, não sofre mais alterações.

A heurística utilizada na implementação dos relacionamentos de generalização/especialização foi herança por refinamento, na qual os atributos migram para as entidades mais especializadas. Essa abordagem foi selecionada pela necessidade de limitar o número de tabelas, acomodando os dados nas entidades mais especializadas, facilitando, assim, o posterior acesso. Entretanto, essa foi uma decisão de projeto, podendo, tranqüilamente, ter sido escolhido um outro tipo de abordagem, como, por exemplo, relacionamento de herança por extensão, no qual as propriedades permanecem no nível em que foram definidas. Essa característica, presente no TVM, facilita o manuseio de evolução de esquemas, possibilitando modificar arbitrariamente determinados níveis de abstração da estrutura de dados sem se preocupar com os demais, garantindo a coerência e a validade de correspondência entre eles.

A grande dificuldade existente no processo de evolução caracterizou-se pelo fato de a evolução se dar pela substituição do banco de dados, esquema e aplicação, tornando impossível retornar a um estado anterior após realizada a transição. A utilização de mecanismos de versões, além de manter todo o histórico da aplicação, permite voltar a situações anteriores à evolução, proporcionando maior segurança e flexibilidade ao processo de evolução de esquemas.

O uso de versões permite, também, trabalhar com esquemas concorrentes, possibilitando o estudo e a análise de esquemas paralelos, ajudando na escolha da melhor forma de se tratar um padrão de evolução em questão.

Versões e tempo possibilitam ainda uma melhor documentação do histórico da base de dados existente, pois, juntamente com os dados, ficam armazenadas as versões de seus esquemas correspondentes, permitindo a verificação de todas as restrições e a semântica existente para as informações armazenadas em cada época. Por exemplo, a compreensão total de todas as mudanças sofridas pelo Sistema Discente representa um árduo trabalho, pois é possível definir etapas e analisá-las através da documentação existente, mas não identificar alterações realizadas em etapas intermediárias às fases definidas devido à substituição das mudanças realizadas e à conversão da base de dados.

4.2 Arquitetura Proposta para o Modelo Temporal de Versionamento com suporte a Evolução de Esquemas

O processo de evolução de esquemas do TVSE tem por objetivo propor uma camada intermediária entre um SGBD e as aplicações dos usuário, de forma que o usuário possa especificar suas aplicações em um ambiente que gerencia as mudanças dinâmicas que ocorrem no banco de dados durante a vigência das aplicações. Esta camada tem dois objetivos principais: realizar o gerenciamento da modificação de esquemas e da propagação de mudanças, e possibilitar a atualização dos dados durante o processo de evolução.

A Figura 4.1 ilustra, simplificadamente, a arquitetura proposta para o modelo. Em linhas gerais, o gerenciamento da evolução de esquemas pode ser dividido em quatro módulos:

- Gerenciamento de Versões - representa o núcleo do processo de versionamento esquemas, gerenciando toda a história da evolução das versões de esquemas. Esse módulo é responsável por especificar todas as normas que regem o processo de derivação de versões de esquemas, bem como as regras temporais associadas às versões. Esse módulo está completamente especificado na seção 5.1;
- Gerenciamento de Evolução de Esquemas - controla o gerenciamento das modificações realizadas no esquema, bem como a repercussão de tais mudanças na base de dados. Esse processo é separado em dois módulos distintos:

  – Gerenciamento de Modificação de Esquemas - controla o gerenciamento das modificações de esquema, definindo uma taxonomia de operações de mudança aplicáveis aos esquemas e restrições de integridade que garantem que tais operações preservam a integridade das versões de esquemas. Esse módulo está completamente especificado na seção 5.2;
  – Gerenciamento de Propagação de Mudanças - interage com o módulo de modificação de esquemas, especificando as estratégias para atualização da base de dados de acordo com as modificações realizadas no esquema. Esse módulo é responsável por garantir a integridade entre as instâncias e seus esquemas associados, e está completamente especificado na seção 5.4;

- Gerenciamento de Atualização de Dados - trata a semântica da atualização de dados (`insert`, `update` e `delete`) de forma a garantir o sincronismo entre a disponibilidade dos dados e as propagações de mudança nos níveis de esquemas e dados. Esse módulo está completamente especificado no capítulo 6;
- Gerenciamento de Armazenamento de Dados - controla o armazenamento das versões de esquemas e seus dados associados. Esse módulo está dividido em duas partes, sendo detalhamente especificado na seção 8.2:

  – Metadados - mantém informações a respeito da evolução dos esquemas e de suas respectivas classes, atributos, métodos e relacionamentos, servindo de base para todo processo de evolução e versionamento de esquemas;
  – Intenção e Extensão - é o banco de dados que armazena as aplicações dos usuários e seus dados associados.

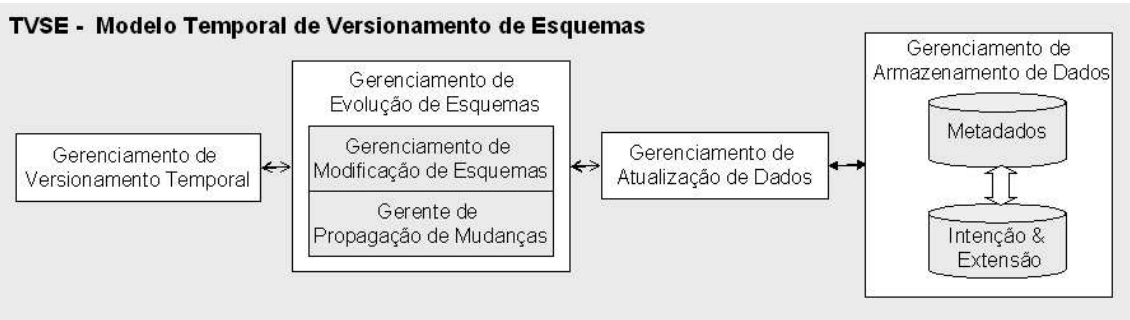


Figura 4.1: Arquitetura para o Modelo Temporal de Versionamento com suporte a Evolução de Esquemas

Cada um desses módulos será exemplificado na próxima seção. Cabe salientar que o núcleo fundamental do trabalho consiste na incorporação do TVM tanto no nível de

esquemas quanto no nível de dados. Os conceitos de tempo e versão são utilizados para controlar não somente o versionamento de esquemas, mas também o armazenamento das instâncias e a propagação de mudanças. Nesse sentido, o TVM foi escolhido por modelar, de forma completa e concisa, os requisitos exigidos pelo processo de evolução de esquemas, conduzindo a uma mais completa especificação do modelo proposto.

4.3 Visão Geral do Processo de Versionamento Temporal de Esquemas

Esta seção apresenta uma visão geral do processo de versionamento temporal de esquemas, através de uma série de exemplos de modificações, aplicada no Sistema Discente durante seu processo de evolução. O objetivo principal é delinear os requisitos básicos exigidos pelo mecanismo de versionamento temporal de esquemas, de forma que o princípio geral de funcionamento seja entendido. A evolução do Sistema Discente é utilizada como exemplo, a fim de esclarecer e ilustrar cada um dos módulos propostos na arquitetura, bem como as interações existentes com o usuário. Os exemplos apresentados nesta seção serão utilizados ao longo de todo texto para exemplificação do modelo proposto.

4.3.1 Versionamento de Esquemas

O gerente de evolução de esquemas estabelece os critérios de derivação de versões e as normas de manipulação de tais versões pelos usuários (permitindo, nesse caso, a manipulação diretamente nas versões do esquema conceitual). Cabe salientar que as operações de modificação (diferentemente da maioria das propostas presentes na literatura) não representam o único caminho de derivação de versões de esquemas. Ao invés disso, o usuário tem a liberdade de interferir no processo de versionamento, guiando a evolução a partir de critérios próprios requeridos pela aplicação modelada. Assim, o gerente de versionamento temporal possui operações de manipulação de versões que permitem ao usuário derivar, excluir e instanciar versões de esquema, bem como realizar consultas envolvendo diversas versões de esquemas e de objetos.

A Figura 4.2 ilustra uma modificação de esquema, na qual a evolução é controlada manualmente pelo usuário. A Figura 4.2-a ilustra a primeira versão do Sistema Discente que inclui os módulos de Alunos e Currículo. A Figura 4.2-b[1] ilustra a inclusão de um módulo completo no sistema (Turmas). O novo módulo foi incluído tendo como base a primeira versão do esquema (Figura 4.2-a), acarretando a derivação de uma nova versão de esquema (`'Esquema,2'`). Nesse caso, o `'Esquema,2'` é uma cópia do esquema-base, ficando disponível para que o usuário possa fazer as modificações que julgar necessárias. Nesse exemplo em particular, todo módulo de programação de turmas foi especificado, sem que cada operação atômica de modificação levasse à derivação de uma nova versão do esquema. Essa facilidade permite controlar a excessiva proliferação de versões, ou seja, quando o usuário julgar conveniente, pode controlar manualmente o processo de derivação de versões de esquemas. Caso contrário, o gerenciamento é efetuado automaticamente pelo sistema, conforme apresentado na próxima seção. A Figura 4.2-c mostra o grafo de derivação das versões de esquema.

No gerente de versionamento temporal de esquemas são especificadas as regras

[1]Essa figura foi previamente apresentada no capítulo 3 (Figura 3.10), sendo repetida na Figura 4.2-b para facilitar a leitura do texto.

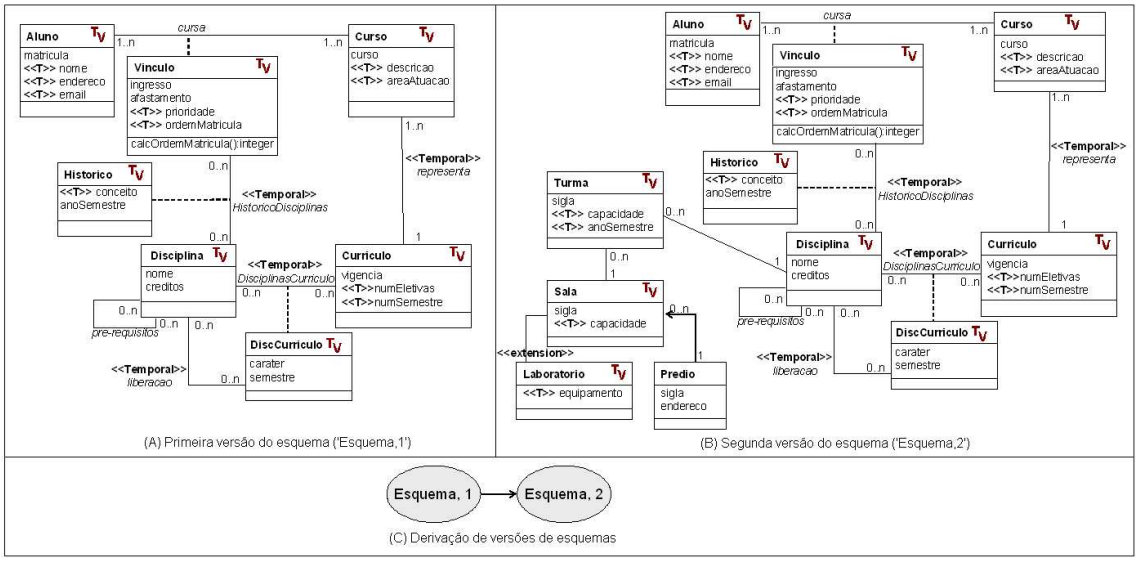


Figura 4.2: Exemplo da evolução do Sistema Acadêmico - Etapa I

de integridade que permitem associar o tempo aos esquemas, suas versões, classes, atributos, métodos e relacionamentos, permitindo uma modelagem mais natural e flexível da realidade. Como o modelo é proposto para banco de dados bitemporais, o banco de dados incorpora para cada versão de esquema o tempo de transação e o tempo de validade, permitindo a existência de mais de uma versão de esquema em operação no mesmo instante de tempo.

4.3.2 Evolução de Esquemas

O processo de evolução de esquemas está dividido em duas etapas distintas: (*i*) gerenciamento das modificações de esquemas e derivação de versões; e (*ii*) atualização da base de dados a fim de acomodar as modificações realizadas no esquema.

4.3.2.1 Gerenciamento de Mudanças

O processo de gerenciamento de mudança inicia-se sempre com uma operação de modificação de esquema pré-definida pelo modelo, que sempre acarreta a derivação de uma nova versão de esquema.

O exemplo ilustrado na Figura 4.3 possui três versões que representam a história da evolução de um esquema conceitual, após sucessivas modificações aplicadas na estrutura do esquema. Tomando como base a primeira versão do esquema (`'Esquema,1'` - Figura 4.2-a), as seguintes operações de modificação são aplicadas na estrutura do esquema:

- um novo atributo é incluído na classe curso (`perfilEgresso`), da primeira versão do esquema (`'Esquema,1'`), acarretando a derivação de uma nova versão para o esquema (`'Esquema,3'`), conforme ilustrado na Figura 4.3-a;
- uma classe associativa (`Histórico`) é excluída da terceira versão do esquema (`'Esquema,3'`), acarretando a derivação de uma nova versão para o esquema (`'Esquema,4'`), conforme ilustrado na Figura 4.3-b.

O foco das modificações está em duas operações básicas, a saber: `excluir` e `incluir`. Essas operações foram selecionadas porque podem afetar a estrutura do

esquema por caminhos diferentes, como, por exemplo, classes, atributos, métodos e relacionamentos. A operação de alteração não aparece no exemplo, pois sua semântica é a execução da seqüência dessas duas operações mencionadas.

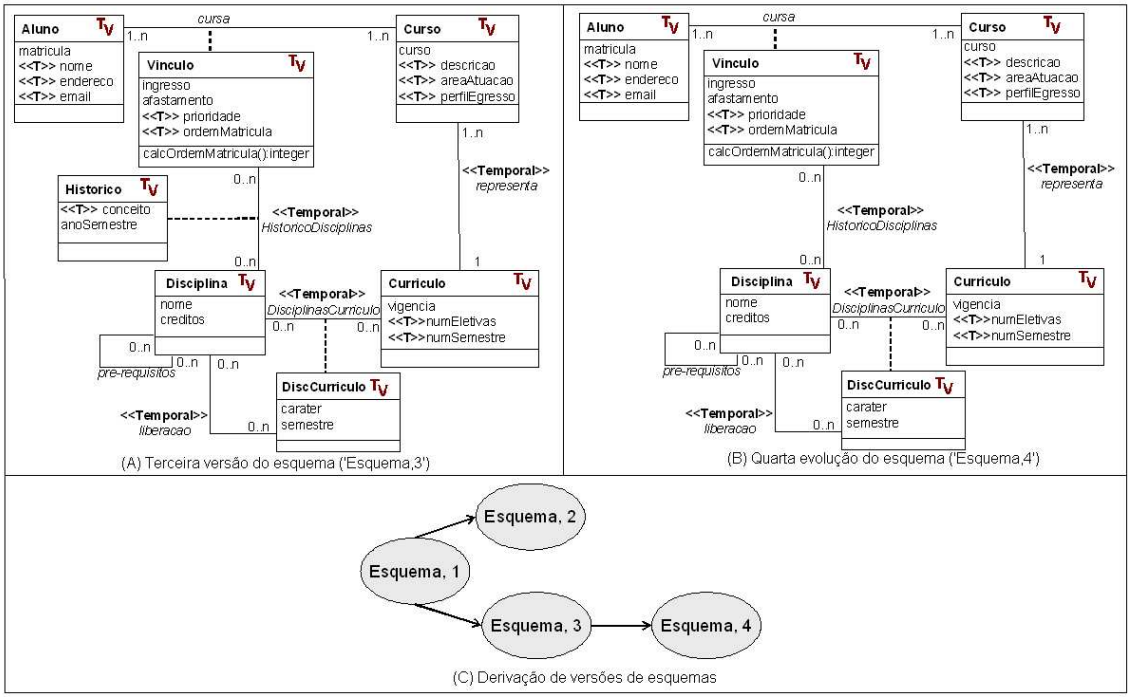


Figura 4.3: Exemplo da evolução do Sistema Acadêmico - Etapa II

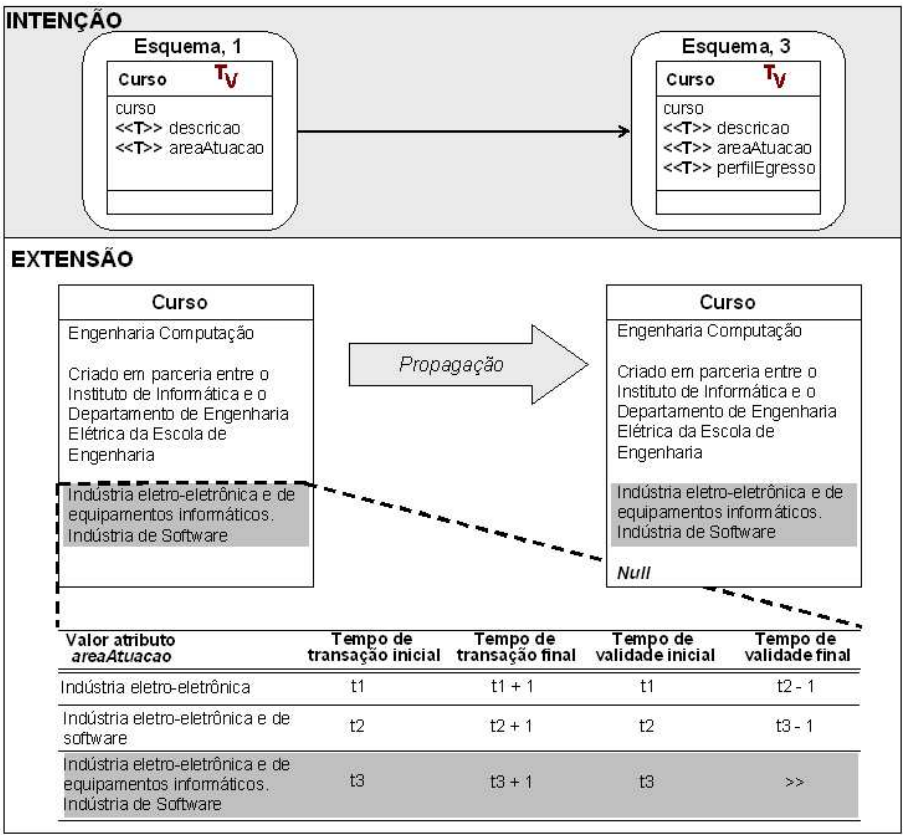
A figura 4.3-c apresenta o grafo de derivação de versões, representado o histórico de evolução das versões de esquema. A evolução pode ser realizada de duas formas: (*i*) gradativa (por exemplo, `'Esquema,1'`, `'Esquema,3'` e `'Esquema,4'`), em que cada operação de modificação leva à derivação de uma nova versão do esquema; ou (*ii*) manual (por exemplo, `'Esquema,1'` e `'Esquema,2'`) na qual o usuário utiliza as operações de manipulação de versões. Cabe salientar ainda que as versões podem ser criadas uma após a outra, gerando uma evolução (por exemplo, `'Esquema,1'`, `'Esquema,3'`, `'Esquema,4'` ou `'Esquema,1'`, `'Esquema,2'`), ou em paralelo (por exemplo, `'Esquema,2'` e a seqüência de esquemas `'Esquema,3'`, `'Esquema,4'`), representando projetos alternativos.

4.3.2.2 Propagação de Mudanças

Realizadas as modificações no esquema conceitual, as instâncias vigentes no banco de dados precisam ser adaptadas às novas especificações para manter a coerência entre a estrutura definida e os dados armazenados. No processo de propagação de mudanças, são definidos os mecanismos que atualizam os objetos persistentes da base de dados para que respeitem as novas definições impostas pela estrutura do esquema.

O processo de propagação de mudanças é realizado explicitamente pelo usuário através de uma operação de ativação. Quando o usuário julgar que a versão de esquema está pronta para uso, a operação de ativação cria a base de dados e realiza a propagação dos dados. Para tanto, funções de adaptação são definidas a fim de adequar as informações presentes na base de dados com a requerida versão de esquema, acionadas no instante da ativação do esquema. Para cada operação de modificação existe um procedimento padrão

que será adotado no processo de propagação, podendo ser alterado pelo usuário quando este julgar necessário.



Valor atributo *areaAtuacao*	Tempo de transação inicial	Tempo de transação final	Tempo de validade inicial	Tempo de validade final
Indústria eletro-eletrônica	t1	t1 + 1	t1	t2 - 1
Indústria eletro-eletrônica e de software	t2	t2 + 1	t2	t3 - 1
Indústria eletro-eletrônica e de equipamentos informáticos. Indústria de Software	t3	t3 + 1	t3	>>

Figura 4.4: Exemplo de Propagação de Mudanças

A Figura 4.4 mostra a representação gráfica da propagação de mudanças do `Esquema,1` para o `Esquema,3`, conforme ilustrado previamente na Figura 4.3. Para simplificar, apenas a classe `Curso` é apresentada no exemplo. A inserção do atributo `perfilEgresso` desencadeou a derivação de uma nova versão de esquema (`Esquema,3`). O `Esquema,1` possui uma instância do curso `Engenharia da Computação`, sendo apresentado também o histórico do atributo temporal `areaAtuacao`. Observa-se a existência de três descrições para o atributo `areaAtuacao`, sendo o valor corrente representado pelo tempo de validade final em aberto. O processo de propagação inicia-se no momento em que a operação de ativação é aplicada no `Esquema,3`. Os dados da versão que serviu de base para derivação (`Esquema,1`) são copiados e associados ao novo esquema (`Esquema,3`). Nesse caso, somente os valores correntes dos atributos e relacionamento temporais são propagados. Por exemplo, apenas o último valor do atributo `areaAtuacao` foi associado ao novo esquema. O valor nulo é atribuído ao novo atributo criado (`perfilEgresso`), representando a operação de modificação realizada.

Na presença de versionamento de esquemas, a base de dados extensional deve estar armazenada para garantir flexibilidade no acesso e na atualização. O modelo de evolução de esquemas proposto nesta tese é analisado frente aos dois mecanismos de armazenamento existentes: repositório único (RODDICK, 1991) e múltiplos repositórios (CASTRO; GRANDI; SCALAS, 1997). Cabe salientar ainda que a extensão de dados pode armazenar tanto objetos convencionais como temporais versionados (TVM),

dependo da modelagem proposta para aplicação.

4.3.3 Manipulação dos Dados

O objetivo do processo de atualização de dados é permitir ao usuário inserir, alterar e remover dados, durante todo processo de modificação e versionamento de esquemas. A principal motivação é a incapacidade das linguagens convencionais de atualizar os dados em um ambiente com múltiplas versões de esquemas, que permanece em constante evolução. Além disso, os comandos de atualização exigem o tratamento adequado dos rótulos temporais associados aos esquemas e aos dados. Por exemplo, a remoção do endereço de um determinado aluno poderia envolver a atualização dos dados em mais de uma versão de esquema. Esse tipo de atualização é conhecido como atualização multi-esquema e permite ao usuário atualizar instâncias pertencentes a mais de uma versão de esquema ao mesmo tempo.

Nesta tese é proposta uma linguagem multi-esquema para atualização de dados durante o processo de evolução, sendo também levantadas as premissas para uma linguagem de consulta que trate com versionamento temporal de esquema e gerência de dados temporais versionados. Nesse caso, é desejável que a linguagem de consulta estabeleça um comportamento o mais homogêneo possível frente às diversas versões de esquema e objetos versionados. Porém, o processamento de consulta está fora da área de abrangência desta tese.

4.4 Considerações Finais

Este capítulo apresentou inicialmente um estudo de caso real a fim de justificar a importância do modelo proposto, contextualizando de forma prática. Cabe salientar que todas as transições efetuadas no Sistema Discente foram realizadas de forma empírica, visando a suprir os requisitos impostos pela necessidade de evolução. Todo o processo de evolução do esquema foi executado manualmente, a partir do levantamento realizado pelo grupo de projeto do CPD. As demandas de evolução basearam-se nas necessidades provenientes dos usuários, nas carências apresentadas pelo sistema no decorrer de sua existência e nos eventuais problemas que surgiram durante o desenvolvimento dos trabalhos. Com o objetivo de facilitar o processo de evolução e garantir uma maior segurança às modificações efetuadas, ficou evidente a necessidade de um mecanismo de evolução de esquemas que proporcionasse um eficaz e rápido processo de evolução, capaz de guiar as alterações, assegurando a validade da semântica especificada para o sistema, como também a consistência da base de dados e sua coerente correspondência com o esquema definido. Além de auxiliar fortemente as fases de evolução, a definição de padrões de modificações permitiria uma melhoria na qualidade do trabalho.

Em seguida, a arquitetura do TVSE foi apresentada com o intuito de delinear os objetivos propostos para o trabalho, utilizando exemplos esclarecedores que ilustrassem cada um dos módulos propostos e as interações com o usuário. Encerrada esta visão geral, os capítulos 5, 6 e 7 especificam detalhadamente as etapas do modelo de evolução de esquemas proposto.

5 MODELO TEMPORAL DE VERSIONAMENTO COM SUPORTE A EVOLUÇÃO DE ESQUEMAS

Neste capítulo, o Modelo Temporal de Versionamento com Suporte à Evolução de Esquemas (TVSE) é proposto como forma de guiar as alterações em bancos de dados temporais orientados a objeto. As idéias apresentadas são adaptáveis a vários sistemas de banco de dados, pois os mecanismos são propostos de forma genérica e não definidos para um sistema específico. A principal motivação advém do problema de gerenciar as modificações em esquemas conceituais enquanto o banco de dados permanece disponível, buscando causar o mínimo impacto possível nas aplicações existentes.

O TVSE tem por objetivo conduzir as modificações realizadas em sistemas de banco de dados orientados a objetos, mantendo o histórico das alterações através dos conceitos de tempo e de versão. O estado anterior às transformações é mantido, permitindo a navegação retroativa e pró-ativa entre as versões de esquema, para realização de operações de alteração e consulta na base de dados extensional. O modelo assegura, ainda, a integridade das instâncias armazenadas no banco de dados em qualquer perspectiva de versão sob a qual são requeridas. Em linhas gerais, o TVSE é resultado da união entre o Modelo de Evolução de Esquemas (GALANTE, 1998) e o Modelo Temporal de Versões (MORO, 2001). Essa união representa o diferencial do TVSE à medida em que o histórico da evolução da base de dados intencional e extensional é mantido através da utilização de versionamento temporal.

As próximas seções se especifica detalhadamente a arquitetura geral proposta no capítulo 4. Primeiramente, são descritos os mecanismos de versionamento de esquema e a representação temporal. A seguir, o gerenciamento das modificações de esquema é especificado, definindo uma taxonomia de operações de mudança bem como as restrições de integridade que visam a garantir a validade das versões de esquema frente às modificações realizadas. Uma linguagem para versionamento e modificação de esquemas é também especificada, bem como o mapeamento dessa linguagem para ODMG. Por fim, o mecanismo de propagação de mudanças é exposto, definindo como as instâncias devem ser modificadas para se tornarem consistentes com as novas versões do esquema, representando os efeitos das mudanças para as aplicações.

Uma visão geral do TVSE foi publicada nos anais da *XXVIII Conferencia Latinoamericana de Informática* (GALANTE; EDELWEISS; SANTOS, 2002a), contextualizando as etapas envolvidas no processo de evolução de esquemas. O enfoque é dado à especificação dos requisitos de versionamento de esquemas e de gerenciamento temporal.

Uma versão preliminar do TVSE está presente em um artigo submetido em fevereiro deste ano à revista *Data & Knowledge Engineering* (GALANTE; SANTOS;

EDELWEISS, 2003). O artigo se encontra no primeiro processo de revisão, iniciado em novembro de 2003.

5.1 Versionamento Temporal de Esquemas

Esta seção especifica os requisitos de versionamento e gerenciamento temporal propostos para o TVSE, considerando um ambiente para evolução de esquemas. A utilização de versionamento temporal permite gerenciar a história de evolução dos esquemas e de seus dados associados. O aspecto inovador diz respeito ao gerenciamento de versões temporais tanto no nível intencional quanto no extensional do banco de dados.

5.1.1 Versões de Esquema e Esquema Versionado

Durante o desenvolvimento de um projeto de esquema, a evolução é mantida através do mecanismo de versionamento de esquemas. Um esquema que possui versões é representado através de um esquema versionado, que é o agrupamento de todas as suas versões, de tal forma que cada uma delas pode ser utilizada onde o esquema versionado é requerido. O esquema versionado possui características próprias, bem como propriedades que podem ser comuns a todas as suas versões.

As versões de um esquema versionado estão relacionadas através de um relacionamento de derivação, formando um grafo acíclico dirigido, no qual uma versão pode ser derivada de uma ou mais existentes. A Figura 5.1 ilustra graficamente um esquema sem versões (esquema convencional) e um esquema versionado com suas respectivas versões (SV1, SV1, SV2, SV3, SV4, SV5).

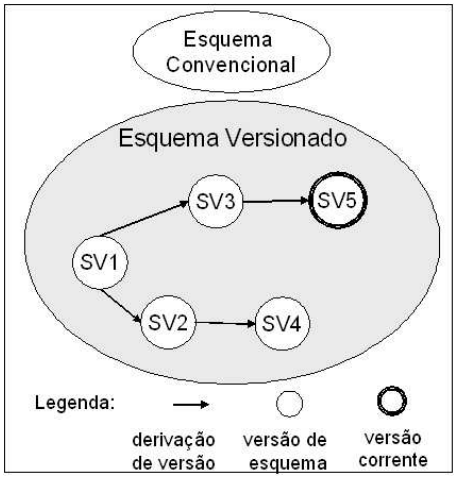


Figura 5.1: Esquema versionado e suas versões

Cada esquema versionado possui uma versão que é considerada a versão corrente. A versão corrente é mantida como a mais recentemente criada. O usuário pode especificar um versão diferente para ser a corrente e, neste caso, a versão deve ficar fixa, não mudando com o surgimento de novas versões. A versão corrente é utilizada sempre que o usuário solicita uma operação sobre um esquema versionado, sem especificar uma de suas versões. Na Figura 5.1 a versão corrente é ilustrada através de linhas duplas (versão SV5).

5.1.1.1 Derivação de Versões de Esquemas

A criação de uma nova versão de esquema pode ser explícita, quando o controle é realizado pelo usuário, ou implícita, quando derivada automaticamente pelo sistema.

A derivação implícita é a operação adotada por *default* e é utilizada para retratar a seqüência histórica de evolução do esquema. Uma nova versão é derivada em decorrência de operações de modificação aplicadas sobre o esquema ou para fins de adaptação das instâncias presentes no banco de dados. Quando uma versão de esquema é derivada em decorrência de modificação de esquemas, a versão sucessora é uma cópia da primeira, sendo acrescida a alteração realizada.

A derivação explícita de versões é especificada pelo usuário e representa a cópia de uma (ou mais) versão antecessora do grafo de derivação, permanecendo livre para modificações e testes. Nesse caso, todo controle deve ser realizado manualmente pelo usuário.

5.1.1.2 Granularidade do Versionamento

O TVSE adota a técnica de versões alternativas, isto é, as versões permanecem armazenadas em um espaço comum, evoluem em paralelo e operam sobre a mesma coleção de dados.

Três abordagens são consideradas quanto à granularidade do versionamento, (entretanto as versões são tratadas de maneira uniforme, sendo consideradas objetos versionáveis):

- versões de esquema - são derivadas em decorrência de modificações realizadas na estrutura do esquema. Por exemplo, a eliminação de uma classe desencadeia a derivação de uma versão completa do esquema, representando a atualização aplicada;

- versões de objetos - são definidas para fins de adaptação das instâncias presentes no banco de dados, para que permaneçam em conformidade com as definições especificadas nas diversas versões de esquemas.

Sendo assim, uma versão de esquema especifica um esquema completo num dado instante de tempo, no qual as alterações realizadas servem como um ponto de partida para sua evolução durante o ciclo de vida do projeto.

5.1.2 Especificação dos Requisitos Temporais

O TVSE adota versionamento bitemporal que permite a manutenção de todas as versões com seus tempos de validade, ao longo do tempo de transação. A escolha da representação bitemporal é justificada pela possibilidade de mudanças retro e pró-ativas, que mantêm o registro dessas mudanças. O fato de que diversas versões de esquema podem evoluir em paralelo, estando disponíveis no mesmo instante de tempo, acarreta a possibilidade de duas ordens de tempo: ramificada para o esquema versionado; e linear para suas versões.

O tempo é associado ao esquema versionado e suas versões, permitindo uma modelagem mais natural e flexível da realidade. O tempo de vida de um esquema versionado é representado pelo atributo `alive`. Quando um esquema é criado, o atributo *alive* recebe valor *true*, sendo armazenado também o seu tempo de validade. O atributo `alive` recebe valor *false* no momento em que uma exclusão lógica é realizada.

Para visualizar estas definições, a Figura 5.2 ilustra a linha de tempo de um esquema versionado e de suas quatro versões: `SV1`, `SV2`, `SV3` e `SV4`, respectivamente. São apresentados os instantes iniciais do atributo `alive` (=T_0) e de cada uma de suas versões (T_1, T_2, T_3 e T_4), tornando claro o conceito de ordem de tempo ramificado. O atributo `alive` final é mostrado para a terceira versão (`SV3`), ilustrando que essa versão foi excluída no instante $t_{3.end}$ (`alive`=*false*).

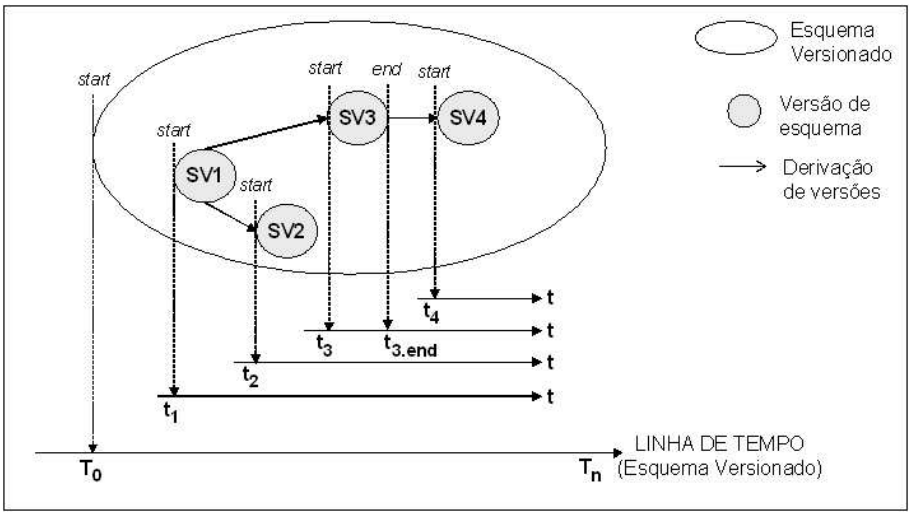


Figura 5.2: Versionamento de esquemas bitemporal

A temporalidade é representada através de elementos bitemporais: conjunto disjunto de intervalos temporais, tanto para o tempo de transação como para o tempo de validade. Assim, as versões de esquemas estão associadas com os rótulos pré-definidos `tTimei`, `tTimef`, `vTimei`, `vTimef`, que representam respectivamente, os tempos de transação inicial e final, e os tempos de validade inicial e final. A Figura 5.3 ilustra os rótulos temporais pré-definidos associados a cada versão de um esquema versionado (note que o grafo de derivação de versões é o mesmo apresentado na Figura 5.2). É importante destacar que apesar do tempo de validade da quarta versão (`SV4`) estar contido no tempo de validade da terceira versão (`SV3`), ambas ficam registradas no histórico de derivação através do tempo de transação.

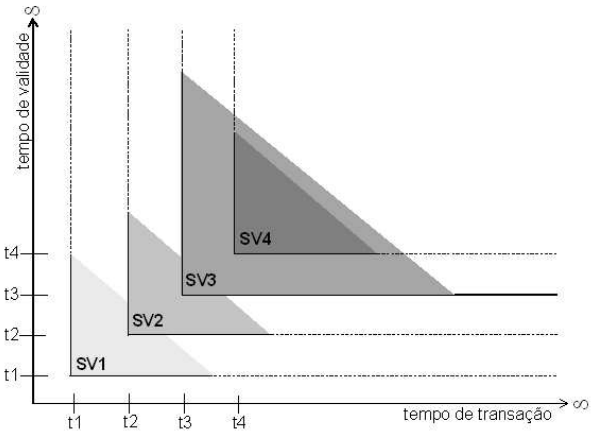


Figura 5.3: Versionamento de esquemas bitemporal

Considerando o versionamento de esquemas, cabe ressaltar que o tempo de validade está restrito ao presente e ao futuro. A hipótese de modificar o passado foi descartada por ser julgado que esse tipo de mudança não faz sentindo, uma vez que esquemas modelam a realidade e a alteração do passado não manteria fiel essa representação. Além disso, se a modificação do passado fosse permitida, o processamento de consulta não recuperaria mais a realidade, uma vez que ela foi alterada.

5.1.2.1 Restrições de Integridade Temporal

De forma análoga ao TVM, esta seção apresenta um conjunto de regras temporais a fim de garantir que os rótulos temporais associados às versões de esquema estejam contidos no tempo de vida do esquema versionado. Os rótulos temporais são abreviados: *TVI* para o tempo de validade inicial, *TVF* para o tempo de validade final, *TTI* para o tempo de transação inicial e *TTF* para o tempo de transação final.

O rótulo de tempo associado às versões deve estar contido no tempo de vida do esquema versionado. Sendo assim, definem-se as seguintes regras de integridade temporal estabelecidas por esquemas (conforme ilustrado na Figura 5.4):

1. o tempo de vida inicial (*alive.TVI*) de uma versão de esquema deve ser menor que o final (*alive.TVF*) e maior ou igual ao tempo de vida inicial (*alive.TVI*) do esquema versionado;

2. o tempo de vida final (*alive.TVF*) de uma versão de esquema deve ser maior que o inicial (*alive.TVI*) e menor ou igual ao tempo de vida final (*alive.TVF*) do esquema versionado;

3. o tempo de vida final (*alive.TVI*) de um esquema versionado deve ser maior que o inicial (*alive.TVI*) e maior ou igual ao maior tempo de vida final de suas versões (*alive.TVI*).

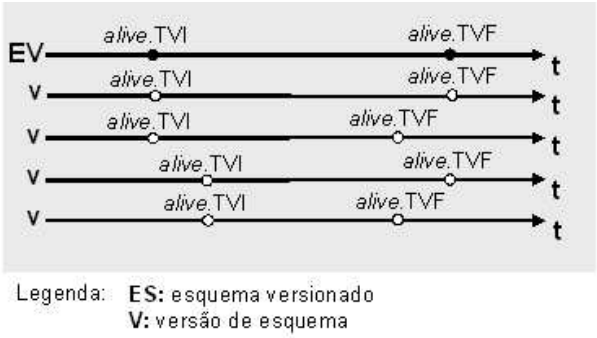


Figura 5.4: Regras de integridade temporal para esquemas

Os tempos de vida inicial e final do esquema versionado são informados pelos valores *TVI* e *TVF* do atributo *alive* quando o mesmo é *true*. Essas regras valem dentro de cada vida do esquema. As vidas de um mesmo objeto não podem ter um (ou mais) instante em comum. No momento em que um objeto excluído é restaurado, seu novo tempo de validade inicial tem que ser obrigatoriamente maior que o tempo de validade final, nem que seja por um instante temporal.

5.1.2.2 Representação do Tempo de Validade

Considerando que mais de uma versão de esquema pode ser definida em um mesmo intervalo de tempo, as seguintes possibilidades são estabelecidas, conforme ilustrado na Figura 5.5:

- totalmente sobreposto - duas versões de esquemas são definidas para o mesmo tempo de validade. Por exemplo, na Figura 5.5-a as versões *SV2* e *SV3*;
- parcialmente sobreposto - duas versões de esquema têm parte do seu tempo de validade interceptado no mesmo instante de tempo. Por exemplo, na Figura 5.5-b as versões SV2 e SV3 têm seus tempos de validade interceptados no intervalo de tempo *[t2, t3]*;
- incluso - o intervalo de validade de uma versão está totalmente contido no intervalo de validade de outra versão. Por exemplo, a Figura 5.5-c ilustra a versão *SV3* com seu tempo de validade `[t2,t3]` contido no tempo da versão *SV2*;

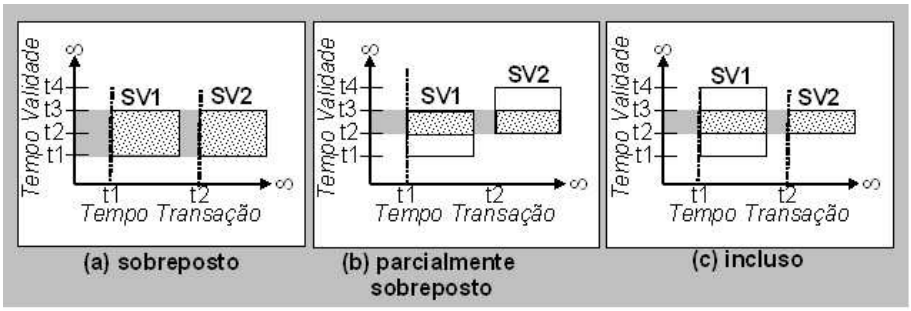


Figura 5.5: Representação do tempo de validade

5.1.3 Estados das Versões de Esquemas

De forma similar às versões de objetos, as versões de esquemas passam por estados que refletem seu estágio de desenvolvimento e/ou consistência. As transições entre os estados e os eventos que causam tais transições são ilustrados na Figura 5.6.

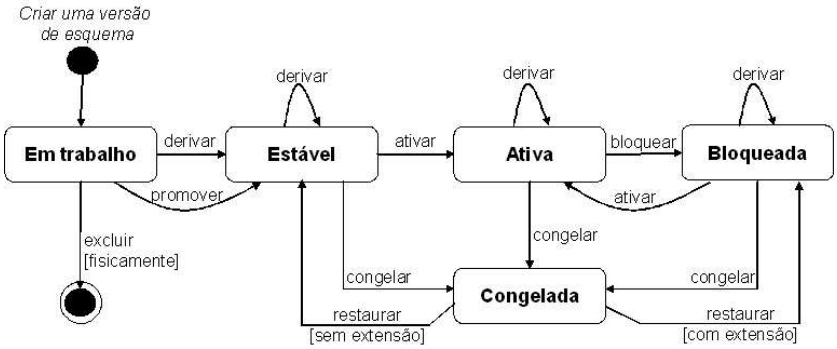


Figura 5.6: Diagrama de estados de uma versão de esquema

Cada estado define um conjunto de operações que pode ser aplicado sobre as versões de esquema, tal que o estado:

- em trabalho - representa um esquema que está em fase de criação. Toda modificação causa uma correção, não sendo mantido o histórico da evolução. Um versão neste estado pode servir como base para derivação de novas versões. Nesse caso, a versão base é promovida para o estado estável, e a nova versão é criada no estado em trabalho. A versão no estado em trabalho pode ainda ser promovida para estável, consultada e modificada. A exclusão de um esquema em trabalho é sempre física, pois esta versão não possui uma extensão de dados associada a ela;
- estável - representa um esquema consistente e completo, cujo histórico deve ser mantido. Não pode mais ser alterado, sendo que qualquer modificação deve levar à derivação de uma nova versão. Uma versão estável não possui uma extensão associada a ela. Uma versão neste estado pode servir como base de derivação de novas versões, ser consultada, congelada (exclusão lógica, somente para as versões folha), ou ainda ativada (tornada instanciável);
- ativa - é o único estado em que o esquema pode ser instanciado (instâncias em operação). Uma versão neste estado pode servir de base para derivação de novas versões, bloqueada, consultada ou ainda congelada[1];
- bloqueada - as versões bloqueadas possuem uma extensão associada a elas, porém novas instanciações não são permitidas (extensão estável). Pode servir como base de derivação de novas versões, ser consultada, ativada ou ainda congelada;
- congelada - corresponde a uma exclusão lógica, no qual a versão pode ser somente consultada ou restaurada.

Sintetizando, novas versões são criadas sempre no estado em trabalho. Quando uma versão é derivada de outra existente, suas predecessoras são automaticamente promovidas para o estado estável (se estiverem no estado em trabalho), evitando assim modificações consideradas importantes do ponto de vista histórico. Somente as versões em trabalho podem ser fisicamente modificadas e excluídas. Nos demais estados, qualquer alteração leva à derivação de uma nova versão, como sucessora da versão modificada, sendo acrescida da modificação realizada. Por fim, enquanto as versões congeladas representam uma exclusão lógica (podendo ser apenas consultadas), as versões bloqueadas não podem ser populadas, mas podem ser consultadas e utilizadas como base para novas derivações.

5.1.3.1 Operações sobre as versões de esquema

Conforme ilustrado sucintamente no diagrama de estados da Figura 5.6, são definidas operações que podem ser aplicadas sobre as versões de esquema, dependendo do estágio de desenvolvimento em que elas se encontram.

A operação `criar` é utilizada para dar origem a um esquema ainda sem versões, ou seja, à primeira versão de um esquema, que será a raiz do processo de derivação. Neste caso, é um nodo isolado no grafo de derivação de versões de esquemas. Todas as versões são criadas no estado em trabalho.

A operação `excluir` permite remover fisicamente uma versão de esquema, podendo, entretanto, ser aplicada somente às versões no estado em trabalho. A exclusão física está

[1]Neste ponto, é importante esclarecer a diferença entre versão corrente e versão ativa. A versão corrente é definida para permitir a manipulação das versões de esquema quando o esquema versionado é solicitado sem a especificação de nenhuma versão em particular. Uma única versão do esquema pode ser a versão corrente, a cada momento. Em contrapartida, um esquema versionado pode possuir mais de uma versão ativa. Versões ativas representam versões de esquema em operação, sendo manipuladas independentemente pelas aplicações.

restrita ao estado em trabalho porque as versões estão em fase de criação e o histórico da evolução ainda não é mantido.

A operação `promover` permite que versões em trabalho sejam explicitamente promovidas para o estado estável. Isto significa que a versão do esquema está em um estado consistente cujo histórico de evolução deve ser mantido.

A operação `ativar` libera a versão de esquema para que possa ser instanciada, ou seja, permite a manipulação pelas aplicações. A operação `ativar` pode ser aplicada às versões no estado estável ou bloqueada. A ativação de uma versão que se encontra no estado estável acarreta a associação de uma extensão de dados ao esquema, liberando em seguida para instanciações. A ativação de uma versão bloqueada libera o acesso aos dados associados a ela.

A operação `bloquear` torna as versões indisponíveis para os usuários, impedindo a atualização da extensão de dados. Esta operação deve ser aplicada exclusivamente nas versões ativas. Cabe enfatizar que as demais operações, como por exemplo, `derivar`, `consultar`, permanecem disponíveis para as versões bloqueadas.

A derivação explícita de versões de esquema deve ser realizada somente pela operação `derivar`. A operação `derivar` gera uma nova versão como sucessora de uma ou mais versões existentes. A nova versão criada é uma cópia da versão utilizada como base para derivação. Considerando o grafo de derivação, a versão base para derivação torna-se a antecessora da nova versão criada.

Com respeito à derivação a partir de várias versões de esquema, estabelecida pelo TVM, continua a mesma, ou seja: é feita uma cópia da primeira versão especificada para derivação, sendo que o usuário deve fazer o agrupamento das demais versões. A versão derivada torna-se automaticamente sucessora das versões fornecidas. A hipótese de permitir o processo de integração entre duas ou mais versões de esquema foi descartada por envolver diversos aspectos longínquos ao tema central desta tese [2].

Um aspecto importante a ser considerado ocorre quando a versão-base para derivação encontra-se no estado em trabalho. A versão selecionada como base para derivação deve ser promovida para o estado estável antes de dar origem à nova versão. Esse procedimento garante a consistência do grafo de derivação (uma vez que versões em trabalho podem ser fisicamente removidas), além de manter o histórico completo da evolução das versões de esquema. Semanticamente, efetuar uma derivação, tendo como base uma versão em trabalho, produz o mesmo efeito da seqüência de operações `promover` e `derivar`.

Cabe salientar ainda que, para a nova versão criada, não precisam ser especificados os intervalos temporais de transação e de validade, toda versão é criada no estado em trabalho, não possuindo tempos associados. Com isso, o novo esquema pode ser livremente modificado sem que novas versões sejam derivadas.

A operação `congelar` permite excluir logicamente as versões de esquema, promovendo-as para o estado congelado. A operação `congelar` encerra o tempo de validade das versões de esquema. Entretanto, as versões permanecem no grafo de derivação, podendo ser livremente consultadas. Como o histórico da evolução das versões é mantido a partir do estado estável, a operação `congelar` não pode ser aplicada às versões que se encontram no estado em trabalho. Para todos os demais estados esta operação é aplicável.

Além dessas operações, para que a realidade seja completamente modelada, o TVSE permite a restauração de versões de esquemas que foram logicamente excluídas, através

[2] Cabe ressaltar que este tema está sendo tratado como um trabalho futuro, sendo discutido em maiores detalhes nas conclusões (seção 9.2)

Tabela 5.1: Operações aplicáveis sobre as versões de esquema

Operações	Estados	Descrição
`Derivar`	T, E, A, B	Cria uma nova versão, a partir de outra existente. A nova versão representa uma cópia da versão utilizada como base para derivação.
`Promover`	T	Promove a versão em trabalho para estável.
`Ativar`	E, B	Define a versão como (ativa), associando a ela uma extensão (para versões no estado estável) e permitindo novas instanciações (para versões no estado bloqueada)
`Congelar`	E, A, B	Exclui logicamente a versão, através do encerramento do seu tempo de validade
`Restaurar`	C	Restaura a versão para o estado estável (quando não possui extensão) ou para bloqueada (quando possui extensão)
`Excluir`	T	Exclui fisicamente a versão
`Modificar`	T	Modifica fisicamente a versão
	E, A, B	Qualquer modificação provoca a derivação de uma nova versão
`Consultar`	T, E, A, B, C	Consulta a versão e seus dados associados

T - Em trabalho, **E** - Estável, , **A** - Ativa, **B** - Bloqueada, **C** - Congelada

da operação `restaurar`. O intuito dessa operação é reutilizar um esquema que foi erroneamente excluído ou para reavaliar uma alternativa descartada. Essa restauração se dará para o estado bloqueada (caso exista uma extensão associada a esta versão de esquema) ou estável (quando ainda não existe extensão).

Sucintamente, a Tabela 5.1 apresenta a definição das operações que podem ser aplicadas sobre as versões de esquema em cada um dos estados em que se encontram.

5.1.4 Especificação dos Metadados

Uma estrutura de metadados é definida para o armazenamento das informações específicas sobre a evolução temporal das versões de esquemas e de suas respectivas classes, operações, atributos e relacionamentos. A Figura 5.7 ilustra a especificação da estrutura de metadados[3] através de um diagrama de classes. A especificação das operações de cada classe dos metadados é deixada para trabalhos futuros por não ser necessária nesta tese.

A estrutura de metadados é composta pelas seguintes classes e atributos:

- *VersionedSchema* - possui atributos para o tempo de vida, o contador para as configurações, a versão corrente do esquema, a primeira versão, a última versão, o número para próxima versão, o controle de geração de versões do esquema versionado, controle da versão corrente pelo usuário, e os rótulos temporais;
- *SchemaVersion* - possui atributos para o tempo de vida, o nome, o estado (em trabalho, estável, ativa, consolidada e congelada), versões sucessoras e predecessoras, e os rótulos temporais;
- *Entity* - possui os atributos para o identificador e o nome da entidade;

[3]Cabe justificar que a estrutura de metadados foi mantida em inglês para manter a coerência com os artigos publicados e a ferramenta implementada.

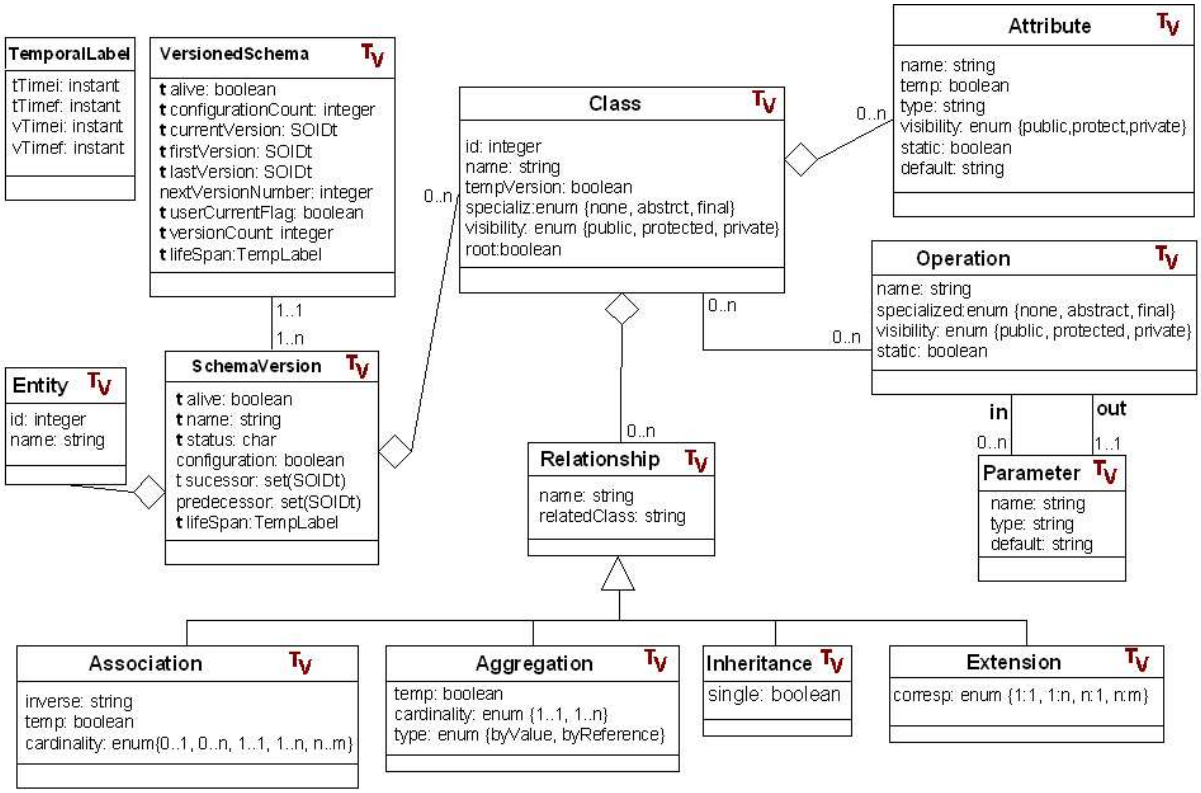


Figura 5.7: Especificação da estrutura de metadados

- *Class* - possui atributos para o identificador e o nome da classe, se é temporal versionada, o tipo de especialização que pode ser especificada a partir da classe (normal - *nome*, classe abstrata - *abstract*, classe final sem subclasses - *final*), visibilidade (pública - *public*, protegida - *protected*, e privada - *private*), e se é raiz de uma hierarquia de herança (refinamento ou extensão);
- *Attribute* - possui os atributos para o nome, se é atributo temporal, o tipo, a visibilidade (público - *public*, protegido - *protected*, e privado - *private*), se é estático (escopo de classe - *static*), e o valor por omissão;
- *Operation* - possui atributos para o nome da operação, o tipo de especialização que pode ser especificada a partir da operação (normal - *none*, classe abstrata - *abstract*, classe final sem subclasses - *final*), a visibilidade (pública - *public*, protegida - *protected*, e privada - *private*), e se é estática (escopo de classe - *static*), podendo se relacionar com parâmetros de entrada ou saída:
- *Parameter* - possui atributos para o nome, o tipo e o valor por omissão do parâmetro da operação;
- *Relationship* - possui atributos para o nome do relacionamento e da classe relacionada, sendo especializado em quatro subtipos:

  – *Association* - possui atributos para o nome da referência inversa, se é temporal, e a cardinalidade (*0..1*, *0..n*, *1..1*, *1..n*, *n..m*);
  – *Aggregation* - possui atributos para armazenar se é um relacionamento temporal, a cardinalidade (*1..1*, *1..n*) e o tipo (por valor - *byValue*, ou por referência - *byReference*);
  – *Inheritance* - possui atributo para indicar se a herança é simples ou múltipla;

  - *Extension* - possui atributo para cardinalidade do relacionamento (*1..1*, *1..n*, *n..1*, *n..m*);

- *TemporalLabels* - possui atributos para os rótulos temporais pré-definidos para o tempo de transação inicial, tempo de transação final, tempo de validade inicial, tempo de validade final.

5.2 Gerenciamento da Modificação de Esquemas

O gerenciamento da modificação de esquemas envolve três etapas distintas, conforme ilustrado na Figura 5.8:

- operações de modificação de esquema - especificação de um conjunto de operações permitido para as versões de esquemas e de seus elementos. O TVSE separa as operações de modificação de esquemas em primitivas e compostas, conforme especificado na seção 5.2.1;

- restrições de integridade - especificação de critérios que visam a garantir que cada operação de modificação de esquema preserve a integridade das versões de esquemas e de seus elementos (por elementos de esquemas entende-se: classes, atributos, métodos e relacionamento);

- derivação de versões de esquema - especificação de um mecanismo para repercutir as mudanças nas versões de esquema (o versionamento temporal de esquema está completamente detalhado na seção 5.1).

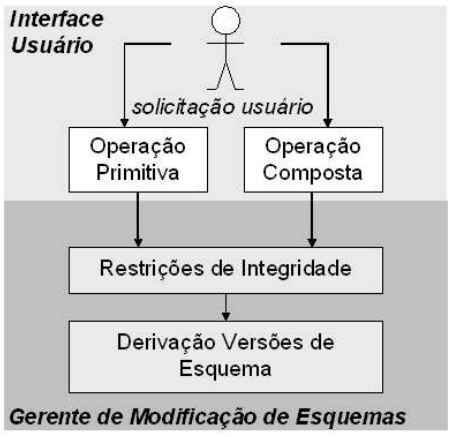


Figura 5.8: Gerente de Modificação de Esquemas

5.2.1 Operações de Modificação de Esquemas

O versionamento de um esquema permite suporte às sucessivas alterações do esquema conceitual, retendo as definições passadas e armazenando assim o histórico da sua evolução. A derivação de uma nova versão de esquema é desencadeada pela aplicação de uma operação de modificação de esquema. Em linhas gerais, as operações de modificação de esquemas estão separadas em duas hierarquias: primitiva e composta. A Figura 5.9 ilustra a hierarquia definida para as operações de modificação de esquemas.

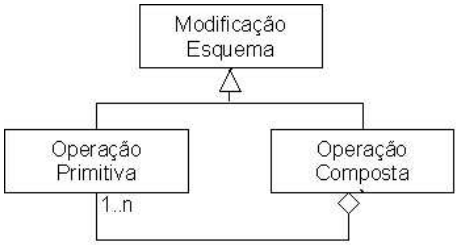


Figura 5.9: Hierarquia das Operações de Modificação de Esquemas

Cabe salientar que, na maioria dos casos, tais modificações devem ser propagadas às instância da base de dados a fim de assegurar a consistência com a nova versão de esquema. Este tópico é explorado na seção 5.4.

5.2.1.1 Operações Primitivas

O TVSE provê uma coleção de operações de modificação de esquema aplicáveis a um esquema. Banerjee (BANERJEE et al., 1987) define uma taxonomia de atualização de esquema para modelos orientados a objetos. As operações primitivas estão definidas em (GALANTE, 1998; GALANTE; SANTOS; RUIZ, 1998), sendo que foram estendidas a fim de contemplar os requisitos temporais do modelo bem como alterações em relacionamentos (temporais e não temporais). Cabe salientar que essas características adicionais não estão definidas em trabalhos prévios, destacando o TVSE dos demais modelos e sistemas presentes na literatura. As primitivas de atualização realizáveis pelo modelo estão separadas em três categorias distintas, representando o tipo de modificação que acarretam no esquema: alterações na estrutura do esquema, alteração na estrutura das classes e alteração da estrutura dos relacionamentos. A Tabela 5.2 lista a taxonomia de mudança especificada para o TVSE[4].

5.2.1.2 Operações Compostas

Uma operação composta, conforme definido em (ROMA; GALANTE; SANTOS, 2000, 2001), é formada por um grupo de operações primitivas executadas seqüencialmente. De forma análoga às primitivas de atualização, a execução de uma operação composta leva à derivação de uma versão de esquema, cujo resultado é uma nova versão de esquema acrescida das modificações realizadas. Essas operações evitam a derivação de versões a cada vez que o esquema é modificado, inibindo a proliferação excessiva de versões. Isto significa que o usuário pode definir um conjunto de operações de modificação sem que o sistema, automaticamente, gere novas versões para cada operação individualmente.

5.2.2 Restrições de Integridade

A amplitude de um modelo de evolução de esquemas pode ser medida pelas possibilidades de transformações que oferece ao esquema do banco de dados. Entretanto, o mecanismo de suporte deve assegurar que essas modificações preservem a integridade do esquema. Para tanto, o modelo provê um conjunto de requisitos, denominados invariantes, para garantir a validade do esquema, assim como das instâncias armazenadas na base de dados. Regras são estabelecidas para reger essa evolução, permitindo a

[4]As primitivas de atualização são tratadas extensivamente na seção 5.3, que apresenta a Linguagem Temporal Versionada para Evolução de Esquemas.

Tabela 5.2: Taxonomia para Evolução de Esquemas

Categoria	Operações
Estrutura Esquema	incluir uma classe
	excluir uma classe
	alterar o nome de uma classe
	alterar o tipo de uma classe (temporal versionada ou não)
Estrutura Classes	incluir um atributo (temporal ou não temporal)
	incluir um método
	excluir um atributo
	excluir um método
	alterar o nome de um atributo
	alterar o domínio de um atributo
	alterar o tipo de um atributo (temporal ou não temporal)
	alterar o nome de um método
Estrutura Relacionamentos	incluir um relacionamento (temporal ou não temporal)
	excluir um relacionamento
	alterar o nome de um relacionamento
	alterar o tipo de um relacionamento (temporal ou não)
	alterar a cardinalidade de um relacionamento

realização de atualizações somente quando não conduzem o esquema a um estado inválido.

O conjunto de restrições é definido a fim de assegurar a integridade do esquema frente às atualizações, assegurando que o novo estado é sempre consistente. Diversas propostas presentes na literatura, como, por exemplo (SKARRA; ZDONIK, 1986; BANERJEE et al., 1987; LAUTEMANN, 1997b; AL-JADIR et al., 1995), adotam invariantes como forma de garantir a integridade perante as modificações. A principal abordagem para o gerenciamento das restrições de integridade, utilizada, por exemplo, no ORION (BANERJEE et al., 1987) e no O_2 (FERRANDINA et al., 1995), consiste em definir e fazer cumprir um conjunto de *invariantes*, separados em três níveis:

- consistência estrutural - asseguram a integridade do conjunto de classes, suas descrições internas e os relacionamentos de generalização e agregação. Alguns requisitos importantes são, por exemplo, nomes únicos, existência de domínios, e herança dos atributos e métodos;

- consistência comportamental - asseguram a integridade do conjunto de operações declarados no esquema. O requisito principal, por exemplo, é que todas as operações chamadas estejam definidas;

- consistência de instanciação - asseguram a integridade das instância com relação ao esquema definido. Como o esquema serve de interface para a interação com o banco de dados, deve sempre manter-se compatível tanto com as instâncias vigentes na base de dados quanto com as aplicações definidas.

O TVSE adota as restrições propostas por Banerjee (BANERJEE et al., 1987), uma vez que o problema com relação à correção dos esquemas frente às modificações está bem investigado no contexto de evolução de esquemas. Não consiste em um padrão, embora tenha sido adotado pela maioria das propostas de gerenciamento de evolução

de esquemas presentes na literatura (conforme tratado na seção 2.3). Esses critérios foram introduzidos no contexto do TVSE para garantir que cada operação de modificação preserve a integridade do esquema e a validade do banco de dados. As alterações são aplicadas diretamente nos esquemas. Entretanto, as mudanças são realizadas somente quando o sistema garante que o esquema permanece em um estado que satisfaça todas as condições definidas pelos invariantes. Além disso, a derivação de versões deve resultar sempre em esquemas estruturalmente válidos e corretos. Assim, as modificações são efetuadas somente quando garantem essa consistência; caso contrário, devem ser recusadas e o usuário, notificado. O processo de derivação de versões de esquema foi amplamente discutido e especificado na seção 5.1.

5.3 Linguagem Temporal Versionada para Evolução de Esquemas

Após apresentar os requisitos do gerente de modificação de esquemas, esta seção descreve a linguagem de especificação, versionamento e modificação de esquemas. A linguagem é denominada **TVL/SE**, uma sigla para ***T**emporal **V**ersioning **L**anguage for **S**chema **E**volution*.

Essa linguagem deve permitir, além das operações básicas de especificação de esquemas, novas operações que consintam a modificar os esquemas, manipulando as versões e os rótulos temporais. Contemplando esses requisitos, a seguir são apresentadas as características gerais da linguagem, as características específicas de criação de esquemas, de versionamento temporal de esquemas e as operações de modificação de esquemas.

Um artigo descrevendo a linguagem de especificação, versionamento e modificação de esquemas e seu mapeamento para ODMG foi publicado nos anais da *14th International Conference on Database and Expert Systems Applications - DEXA 2003* - (GALANTE; EDELWEISS; SANTOS, 2003).

5.3.1 Especificação da Linguagem TVL/SE

A linguagem de especificação, versionamento e modificação de esquemas do TVSE baseia-se nas linguagens ODL e OQL propostas pelo padrão ODMG (CATTELL et al., 2000), apresentando também alguma influência da linguagem TSQL2 (SNODGRASS, 1995) no que diz respeito ao tratamento dos aspectos temporais. Cabe salientar que o objetivo principal não é propor uma nova linguagem para versionamento e modificação de esquemas, uma vez que diversas propostas, como, por exemplo, (GRANDI; MANDREOLI, 2003; CLAYPOOL; JIM; RUNDENSTEINER, 1998; RIEDEL, 1999; DAVIDSON; KOSKY, 1999), apresentam tais funcionalidades. Ao invés disso, o objetivo consiste em incorporar uniformemente as características de tempo e de versão na linguagem ODL, a fim de permitir o gerenciamento apropriado de evolução de esquemas.

A TVL/SE está separada em três partes distintas:

- especificação e versionamento de esquemas, que controla o processo de criação, derivação e manipulação das versões de esquemas. Esse mecanismo está conceitualmente detalhado na seção 5.1.3;
- modificação de esquemas, que controla o processo de modificação das versões de esquemas, especificando cada operação primitiva de modificação de esquema proposta pelo modelo. Esse mecanismo está conceitualmente detalhado na seção 5.2;

- atualização de objetos, controla o processo de atualização de objetos (`insert`, `update` e `delete`) durante o versionamento temporal de esquemas.

As definições da linguagem TVL/SE são parcialmente baseadas no modelo de objetos proposto pelo ODMG, sendo que novos conceitos são introduzidos para gerenciar apropriadamente os conceitos de tempo e de versão. O mapeamento completo da linguagem TVL/SE para o padrão ODMG está publicado em (GALANTE; EDELWEISS; SANTOS, 2003)[5]. Esse mapeamento permite gerenciar apropriadamente o mecanismo de modificação de esquemas colocando a TVL/SE disponível para uso em qualquer SGBDOO.

As subseções a seguir apresentam a sintaxe geral e simplificada da linguagem para especificação, versionamento e modificação de esquemas[6].

5.3.1.1 Especificação e Versionamento de Esquemas

A maioria das linguagens de especificação de esquemas, como, por exemplo a linguagem ODL definida pelo grupo ODMG (CATTELL et al., 2000) para BDOO, permite a criação de um único esquema por banco de dados. Um dos principais objetivos da linguagem TVL/SE é introduzir o conceito de versão de esquema no nível da linguagem de especificação, permitindo o pleno desenvolvimento e manutenção de versões de esquemas. A Figura 5.10 apresenta a sintaxe geral e simplificada das operações que atuam no grafo de derivação, ou seja, as operações de criação e manipulação de versões de esquemas. A BNF completa da linguagem pode ser conferida no Anexo A.

```
<schemaVersionDagOperation> ::= ( <create>
                                | <delete>
                                | <derive>
                                | <promote>
                                | <activate>
                                | <block>
                                | <freeze>
                                | <restore> )
<create> ::=  CREATE SCHEMA <schema version name>
              <schema specification>
<derive> ::= DERIVE SCHEMA <schema version name>
             FROM <schema version selection>
<delete> ::= DELETE SCHEMA <schema version selection>
<promote> ::= PROMOTE SCHEMA <schema version selection> [<initial valid time>]
<activate> ::= ACTIVATE SCHEMA <schema version selection> [<initial valid time>]
<block> ::= BLOCK SCHEMA <schema version selection> [<initial valid time>]
<freeze> ::= FREEZE SCHEMA [ [cascade] <schema version selection>] [<final valid time>]
<restore> ::= RESTORE SCHEMA [ [top] <schema version selection>] [<initial valid time>]
```

Figura 5.10: Sintaxe simplificada para versionamento de esquemas

A granularidade do versionamento está associada aos esquemas. A operação **`CREATE SCHEMA`** é utilizada para criar um esquema ainda sem versões, ou seja, a primeira versão de um esquema, que será a raiz do processo de derivação (ou seja, é um nodo isolado no grafo de derivação). Todos os esquemas criados a partir da linguagem TVL/SE

[5]O mapeamento da linguagem TVL/SE não é apresentado nesta tese para não tornar repetitiva a apresentação da linguagem.

[6]Cabe justificar que a gramática que define a linguagem TVL/SE está especificada em inglês para manter a coerência com os artigos publicados, evitando, assim, a necessidade de releitura da definição dos comandos e da formalização realizada. A fim de facilitar a leitura do texto, na primeira ocorrência de cada comando da linguagem, o nome corresponde em português é apresentado entre parênteses.

são considerados temporais e versionados, mesmo que posteriormente não possuam nenhuma outra versão sucessora. Um esquema criado com a operação **CREATE SCHEMA** representa uma versão no estado em trabalho, não possuindo tempos associados.

O nome do esquema deve ser informado através da cláusula <schema version name>, facilitando a identificação das versões de esquemas pelas operações de manipulação durante a evolução.

A cláusula <schema specification> permite a definição completa de um esquema, incluindo a especificação das classes, dos atributos, das operações e dos relacionamentos. Cabe salientar que a especificação de um esquema deve seguir a notação definida pela Linguagem de Definição de Classes proposta para o TVM (MORO, 2001), previamente ilustrada na seção 3.2.7. Uma das grandes vantagens é que essa linguagem permite a definição de classes sem tempo e sem versões entre as classes temporais versionadas, mantendo a compatibilidade com as aplicações convencionais existentes.

O Exemplo 5.1 ilustra a especificação da primeira versão de um esquema utilizando a linguagem TVL/SE. Por motivos de simplificação, o exemplo apresenta apenas parte da modelagem do Sistema Discente apresentada no capítulo 3 (Figura 3.10).

> **Exemplo 5.1** Suponha que está sendo criado o esquema denominado "Alunos01". São especificadas duas classes: "Aluno" e "Curso", bem como o relacionamento entre elas ("cursa"). A definição do esquema utilizando a linguagem TVL/SE seria:
>
> ```
> CREATE SCHEMA Alunos01
>    Class Aluno hasVersions
>         ( Properties:
>                matricula : integer ;
>                temporal endereco : string ;
>                temporal email : string ;
>           Relationships:
>              cursa 1:n Curso );
>     Class Curso hasVersions
>         ( Properties:
>                curso : string ;
>                temporal descricao : string ;
>                temporal areaAtuacao : string );
> ```
>
> Note que a especificação das classes mantém o padrão com a Linguagem de Definição de Classes do TVM. ■

5.3.1.1.1 Seleção das Versões de Esquema para Aplicação das Operações

Antes de especificar as operações que atuam sobre o grafo de derivação de versões é importante ressaltar a forma como tais versões são recuperadas. Isto é essencial, uma vez que a execução de tais operações requer a seleção prévia de uma determinada versão de esquema. A especificação da versão do esquema pode ser feita de maneira implícita (sempre o esquema corrente) ou de maneira explícita através da cláusula SET SCHEMA. O comando SET SCHEMA é basicamente o mesmo introduzido pela linguagem TSQL2 (SNODGRASS, 1995) e estendido em (CASTRO; GRANDI; SCALAS, 1997) para o suporte a intervalos temporais. A Figura 5.11 apresenta a gramática simplificada para a cláusula <schema version selection>.

A cláusula <schema version name> permite a identificação de uma versão específica do esquema, através do seu identificador (nome da versão de esquema). Alternativamente, é possível informar um intervalo de tempo de transação e/ou de tempo de validade. Em geral, a utilização do comando SET SCHEMA pode resultar na seleção

```
<schema version selection> ::= SET SCHEMA <selection condition>
<selection condition> ::= <schema version name>
   | [AND VALID <defined interval>]
     [AND TRANSACTION <defined interval>]
```

Figura 5.11: Sintaxe simplificada para seleção de versões de esquema

de mais de uma versão de esquema, por exemplo, quando um intervalo de transação e/ou validade é especificado. Entretanto, as operações que atuam sobre as versões de esquema devem manipular exclusivamente uma única versão e nunca um conjunto delas. Para solucionar este problema, quando o comando `SET SCHEMA` recupera mais de uma versão de esquema, a transação deve ser rejeitada[7]. Por outro lado, a execução do comando `SET SCHEMA` pode resultar na recuperação de nenhuma versão de esquema, neste caso a operação deve ser igualmente rejeitada.

É importante ressaltar que os valores selecionados com o comando `SET SCHEMA` dizem respeito somente aos dados da intenção, não tendo nenhuma influência sobre a recuperação dos dados da extensão. Em se tratando da extensão, as condições de seleção temporal são normalmente utilizados no comando `SELECT`[8].

5.3.1.1.2 Promoção de uma Versão de Esquema

A promoção de uma versão de esquema (operação `promover`, seção 5.1.3.1) é efetuada através do comando `PROMOTE SCHEMA`. A cláusula opcional `<initial valid time>` é introduzida para especificar o instante em que a versão do esquema assume o novo estágio de desenvolvimento. O valor final do tempo de validade é definido como `UC` (*until change*), uma vez que será finalizado somente quando a versão do esquema passar para o próximo estágio, ou seja, somente quando uma nova operação de transição de estado for aplicada. Quando o valor do tempo de validade inicial não é especificado, assume-se o valor atual do sistema (*now*). O tempo de transação deve ser gerenciado automaticamente pelo sistema. Nesse caso, a pertinência temporal da versão de esquema promovida para o estado estável é `[now,∞]`, ou seja, o valor inicial definido para o tempo de transação é o atual (*now*) e o valor final é infinito (∞)[9].

Cabe salientar que, para manter a integridade na definição dos estágios de desenvolvimento, o valor final do intervalo de validade não pode ser fornecido na execução das operações que manipulam as versões de esquema. Por exemplo, se o valor final do intervalo de validade pudesse ser informado, seria possível haver um intervalo no tempo em que a versão não estaria associada a nenhum estágio de desenvolvimento.

O Exemplo 5.2 ilustra a promoção da primeira versão de esquema criada no exemplo 5.1, para o estágio de desenvolvimento estável.

---

[7] No contexto de atualização de objetos, a recuperação de diversas versões de esquemas, denominada atualizações multi-esquemas, é essencial a fim de permitir a manipulação de esquemas e dados frente ao processo de evolução de esquemas. Esta questão é abordada em detalhes no capítulo 6.

[8] A consulta aos dados temporais considerando apenas uma única versão de esquema está especificada na linguagem TVQL (MORO et al., 2002), apresentada na seção 3.3. A resolução de consultas aos dados temporais considerando diversas versões de esquema é tratada detalhadamente no capítulo 6.

[9] A especificação para os intervalos de transação e de validade segue os mesmos critérios para as demais operações.

Exemplo 5.2 Suponha que a primeira versão do esquema, "Alunos01", é promovida para o estado estável, a partir da data 01/01/2004. A promoção da versão do esquema utilizando a linguagem TVL/SE seria:

```
PROMOTE SCHEMA Alunos01 "01/01/2004"
```

Note que nenhuma nova versão é gerada. A mesma versão é promovida para estável, passando a ser registrados os tempos de transação [now,∞] e de validade [01/01/2004,UC]. ■

5.3.1.1.3 Derivação de uma Versão de Esquema

O comando DERIVE SCHEMA (operação derivar, seção 5.1.3.1) gera uma nova versão como sucessora de uma ou mais versões existentes. A nova versão criada é uma cópia da versão utilizada como base para derivação. É importante ressaltar que, quando a versão utilizada como base para derivação estiver no estágio em trabalho, deve ser promovida para estável antes da criação da nova versão. De forma similar, à operação de especificação de esquemas, através da cláusula <schema version name> o usuário deve informar o nome da nova versão de esquema criada. A cláusula **FROM** permite a especificação da versão utilizada como base para derivação, caso não seja informado, é tomada a versão corrente. A cláusula <schema version selection> recupera a(s) versão(ões) de esquema utilizada(s) como base para derivação.

O Exemplo 5.3 ilustra a geração de uma nova versão como sucessora do esquema promovido no Exemplo 5.2.

Exemplo 5.3 Suponha que uma nova versão "Alunos02" é gerada tendo como base a primeira versão do esquema "Alunos01". A derivação de uma nova versão do esquema utilizando a linguagem TVL/SE seria:

```
DERIVE SCHEMA Alunos02 FROM Alunos01
```

Note que o nome da versão do esquema foi utilizado para seleção do esquema-base de derivação. Neste caso, uma cláusula de consulta (SET SCHEMA) poderia ter sido utilizada também. ■

5.3.1.1.4 Ativação de uma Versão de Esquema

A ativação de uma versão de esquema é realizada através do comando **ACTIVATE SCHEMA** (operação ativar, seção 5.1.3.1), habilitando o esquema para ser populado, ou seja, tornando-o disponível para manipulação dos dados da extensão. A cláusula <initial valid time> pode ser especificado o instante em que a versão assume o estágio ativa. Cabe salientar que um novo intervalo de validade deve ser criado para a versão, sendo encerrado o intervalo de validade em que ela permaneceu estável. O Exemplo 5.4 ilustra a ativação da segunda versão do esquema, promovido para estável no exemplo 5.2.

Exemplo 5.4 Suponha que a segunda versão do esquema, "Alunos02", agora é ativada, a partir da data 01/02/2004. A ativação da versão do esquema utilizando a linguagem TVL/SE seria:

```
ACTIVATE SCHEMA Alunos02 "01/02/2004"
```

Note que um novo intervalo de validade é criado tanto para o tempo de transação ([now,∞]) quanto para o tempo de validade ([01/02/2004,UC]).

■

5.3.1.1.5 Bloqueio de uma Versão de Esquema

O comando **`BLOCK SCHEMA`** é utilizado para desabilitar a atualização da extensão associada à versão de esquema. Cabe salientar que um novo intervalo de validade deve ser criado para à versão de esquema, sendo encerrado o intervalo de validade em que ela permaneceu ativa. O Exemplo 5.5 ilustra o bloqueio da segunda versão do esquema, ativada previamente no exemplo 5.4.

Exemplo 5.5 Suponha que a segunda versão do esquema, "Alunos02", agora é bloqueada, a partir da data 01/06/2004. O bloqueio da versão do esquema utilizando a linguagem TVL/SE seria:

```
BLOCK SCHEMA Alunos02 "01/06/2004"
```

Note que um novo intervalo de validade é criado tanto para o tempo de transação (`[now,∞]`) quanto para o tempo de validade (`[01/06/2004,UC]`). ■

5.3.1.1.6 Exclusão Física uma Versão de Esquema

Uma versão de esquema no estágio em trabalho pode ser excluída fisicamente do banco de dados através do comando `DELETE SCHEMA` (operação `excluir`, seção 5.1.3.1). Cabe salientar que esta operação não possui os parâmetros para os intervalos temporais, uma vez que tais versões não têm tempos associados a elas. Além disso, considerando o grafo de derivação de versões, uma exclusão física não acarreta nenhum efeito colateral, pois são sempre versões sem sucessores na hierarquia de derivação. O Exemplo 5.6 ilustra a derivação de uma nova versão de esquema (`Alunos03`) a partir da segunda versão do esquema (`Alunos02`) e sua posterior exclusão.

Exemplo 5.6 Suponha que uma terceira versão é derivada da segunda versão do esquema, `Alunos02`, e posteriormente excluída. A derivação e a exclusão da versão do esquema utilizando a linguagem TVL/SE seriam:

```
DERIVE SCHEMA Alunos03 FROM Alunos02
DELETE SCHEMA Alunos03
```

Note que, por ser um esquema em trabalho, a versão `Alunos03` é removida fisicamente do banco de dados. ■

5.3.1.1.7 Exclusão Lógica de uma Versão de Esquema

A exclusão lógica de uma versão (operação `congelar`, seção 5.1.3.1) é realizada através do comando **`FREEZE SCHEMA`**, sendo encerrado o tempo de validade desta versão. A cláusula `<schema version selection>` permite recuperar uma versão específica do esquema para ser excluída. A cláusula opcional `cascade` permite, recursivamente, excluir todas as versões sucessoras da versão recuperada, nos casos em que o usuário julgar que todas as versões sucessoras foram geradas erroneamente. A execução da operação **`FREEZE SCHEMA`** sem nenhum parâmetro associado realiza a exclusão lógica do esquema versionado e, conseqüentemente, de todas as suas versões. O Exemplo 5.7 ilustra a exclusão lógica da segunda versão do esquema (`Alunos02`).

Exemplo 5.7 Suponha que a segunda versão de esquema `Alunos02` é logicamente excluída, a partir da data 31/12/2004. A exclusão da versão do esquema utilizando a linguagem TVL/SE seria:

```
FREEZE SCHEMA Alunos02 "31/12/2004"
```

Note que os tempos de transação (`[31/12/2003, 31/12/2003+1]`) e de validade (`[31/12/2003, 31/12/2003+1]`) são encerrados, porém, a versão do esquema continua armazenada no banco de dados, permanecendo disponível para realização de consultas. ■

Cabe salientar ainda que a execução de uma exclusão lógica acarreta o encerramento do tempo de transação e validade de todos os elementos do esquema. Os intervalos temporais de transação e de validade dos atributos e dos relacionamentos temporais, bem como dos objetos e respectivas versões da base de dados são recursivamente encerrados, com o valor temporal fornecido para o encerramento da versão do esquema.

5.3.1.1.8 *Vaccuming*

O conceito de *vacuuming* (JENSEN, 1999) permite remover fisicamente informações temporais, nos casos em que não são mais relevantes. Conforme explicitado na seção 3.2.1.1, O TVM não estabelece regras para este tipo de exclusão, uma vez que o histórico da evolução dos dados não seria mantido na sua plenitude. Considerações similares são necessárias com respeito às versões do esquema conceitual, especialmente nos casos em que existem dados associados às versões de esquema, referidos através do tempo de transação e/ou de validade. Pragmaticamente, o TVSE não provê regras para este tipo de exclusão, assumindo que as versões de esquema só podem ser excluídas logicamente, exceto para as versões no estado em trabalho, cujo histórico temporal não é retido.

5.3.1.1.9 Restauração de uma Versão de Esquema

O comando **`RESTORE SCHEMA`** (operação `restaurar`, seção 5.1.3.1) permite restaurar uma versão de esquema que foi logicamente excluída. A restauração de uma versão de esquema é feita através da sua especificação pela cláusula `<schema version selection>`. Porém, quando se deseja restaurar uma versão de esquema cuja predecessora foi excluída, a versão predecessora pode ser restaurada antes a fim de manter a seqüência de derivação (e assim recursivamente até que não exista nenhuma versão predecessora excluída). A cláusula **`top`** permite restaurar, recursivamente, todas as versões predecessoras daquela que se deseja restaurar. Neste caso, esta operação é interessante para restabelecer o grafo de derivação de versões. Quando a seleção de versões não retorna nenhuma versão, a operação não é executada. Por fim, a execução da operação **`RESTORE SCHEMA`** sem nenhum parâmetro associado permite restaurar o esquema versionado que foi logicamente excluído, juntamente com suas versões de esquema. A restauração reconstrói o esquema versionado exatamente como estava no momento da exclusão. Nos casos em que uma nova seqüência de derivação foi gerada, a restauração não tem como ser executada.

A cláusula `<initial valid time>` permite especificar o instante inicial do novo intervalo de validade da(s) versão(ões) de esquema que está(ão) sendo restaurada(s). Neste caso, o esquema versionado, bem como suas respectivas versões, terão armazenado nos seus históricos dois tempos de vida. O Exemplo 5.8 ilustra a restauração da segunda versão do esquema (`Alunos02`), excluído previamente no Exemplo 5.7.

Exemplo 5.8 Suponha que a segunda versão de esquema `Alunos02` é restaurada, a partir da data 01/01/2005. A restauração da versão do esquema utilizando a linguagem TVL/SE seria:

```
RESTORE SCHEMA Alunos02 "01/01/2005, 00:00"
```

Note que dois novos intervalos para os tempos de transação (`[now,∞]`) e de validade (`[01/01/2004, UC]`) são gerados. ■

5.3.1.2 Modificação de Esquemas

Conforme apresentado na seção 5.2.1, são definidas primitivas de modificação de esquemas, que representam um dos caminhos para se criar novas versões de esquema a partir outras existentes, registrando a história da evolução. A Figura 5.12 apresenta a sintaxe geral e simplificada da linguagem de modificação de esquemas.

```
<primitiveUpdate> ::= (<add> | <delete> | <modify> ) [<schema version name>]
 [newSV <schema version name>]
```

Figura 5.12: Sintaxe simplificada das operações de modificação de esquemas

Uma primitiva de modificação permite alterar uma versão de esquema. Para as versões que encontram-se no estado em trabalho, as primitivas de atualização realizam correções, ou seja, uma seqüência de modificações podem ser realizadas sem que sejam derivadas novas versões do esquema. Para as versões nos estados estável, ativa ou bloqueada, as operações de modificação sempre acarretam a derivação de uma nova versão.

Os comandos de modificação atuam sobre elementos. Um elemento pode ser uma classe, um atributo, um método ou relacionamento. Esses comandos são classificados em três categorias: inclusão de elemento, exclusão de elemento e modificação de elemento. A cláusula `<schema version name>` permite identificar a versão requerida para ser modificada. A utilização da cláusula **`newSV`** implica na criação de uma nova versão para o esquema, que é constituída por uma cópia da versão antecessora, sendo acrescida a modificação realizada. O nome da nova versão gerada deve ser informado na cláusula `<schema version name>`.

5.3.1.2.1 Inclusão de Elementos

A Figura 5.14 ilustra a sintaxe geral e simplificada da operação para incluir um elemento.

O comando **`ADD CLASS`** permite incluir uma classe de uma versão de esquema específica, sendo que o nome da classe deve ser informado na cláusula `<class name>`. O comando **`ADD ATTRIBUTE`** permite incluir um atributo, no qual devem ser especificados o nome do atributo (cláusula `<attribute name>`), o domínio do atributo (`<attribute domain>`) e a classe a que pertence (`<class name>`). Adicionalmente, o atributo pode ser definido como temporal, através da cláusula **`TEMPORAL`**. O comando **`ADD RELATIONSHIP`** permite incluir um relacionamento, no qual devem ser especificados o nome do relacionamento (<relationship name>), a cardinalidade do relacionamento, as classes entre as quais o relacionamento é especificado (`<class name>`) e alternativamente o nome do relacionamento inverso (cláusula **`INVERSE`**). O

```
<add> ::= ADD ( CLASS <class>
 | ATTRIBUTE [TEMPORAL] <attribute name> : <attribute domain>
   [DEFAULT value] IN <class name>
 | RELATIONSHIP [TEMPORAL] <relationship name>
    (0:1|0:n|1:1|1:n|n:m)
    [INVERSE <relationship name> ] <class name> IN <class name>
 | OPERATION <operation name> "(" [<parameter list> ] ")"
    [ : <returned value domain> ] IN <class name> )
```

Figura 5.13: Sintaxe simplificada para a operação de inclusão

comando **ADD OPERATION** permite incluir uma operação em uma versão de esquema específica, sendo que o nome da classe deve ser informado na cláusula <class name>. Alternativamente, a lista de parâmetros de entrada e/ou o parâmetro de saída podem ser especificados.

5.3.1.2.2 Exclusão de Elementos

A Figura 5.14 ilustra a sintaxe geral e simplificada da operação para excluir um elemento.

```
<delete> ::= DELETE (CLASS <class name>
 | ATTRIBUTE <attribute name> FROM <class name>
 | RELATIONSHIP <relationship name> FROM <class name> <class name>
 | OPERATION <operation name> )
```

Figura 5.14: Sintaxe simplificada para a operação de exclusão

O comando **DELETE CLASS** permite excluir uma classe de uma versão de esquema específica, sendo que o nome da classe deve ser informado na cláusula <class name>. O comando **DELETE ATTRIBUTE** permite excluir um atributo, no qual devem ser especificados o nome do atributo (cláusula <attribute name>) e a classe a que pertence (<class name>). O comando **DELETE RELATIONSHIP** permite excluir um relacionamento, no qual devem ser especificados o nome do relacionamento (<relationship name>) e as classes entre as quais o relacionamento é especificado (className). Por fim, o comando **DELETE OPERATION** permite excluir uma operação, no qual o nome deve ser especificado (cláusula <operation name>).

5.3.1.2.3 Modificação de Elementos

A Figura 5.15 ilustra a sintaxe geral e simplificada da operação para modificar um elemento.

Os comandos **MODIFY CLASS NAME**, **MODIFY ATTRIBUTE NAME**, **MODIFY OPERATION NAME** e **MODIFY RELATIONSHIP NAME** permitem, respectivamente, alterar o nome de uma classe, um atributo, um relacionamento ou uma operação. Os comandos **MODIFY CLASS TYPE** permite alterar uma classe entre temporal versionada e não temporal versionada. Os comandos **MODIFY ATTRIBUTE TYPE** e **MODIFY RELATIONSHIP TYPE** permitem, respectivamente, alterar um atributo entre temporal e

```
<modify> ::= MODIFY (CLASS NAME <class name> INTO <class name>
 | CLASS TYPE <class name> INTO (hasVersions|normal)
 | ATTRIBUTE NAME <attribute name> INTO <attribute name> IN <class name>
 | ATTRIBUTE DOMAIN <attribute name> INTO <attribute domain> IN <class name>
 | ATTRIBUTE TYPE <attribute name> INTO (temporal|normal) IN <class name>
 | OPERATION NAME <operation name> INTO <operation name> ) IN <class name>
 | RELATIONSHIP NAME [INVERSE] <relationship name> INTO <relationship name> )
    IN <class name>
 | RELATIONSHIP CARDINALITY [INVERSE] <relationship name> (1:1|1:n|n:1|n:n)
    IN <class name>
 | RELATIONSHIP TYPE <relationship name> INTO (temporal|normal)
    IN <class name> )
```

Figura 5.15: Sintaxe simplificada para a operação de modificação

não temporal e alterar um relacionamento entre temporal e não temporal. O comando **`MODIFY ATTRIBUTE TYPE`** permite alterar o domínio de um atributo. Por fim, o comando **`MODIFY RELATIONSHIP CARDINALITY`** permite alterar a cardinalidade de um relacionamento e alternativamente do seu relacionamento inverso.

5.4 Propagação de Mudanças

Realizadas as modificações no esquema conceitual, as instâncias vigentes na base de dados precisam ser adaptadas às novas especificações para manter a coerência entre a estrutura definida e os dados armazenados. Esse problema deve ser resolvido através dos seguintes pontos:

- identificar as instâncias que precisam ser modificadas;
- transformar as instâncias de forma que representem as modificações de esquema.

Esta seção apresenta o processo de propagação de mudanças proposto para o TVSE.

O artigo publicado nos anais do *International Workshop on Evolution and Change in Data Management (ECDM)*, no escopo do *International Conference on Conceptual Modelling (ER)*, apresenta o TVSE, enfatizando o processo de propagação de mudanças através da especificação de funções de propagação e conversão (GALANTE; EDELWEISS; SANTOS, 2002b).

5.4.1 Funções de Adaptação

Finalizado o processo de modificação de esquemas, é inicializado o processo de adaptação da base de dados. As instâncias armazenadas no banco de dados são modificadas para que representem, exatamente, as definições da nova classe. Essas modificações constituem, por exemplo, inclusão e exclusão de objetos, e mudança de tipos das propriedades, entre outras.

O processo de adaptação das instâncias é praticamente automático, ou seja, após a modificação do esquema, o mecanismo de atualização é invocado através da operação `ativar` (especificada na Linguagem TVL/SE - seção 5.3), sendo realizadas as atualizações na base de dados através das funções associadas aos esquemas ou às classes. Por exemplo, uma propriedade que tem seu tipo trocado para outro mais especializado, durante o processo de adaptação tem seu tipo convertido de acordo com a nova definição.

Entretanto, em alguns casos existe a necessidade de realizar a definição das funções manualmente, de acordo com as necessidades da aplicação, identificadas pelo usuário. Desse modo, as funções devem ser implementadas para as instâncias que devem ser modificadas. Por exemplo, quando há a necessidade dos valores de uma propriedade migrarem para a nova instância da classe modificada.

Para a propagação das instâncias é proposto um mecanismo híbrido que contempla as técnicas de conversão imediata e conversão adiada. Conforme ilustrado na Figura 5.16, cada versão de esquema tem associado um conjunto de objetos, definindo seu escopo de acesso. Uma modificação de esquema implica a atualização imediata dos objetos que estão sendo utilizados pelas aplicações. Por outro lado, instâncias podem ser manipuladas a partir de outras versões de esquema, diferentes do seu escopo de acesso. Nesse caso, o mecanismo de conversão adiada é utilizado para transformar as instâncias de acordo com as novas definições de esquema.

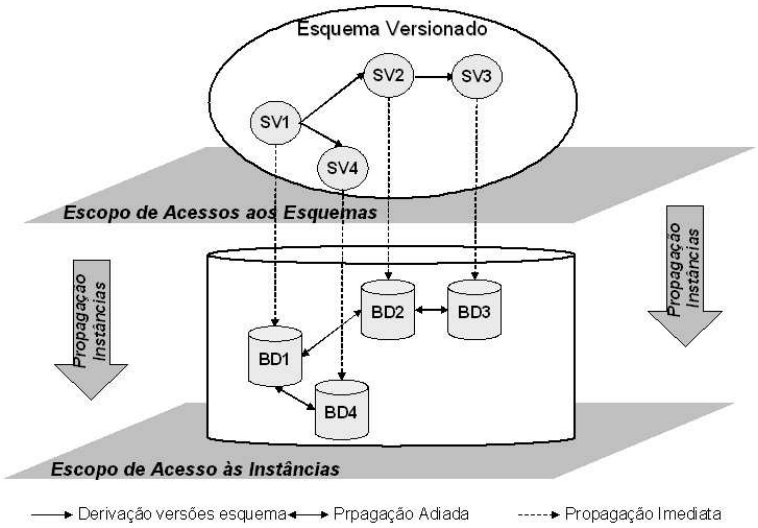


Figura 5.16: Escopo de acesso às instâncias

Para tanto, funções de adaptação são definidas a fim de definir como as instâncias devem ser modificadas para se tornarem consistentes com a nova versão de esquema, representando o efeito das mudanças para as aplicações. Na propagação imediata, essas funções são acionadas no instante em que a operação de ativação é requerida. Na propagação adiada, essas funções são acionadas quando um determinado objeto, inexistente na base de dados do esquema selecionado, é requerido.

Dois tipos de funções são definidos: funções de propagação, definidas no nível do esquema, cujo objetivo é definir relacionamentos de equivalência entre as classes de versões distintas de esquema, e funções de conversão, definidas no nível das classes, cuja finalidade é mapear o conteúdo dos objetos de uma versão de classe para outra. Essas funções garantem que as informações geradas em qualquer versão de esquema permaneçam visíveis e atualizáveis sob qualquer perspectiva de versão através de implementações que realizam uma adaptação, restaurando o conteúdo dos objetos de uma versão para outra.

5.4.1.1 Funções de Propagação

Classes de uma versão de esquema para outra são consideradas idênticas pela comparação de seu nome. Entretanto, devido ao refinamento do esquema, uma classe

pode apresentar nomes diferentes, porém propriedades e comportamentos comuns. Assim, surge a necessidade de estabelecer relacionamentos de equivalência entre as classes de diversas versões de esquema.

Uma função de propagação é definida entre duas classes, em versões distintas de esquema, e é acionada no instante em que a versão do esquema é requisitada. A definição de um relacionamento semântico é composta pela implementação de duas funções de propagação: função *update* e função *backdate*. A função *update* é conectada à versão inicial do esquema e indica a equivalência com a próxima versão do esquema. A função *backdate* é conectada à versão nova do esquema e indica a equivalência com as classes da versão anterior do esquema, conforme ilustrado na Figura 5.17. As funções *update* apresentam como parâmetro o número da versão sucessora e as funções *backdate*, o número da versão antecessora. Isso se faz necessário, pois, devido à derivação de versões gerar um grafo acíclico dirigido, uma versão de esquema pode apresentar uma ou mais funções de propagação.

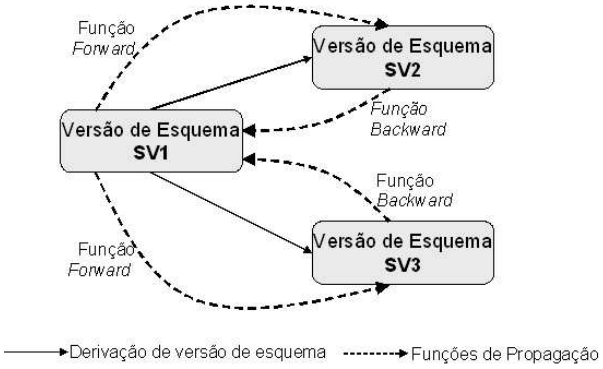


Figura 5.17: Exemplo da especificação das funções de propagação

Para exemplificar, a Figura 5.18 mostra os algoritmos que especificam as funções de propagação *update* e *backdate* entre duas versões de esquemas, no qual uma classe é inserida a partir da primeira versão do esquema (SV1), gerando a segunda versão do esquema (SV3). A primeira função, por exemplo, envolve operações com respeito à inclusão de um objeto e o seu relacionamento com a nova classe criada. A segunda função, por sua vez, envolve operações para excluir um objeto, bem como o relacionamento existente com a classe relatada.

5.4.1.2 Funções de Conversão

Como a estrutura de uma classe pode ser modificada derivando novas versões de esquema para repercutir tal mudança, as instâncias presentes no banco de dados precisam acompanhar estas alterações a fim de manter sempre um estado consistente para qualquer versão sob a qual são requeridas. Funções de conversão devem ser especificadas a fim de mapear o conteúdo dos objetos de uma versão de esquema para outra, considerando as classes em particular.

De forma análoga às funções de propagação, uma função de conversão implica a implementação de duas funções: função *update* e função *backdate*. As funções *update* são associadas à classe em que a atualização foi realizada e descrevem as características dos objetos quando solicitados na próxima versão do esquema, isto é, implementam operações que adaptam os objetos segundo a evolução de esquema. As funções *backdate* são associadas à nova versão do esquema, descrevendo a estrutura dos objetos na versão

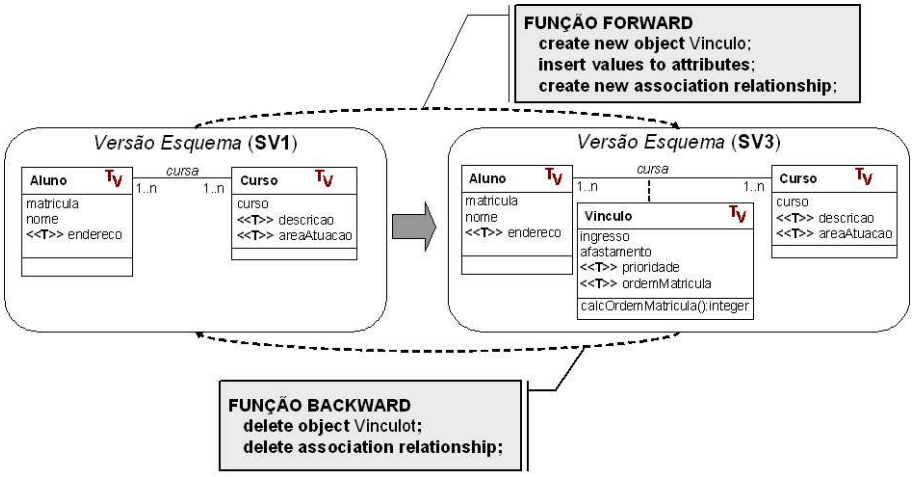


Figura 5.18: Exemplo de uma função de propagação

anterior, ou seja, antes de realizadas as modificações. A Figura 5.19 ilustra as funções de conversão aplicadas quando o atributo `email` é inserido na classe `Aluno`. A função *update* atribui um valor para o email e a função *backdate* inibe a exibição do atributo `email`.

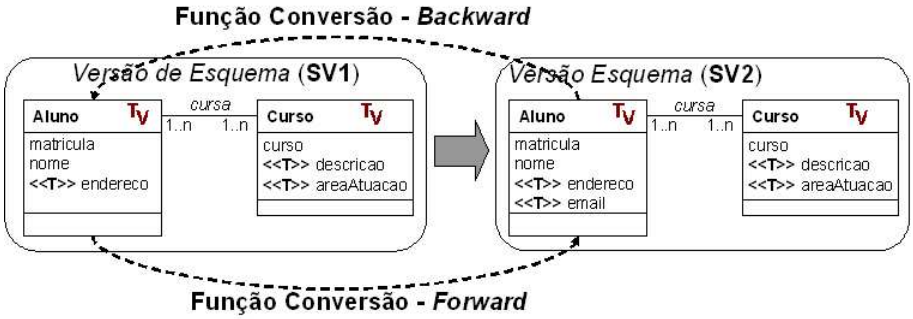


Figura 5.19: Exemplo de uma função de conversão

As funções de conversão, são geradas automaticamente pelo sistema em decorrência da execução de operações de atualização na estrutura das classes. Porém, podem ser construídas e/ou modificadas pelo usuário quando este julgar necessário. Quando uma nova versão de esquema é gerada, caso o usuário não implemente a função de conversão, esta é definida pelo sistema, adotando o procedimento default estabelecido pelo modelo. Cabe salientar que as funções de conversão definidas pelo usuário devem garantir as restrições de integridade definidas na seção 5.2.2 (consistência de instanciação). Caso contrário, a propagação de mudanças deve ser recusada e o usuário notificado.

5.4.2 Estratégia de Propagação de Mudanças nas Instâncias

As mudanças realizadas na estrutura do banco de dados precisam manter a consistência e a compatibilidade dos objetos com todas as versões de esquemas e classes definidas no esquema. A técnica de versionamento de objetos é utilizada na adaptação das instâncias, pois permite o armazenamento do histórico das modificações realizadas no esquema conceitual do banco de dados, como também garante a adaptação das informações com diferentes definições de esquema, permitindo a navegação retroativa e proativa entre as versões para realização de operações de alterações e consultas. Assim,

para cada versão de esquema, versões de objetos são derivadas a fim de adequarem as instâncias atingidas pelas modificações à nova estrutura definida para a versão de esquema.

Quando um esquema é ativado, versões de objetos são derivadas para que sejam adequadas as instâncias atingidas pelas modificações à nova estrutura definida para a versão do esquema. O processo de ativação recupera o estado da base de dados extensional do esquema que serviu de base para derivação para realizar o processo de propagação das mudanças. Como os esquemas passam por estágios de desenvolvimento, três situações podem ocorrer com respeito ao esquema que serviu de base para derivação:

- estável - como um esquema estável não possui uma base de dados associada a ele, o mecanismo de propagação de mudanças apenas cria uma extensão de dados associando a versão do esquema;

- consolidado ou ativo - como um esquema consolidado ou ativo possui dados "ativos", o mecanismo de propagação faz uma cópia da base de dados, sendo que os objetos e versões logicamente excluídos não são selecionados. Com respeito aos atributos e relacionamentos temporalizados apenas o valor atual é também recuperado também;

- congelado - como em um esquema congelado, todos os dados estão logicamente excluídos, o mecanismo de propagação deve iniciar considerando o instante anterior ao que o esquema foi congelado. Retornando a este estado, valem as mesmas regras estabelecidas para os esquemas consolidado ou ativo.

Finalizado o processo de propagação das mudanças e enquanto o esquema estiver no estado ativo, versões dos objetos podem ser criadas gradativamente pelo usuário, seguindo os requisitos definidos pelo TVM.

5.5 Considerações Finais

Este capítulo apresentou o Modelo Temporal de Versionamento com Suporte à Evolução de Esquemas como alternativa para o gerenciamento da evolução de esquemas em bancos de dados orientados a objetos com a utilização dos conceitos de tempo e de versão. A utilização do conceito de versão no tratamento de evolução de esquemas permite o desenvolvimento de esquemas conceituais onde refinamentos são acrescentados gradualmente ao esquema sem causar danos às aplicações em execução, sendo de fundamental importância no desenvolvimento de projetos. Embora a utilização de versões armazene as alternativas de projeto, nem todo o histórico das alterações realizadas é registrado. A fim de suprir tal necessidade, a utilização do conceito de tempo permite armazenar as informações relativas ao tempo no qual as alterações foram realizadas. Por exemplo, o usuário pode estar interessado o estado de um projeto relativo a um período ou data específica, isso incluindo informações que foram descartadas por qualquer motivo. Esse tipo de recuperação não é possível com o uso apenas de versionamento, pois o histórico completo somente é acessível através do conceito de tempo. A característica ímpar do modelo é o gerenciamento do histórico da evolução da base de dados intencional e extensional homogênea e simultaneamente.

A definição de restrições visa garantir a validade do esquema frente às modificações realizadas. A cada operação de modificação, esse mecanismo é acionado, quando

tanto a consistência das versões como a hierarquia de derivação é garantida pela verificação imediata das primitivas de atualização, poupando tempo e evitando que um grande número de operações precise ser desfeito. Entretanto, este capítulo não aborda questões relativas ao tratamento das restrições comportamentais, ou seja, modificações em métodos e aplicações. Esses requisitos foram tratados em trabalhos anteriores não sendo extensivamente expostos neste capítulo. O tratamento da evolução dos métodos está documentado em (GALANTE, 1998; GALANTE; SANTOS; RUIZ, 1998).

6 ATUALIZAÇÃO DE OBJETOS

Este capítulo apresenta a linguagem que especifica o processo de atualização de objetos[1] durante o versionamento temporal de esquemas. A linguagem é apresentada através de uma série de exemplos que mostram como o usuário pode expressar operações de modificação de dados, por exemplo `insert`, `update` e `delete`, considerando a vigência de versões no banco de dados intencional e extensional.

A principal contribuição da linguagem é a possibilidade de expressar modificações que envolvem diferentes versões do esquema conceitual simultaneamente, denominadas atualizações multi-esquemas. Essas modificações são vitais para explorar as potencialidades do versionamento de esquemas, provendo um maior grau de liberdade para projetistas, desenvolvedores de aplicações e usuários finais, à medida que garante a reutilização dos dados e o suporte continuado de aplicações legadas, após serem realizadas modificações no esquema conceitual. São apresentadas, ainda, as premissas para a construção de uma linguagem de consulta para ambientes com evolução de esquemas, identificando os requisitos necessários para tal gerenciamento.

Um artigo apresentando a Linguagem de Atualização de Objetos bem como os primeiros passos da semântica operacional proposta para a especificação das operações de atualização, foi publicado nos anais do *XVIII Simpósio Brasileiro de Banco de Dados (SBBD 2003)* (GALANTE et al., 2003). Diferentemente da ordem apresentada no artigo, a semântica operacional proposta para a Linguagem de Atualização de Objetos é apresentada separadamente no capítulo 7, incorporada à semântica das demais linguagens do TVSE.

6.1 Linguagem de Atualização

Esta seção introduz as principais peculiaridades com respeito à sintaxe da Linguagem de Atualização de Objetos. A linguagem está separada em duas partes: (*i*) atualizações que manipulam uma única versão de esquema; e (*ii*) atualizações multi-esquemas, ou seja, aquelas atualizações que manipulam diversas versões do esquema simultaneamente. A Figura 6.1 apresenta a sintaxe[2] simplificada da Linguagem de Atualização de Objetos proposta para o TVSE. A BNF completa da linguagem pode ser conferida no Anexo C.

A recuperação das versões de esquema é feita através da cláusula

[1] Cabe relembrar que a linguagem de atualização de objetos é parte integrante da linguagem TVL/SE (seção 5.3). As porções da linguagem são apresentadas separadamente por questões didáticas, ou seja, facilitar a ordem do conteúdo apresentado nesta tese.

[2] De forma análoga à linguagem TVL/SE, apresentada na seção 5.3, a gramática que define a Linguagem de Atualização de Objetosé apresentada em inglês a fim de manter a coerência com o artigo publicado em (GALANTE et al., 2003).

```
<query>  ::= ( <insert> | <update> | <delete> ) <schema version selection>
<insert> ::= INSERT INTO <class name>
               [(<attribute name>=<expression> [VALIDITY INTERVAL<defined interval> ]
               [,<attribute name>=<expression> [VALIDITY INTERVAL<defined interval> ]]*)]
               VALUES ( <constant> [, <constant> ]*)
               [VALIDITY INTERVAL defined interval]
           [VALIDITY INTERVAL <defined interval> ]
<delete> ::= DELETE FROM <class name> [WHERE <condition>][fValidity=<instant value>]
<update> ::= UPDATE <class name>
               SET<attribute name>=<expression>[VALIDITY INTERVAL <defined interval>]
               [,<attribute name>=<expression> [VALIDITY INTERVAL <defined interval>]]*
               WHERE <condition>
```

Figura 6.1: Sintaxe simplificada para Linguagem de Atualização de Objetos

<schema version selection>, cuja gramática foi explicitada na seção 5.3.1.1.1. A execução das operações de atualização requer a seleção prévia de uma ou mais versões do esquema. Relembrando, as versões de esquema podem ser recuperadas implicitamente, caso em que a versão corrente é selecionada, ou explicitamente através do comando SET SCHEMA. A utilização do comando SET SCHEMA permite a seleção de uma única versão, através da especificação do nome do esquema. Alternativamente, pode ser informado um intervalo de transação e/ou de validade, resultando na seleção de uma ou mais versões do esquema. Nesse caso, a operação de atualização é efetuada em todas as versões de esquema selecionadas que estiverem no estado ativa, representando uma atualização denominada multi-esquema. Entretanto, cabe salientar que as operações de atualização de objetos devem ser aplicadas somente em versões que se encontram no estado ativa, uma vez que é o único estado em que a base de dados extensional pode ser populada. Quando, entre as versões de esquemas selecionadas, existir uma versão de esquema que se encontra em um estado diferente de ativa, a operação de atualização não deve ser executada em tal versão de esquema.

A recuperação de dados temporais versionados, considerando cada uma das versões de esquema selecionadas, deve ser realizada através da linguagem de consulta TVQL (cláusula <condition>), apresentada na seção 3.3. Por exemplo, para a modificação (UPDATE) ou a exclusão (DELETE) de um objeto, este deve ser primeiro selecionado através de uma expressão de consulta especificada pela cláusula WHERE. Nesse caso, todas as funcionalidades presentes na linguagem TVQL devem ser consideradas, como por exemplo, a recuperação de objetos, versões e atributos temporais e não temporais. No contexto da Linguagem de Atualização de Objetos, após ter sido selecionada uma versão (ou diversas versões) do esquema, a linguagem TVQL deve ser plenamente utilizada para recuperação dos dados da extensão, satisfazendo os critérios da consulta. A linguagem TVQL foi selecionada uma vez que a base de dados extensional permite a existência de objetos convencionais e objetos temporais versionados.

É importante evidenciar que o tempo de validade das versões de esquema está restrito ao tempo presente e futuro. A hipótese de alterar o passado das versões de esquema foi descartada, uma vez que esse tipo de modificação não faz sentido, pois um esquema modela a realidade de um sistema, e a alteração do passado de tal esquema não representaria corretamente tal realidade. Por outro lado, as operações de atualização de objetos devem realizar modificações retro e pró-ativas, ou seja, são permitidas modificações no presente, passado e futuro da base de dados extensional, desde que tais atualizações respeitem as restrições temporais estabelecidas entre as versões de esquema e seus dados associados. Isso significa que o intervalo de validade definido para a operação

de atualização de objetos deve estar contido no tempo de vida da versão de esquema correspondente (vide seção 5.1.2).

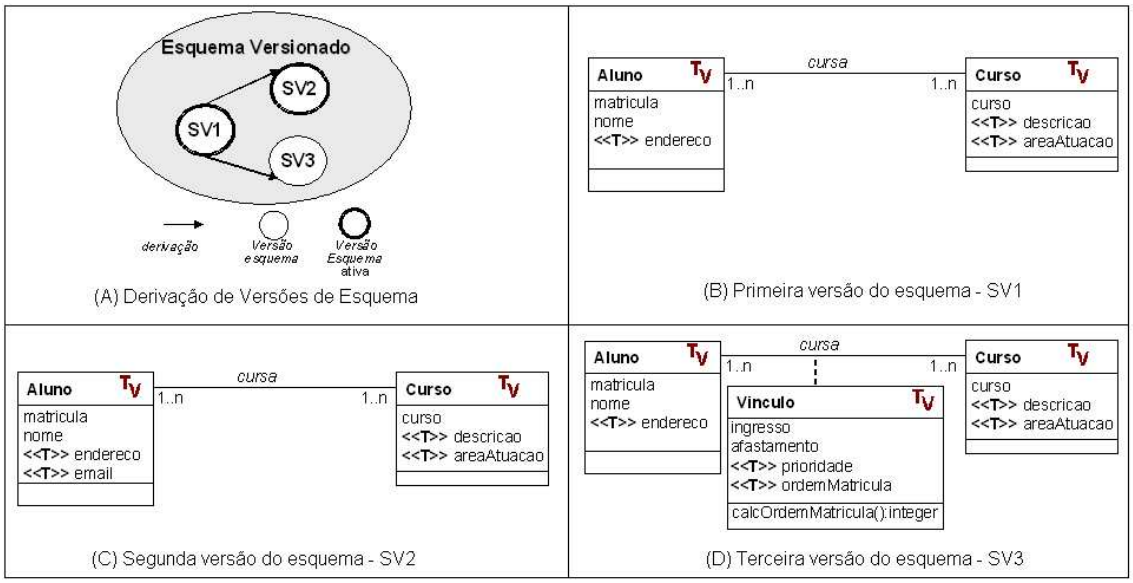


Figura 6.2: Derivação de Versões de Esquemas - Sistema Discente da UFRGS

As subseções a seguir ilustram o uso da linguagem através de uma série de exemplos, apresentando uma discussão completa a respeito de todas as possibilidades de atualização de objetos, versões de objetos, atributos normais e atributos temporais, considerando a vigência de versões de esquemas. Para uma exemplificação mais concreta, as atualizações serão efetuadas considerando a evolução do Sistema Discente da UFRGS, discutido extensivamente nos capítulos 3 e 4 . Para simplificar, a Figura 6.2 ilustra apenas parte da evolução do Sistema Discente. Duas modificações são aplicadas na versão inicial do esquema (SV1), Figura 6.2-b, produzindo duas novas versões para o esquema: SV2 - Figura 6.2-c e SV3- Figura 6.2-d, respectivamente. A Figura 6.2-a mostra o grafo de derivação com todas as versões do esquema. As linhas duplas indicam as versões que se encontram no estado ativa. A versão SV2 é obtida a partir do SV1 pela adição de um atributo temporal, email, na classe Aluno. A versão SV3 também é obtida a partir do SV1 pela inclusão de uma classe, Vínculo, estabelecendo um relacionamento entre as classes Aluno e Curso.

6.1.1 Insert

O comando **INSERT** é utilizado para criar a primeira versão de um objeto. O intervalo de validade do novo objeto pode ser especificado pela cláusula **VALIDITY INTERVAL**. Quando o intervalo de validade não é especificado, assume-se o valor atual do sistema (*now*) para o tempo de validade inicial, ficando o tempo de validade final em aberto (*null*), o que significa que o objeto permanece válido até que o valor seja fornecido. De forma análoga, na presença de atributos temporais, o usuário pode recursivamente informar o intervalo de validade de cada um deles. Por fim, a operação **INSERT** pode ser igualmente utilizada para inserir objetos convencionais na base de dados, como por exemplo, a inserção de objetos pertencentes a classes normais (sem tempo e sem versões). Em tal situação, como a classe não tem informações temporais associadas, os intervalos de validade, quando fornecidos pelo usuário, não são considerados. O Exemplo 6.1 ilustra

a inserção de um objeto em uma classe temporal versionada. Nesse caso, a inserção representa a criação da primeira versão para o objeto.

Exemplo 6.1 Suponha que um objeto é inserido na classe `Aluno`, na primeira versão do esquema (`'Esquema,1'`). A inserção do objeto usando a linguagem seria:

```
INSERT INTO Aluno
(Aluno.matricula = 'DC 103/99'
 Aluno.nome = 'Maria da Graça'
 Aluno.endereco = 'Bento Gonçalves, 1000'
                     VALIDITY INTERVAL ["01/02/2004","null"])
 VALIDITY INTERVAL ["01/01/2004","31/05/2004"]
 SET SCHEMA 'SV1'
```

Note que, como o atributo endereço é temporal, o intervalo de validade é informado, o que não ocorre com os demais atributos que não são temporais. ■

6.1.2 Update

O comando **`UPDATE`** permite alterar o conteúdo dos atributos de um objeto ou versão. Esta operação envolve duas ações distintas, dependendo se o atributo modificado é temporal ou não temporal. Quando um atributo não temporal é modificado, uma nova versão do objeto é derivada, contendo à alteração realizada. Por outro lado, na alteração de um atributo temporal, os dois valores são mantidos para a mesma versão do objeto: o valor antigo e o novo valor, este último correspondendo à alteração realizada. De forma análoga à operação de inserção, para os atributos temporais o intervalo de validade pode ser informado juntamente com a modificação do valor do atributo. Por fim, para a alteração de objetos convencionais, o comando **`UPDATE`** tem equivalente em SQL, substituindo o valor antigo pelo novo valor. O Exemplo 6.2 ilustra a alteração do endereço do aluno inserido no exemplo anterior (Exemplo 6.1).

Exemplo 6.2 Suponha que o endereço do versão corrente da aluna 'Maria da Graça' é alterado, com validade inicial a partir de "01/03/2004", considerando a primeira versão do esquema (`'Esquema,1'`). A alteração do endereço usando a linguagem seria:

```
UPDATE Aluno
SET a1.endereco = 'Protasio Alves, 1500'
                          VALIDITY INTERVAL ["01/03/2004","null"]
WHERE (SELECT a1.endereco
               FROM Aluno.version a1
               WHERE a1.nome = 'Maria da Graça' and a1.tvoid = current)
SET SCHEMA 'SV1'
```

Note que a recuperação da versão do objeto a ser modificada é realizada utilizando uma consulta TVQL. O atributo `a1.tvoid=`**`current`** recupera a versão corrente do objeto solicitado. ■

Considerando bancos de dados bitemporais, os valores dos atributos não são fisicamente modificados, exceto aqueles que estão com o tempo de transação em aberto (ELMASRI; NAVATHE, 2000). As informações não são perdidas ou excluídas, uma vez que o tempo de transação armazenado para os dados possibilita a distinção entre duas ou mais informações que possuam o mesmo tempo de validade. Assim, qualquer definição de um novo valor é realizada com o tempo de transação definindo a semântica das informações. A Tabela 6.1 ilustra a atualização do atributo `endereco`, apresentado os dados fisicamente armazenados. Cada *tupla* da tabela contém os valores dos rótulos

temporais associados: tempo de transação inicial e final e tempo de validade inicial e final, respectivamente.

Tabela 6.1: Exemplo de atualização de valores de atributos bitemporais

	Endereco	**tTimei**	**tTimef**	**vTimei**	**vTimef**
A	Bento Gonçalves, 1000	'05/10/2003'	'02/07/2004'	'01/01/2004'	*null*
B	Bento Gonçalves, 1000	'02/07/2004'	*null*	'01/01/2004'	'01/07/2004'
C	Protásio Alves, 1500	'02/07/2004'	*null*	'02/07/2004'	*null*

A Tabela 6.1 mostra que o endereço `Bento Gonçalves, 1000` foi armazenado no dia `05/10/2003`, começando a representar a realidade em `01/01/2004`. Supondo que o endereço tenha sido alterado para `Protário Alves, 1500` no dia `01/07/2004`, sendo a base de dados atualizada no dia seguinte, encerrando a validade do primeiro endereço, as seguintes atualizações físicas são aplicadas à tabela:

- o valor atual da informação é copiado (**`B`**); `B.tTimei` recebe o tempo da transação, `B.vTimef` recebe o `vTimei` da informação nova menos 1 (instante), `B.tTimef` recebe *null*, assim `B` é uma cópia do valor corrente anterior (**`A`**) depois do mesmo ter sido encerrado em tempo de validade `vTimei -1`;
- o valor atual da informação é novamente copiado (**`C`**); `C.vTimei` recebe o tempo de validade informado, `C.vTimef` recebe *null*, **`C.endereco`** recebe a nova informação, no qual **`C`** representa a informação corrente;
- `C.tTimef` será atualizado quando a informação corrente não representar mais a realidade.

6.1.3 Delete

O comando **`DELETE`** permite excluir um objeto (temporal versionado ou não) ou uma versão. Esta operação realiza uma exclusão lógica, na qual o tempo de validade final é encerrado com o valor especificado no momento da remoção, através da cláusula **`vValidity`**. Quando nenhum valor é especificado para a cláusula **`vValidity`**, assume-se o valor atual do sistema, ou seja, o tempo presente *now*. Quando uma versão é removida, seu tempo de vida deve ser encerrado e, na existência de atributos temporais, estes devem ser recursivamente excluídos. De forma similar, quando um objeto versionado é excluído, todas as suas versões devem ser recursivamente excluídas, bem como os atributos temporalizados de cada versão. Por fim, para a exclusão de um objeto convencional, a semântica do comando **`DELETE`** é equivalente à de SQL, na qual o objeto é fisicamente removido da base de dados. O Exemplo 6.3 ilustra a exclusão do endereço da aluna cujo nome é `Maria da Graça`.

Exemplo 6.3 Suponha que a última versão da aluna `'Maria da Graça'` seja excluído, com validade para `02/02/2004`, considerando a primeira versão do esquema. A exclusão da versão, usando a linguagem seria:

```
DELETE FROM Aluno.version a1
WHERE a1.nome = 'Maria da Graça' and
      a1.tvoid = lastVersion and
      a1.fvTime = null)
fValidity= '02/02/2004'
SET SCHEMA 'SV1'
```

■

6.1.4 Atualizações Multi-Esquemas

As operações **INSERT**, **UPDATE** e **DELETE** foram previamente exemplificadas, tendo como área de aplicação apenas uma versão do esquema conceitual. Entretanto, a utilização da cláusula **SET SCHEMA** pode resultar na seleção de mais de uma versão de esquema, como, por exemplo, quando um intervalo de transação e/ou validade é especificado. A linguagem de atualização permite expressar modificações que envolvem diferentes versões do esquema conceitual simultaneamente, sendo denominadas atualizações multi-esquemas. Neste caso, a atualização é recursivamente repercutida em todas as versões de esquema contidas no intervalo de tempo informado (somente para aquelas versões de esquema que se encontram no estado ativa). O exemplo 6.4 ilustra a alteração do endereço de um aluno em duas versões do esquema simultaneamente.

> **Exemplo 6.4** Suponha que o endereço da versão corrente da aluna 'Maria da Graça' é alterado para 'Ipiranga, 1500', com validade inicial a partir de '01/05/2004'. A seleção do(s) esquema(s) segue(m) o intervalo de validade compreendido entre '01/01/2004' até '31/12/2004'. A atualização multi-esquema usando a linguagem seria:
>
> ```
> UPDATE Aluno
> SET a1.endereco = 'Ipiranga, 1000' VALIDITY INTERVAL ['01/05/2004','null']
> WHERE (SELECT a1.endereco
>                FROM Aluno.version a1
>                WHERE a1.nome = 'Maria da Graça' and a1.tvoid = current)
> SET SCHEMA VALIDITY INTERVAL =['01/01/2004','31/12/2004']
> ```
>
> Note que a operação é executada recursivamente em todas as versões de esquema que respeitam o intervalo de validade especificado na cláusula **SET SCHEMA**. ■

Na presença de atualizações multi-esquemas, o sincronismo adotado para o gerenciamento entre base de dados intencional e extensional é o gerenciamento síncrono (CASTRO; GRANDI; SCALAS, 1997). Nesse caso, cada versão de esquema possui suas respectivas instâncias associadas. Assim, a atualização é sempre realizada de acordo com a versão de esquema correspondente, através da utilização das funções de adaptação definidas na seção 5.4.1. Este tipo de gerenciamento é denominado síncrono porque a pertinência temporal de uma versão do esquema deve incluir a pertinência temporal dos dados correspondentes, tornando-os dependentes entre si. Os trabalhos analisados no capítulo 2 não gerenciam atualizações multi-esquema, colocando a Linguagem de Atualização de Objetos apresentada neste capítulo em destaque com relação às demais abordagens presentes na literatura. O TVSE gerencia atualizações multi-esquemas, adotando o gerenciamento síncrono entre a intenção e a extensão. Esta é uma das razões que permite que o mecanismo seja capaz de atualizar diversas versões de esquema concomitantemente. Assim, desde que o usuário informe um intervalo de validade, o mecanismo recursivamente propaga a atualização em todas as versões de esquema recuperadas. É importante ressaltar que cada versão de esquema é atualizada independentemente. A vantagem deste mecanismo é que um mesmo objeto pode existir em mais de um repositório, sendo as operações de atualização especificadas em uma única cláusula SQL. Além disso, um objeto pode assumir valores diferentes para os mesmos atributos em versões diferentes de esquema. Esse tipo de situação é comumente requerida em sistemas legados que sofrem alterações freqüentes, requerendo, conseqüentemente, a habilidade de manipular diversas versões do esquema juntamente com seus dados associados.

6.2 Integração das Operações de Atualização com as Operações de Modificação de Esquemas

Quando um esquema não sofre modificações, a integridade dos dados não é caracterizada como um problema, visto que o mecanismo de gerenciamento de transações do SGBD é suficiente para garantir o cumprimento de tal consistência. Entretanto, quando um esquema evolui constantemente, o gerenciamento de evolução de esquemas deve ser responsável por gerenciar não somente as operações de modificação de esquemas e propagação de mudanças, mas também as operações de atualização de dados.

Um dos diferenciais do TVSE consiste no fato de prover um mecanismo independente entre a linguagem de modificação de esquemas (seção 5.3.1.2) e a linguagem de atualização de objetos (apresentada neste capítulo). Durante o mecanismo de propagação de mudanças é efetuada uma reorganização automática das instâncias, sucessivamente, após as atualizações do esquema, garantindo sempre a consistência entre cada versão do esquema e seus dados associados. Assim, não é necessário especificar nenhum tipo de verificação estática (por exemplo, verificar a compatibilidade de tipos entre atributos) para garantir que o estado resultante do banco de dados mantenha a consistência entre esquema e base de dados. Neste caso, a linguagem de atualização de objetos atua diretamente na base de dados convertida de acordo com o seu respectivo esquema, dispensando qualquer outro tipo de verificação.

Entretanto, o desafio consiste em permitir a execução concorrente das operações de modificação de esquemas, de propagação de mudanças e de atualização da base de dados. Por exemplo, a manipulação de objetos através das operações `insert`, `update` e `delete` poderia ser realizada enquanto uma operação de modificação de esquemas é aplicada em um esquema. Diversas técnicas empregam mecanismos de bloqueio, para gerenciar a evolução de esquemas, como, por exemplo (LAGORCE; STOCKUS; WALLER, 1997; GORALWALLA et al., 1997), restringindo a concorrência e limitando assim as potencialidades do SGBD. Dentro desse contexto, há uma relação direta entre evolução de esquemas, processamento de transações e mecanismos de concorrência.

Uma vez que transações são construtores que alteram o estado da base de dados, em (GALANTE; EDELWEISS; SANTOS, 2002b) é apresentado um estudo preliminar a respeito dos efeitos da atualização de dados durante o processo de evolução de esquemas e de propagação de mudanças para o TVSE. O resultado deve ser um mecanismo de controle de concorrência que gerencie as operações de modificação de esquemas, o processo de propagação de mudanças na base de dados extensional e as operações de atualização de dados. Tal mecanismo poderia ser especificado como uma transação normal de banco de dados. Nesse estudo foram identificados os primeiros requisitos para tal controle:

- especificação do processamento de transações - especificar como as operações de modificação de esquemas e atualização de dados podem ser manipuladas como transações normais de banco de dados, considerando o versionamento do banco de dados intencional e extensional;

- definição de restrições de integridade - especificar as regras de integridade que garantem que o banco de dados resultante é sempre consistente e correto. Além das regras normais de acesso aos dados, presentes nos SGBDs, devem ser consideradas as restrições temporais e de versionamento;

- especificação do mecanismo de controle de concorrência - especificar as

técnicas para execução concorrente das operações de modificação de esquema, de propagação de mudanças e de atualização de dados.

Como nosso estudo nesta área é relativamente recente, estando fora do escopo desta tese, resultados concretos a respeito do controle de concorrência e do processamento de transações, considerando ambientes com evolução de esquemas, serão apresentados em artigos futuros, como extensão do TVSE apresentado nesta tese.

6.3 Premissas para a Linguagem de Consulta

No contexto de bancos de dados relacionais, várias linguagens de consulta temporal foram propostas nos últimos anos, como por exemplo, TOSQL (ARIAV, 1986), TempSQL (GADIA, 1992), TQuel (SNODGRASS, 1987), TSQL (NAVATHE; AHMED, 1989), entre outras. Porém, a que atingiu maior popularidade foi a linguagem TSQL2 (SNODGRASS, 1995). Dentre suas funcionalidades está o suporte ao versionamento de esquemas de tempo de transação (única dimensão temporal suportada), de acordo com o modelo apresentado em (RODDICK, 1994). A linguagem TSQL2 é estendida em (CASTRO; GRANDI; SCALAS, 1997) a fim de suportar o versionamento de tempo de validade e bitemporal. Outra abordagem proposta em (MOREIRA, 1999) trata exclusivamente do processamento de consulta com suporte ao versionamento de esquemas. O mecanismo proposto suporta qualquer tipo de banco de dados temporal e está de acordo com os princípios da linguagem TSQL2. No campo de bancos de dados orientados a objetos, a linguagem de consulta TQL (PETERS et al., 1993), proposta para o modelo TIGUKAT (ÖZSU et al., 1995), merece destaque por permitir a recuperação do histórico da evolução tratando uniformemente versões de esquemas e de objetos.

Analisando a literatura atual, constata-se que não existe uma linguagem de consulta que aborde homogeneamente os conceitos de tempo e de versão. Devido à complexidade para especificação dos requisitos essenciais de uma linguagem de consulta para ambientes que suportam evolução e versionamento de esquemas, bem como por limites de espaço e tempo, a especificação de tal linguagem está fora do escopo desta tese. Entretanto, esta seção elenca as premissas para a construção de uma linguagem de consulta para o TVSE.

A linguagem de consulta do TVSE deve permitir, além de consultas básicas realizadas pela linguagem-padrão OQL, novas consultas que retornem valores específicos de tempo e de versão, obviamente considerando um ambiente de evolução e versionamento de esquemas e de objetos. Assim, é desejável que a linguagem estabeleça um comportamento o mais homogêneo possível para os elementos convencionais (por exemplo, esquemas e objetos normais) e temporais versionados (por exemplo, versões temporais de esquemas e de objetos).

Dentro desse contexto, além dos princípios básicos de consistência, clareza, minimilidade, ortogonalidade e independência aplicados nas linguagens SQL e TSQL, outros princípios para o versionamento de esquema e de objetos devem ser considerados. Uma possível solução seria separar a linguagem de consulta em três áreas de abrangência distintas: consulta aos dados da intenção, consulta aos dados da extensão e consultas híbridas, envolvendo as versões de esquemas e seus respectivos dados associados.

Em um ambiente com suporte ao versionamento de esquemas, os dados da intenção não são descartados. Assim, um requisito essencial é que a linguagem de consulta forneça comandos que possibilitem consultas à estrutura das diversas versões de esquema, ou seja, a linguagem deve prover mecanismos para o gerenciamento de consultas aos dados da intenção.

Entre as características estruturais estão consultas sobre:

- a estrutura das versões de esquema;
- as modificações sofridas por um esquema.

Entre as características de tempo estão consultas sobre:

- a estrutura das versões de esquemas atuais ou em determinados instantes ou períodos de tempo;
- as modificações sofridas por um esquema ao longo do tempo;
- o histórico da vida de um esquema versionado e suas versões (isto é, o tempo em que permaneceu "ativo").

Entre as características de versionamento estão consultas sobre:

- os estados das versões de esquemas;
- versões correntes;
- navegação na hierarquia de derivação de versões, como, por exemplo, antecessores e sucessores de uma determinada versão de esquema, ou primeira e última versão.

Unindo as características de tempo e de versão, estão consultas sobre:

- os estados das versões de esquemas sobre determinados instantes ou períodos de tempo;
- hierarquia de derivação de versões em determinados instantes ou períodos de tempo;
- versões correntes em determinados instantes e períodos de tempo.

As consultas sobre os dados da intenção devem ser executadas sobre a estrutura de metadados que contém todas as informações a respeito da evolução de cada esquema versionado e de suas versões. Assim, os requisitos da linguagem de consulta devem cobrir todas as informações especificadas na estrutura de metadados. A estrutura de metadados encontra-se especificada na seção 5.1.4.

O gerenciamento de consulta aos dados da extensão diz respeito ao fato de múltiplas versões de esquema coexistirem em uma mesma base de dados, possibilitando a recuperação de versões passadas, presentes e futuras. Entre as características da extensão estão consultas sobre:

- consultas utilizando o tempo de validade dos dados (linha de tempo pertencente à realidade);
- consultas utilizando o tempo de transação dos dados (tempo em que os dados foram armazenados no banco de dados);
- consultas fazendo referência a uma determinada versão de esquema, por meio de seu tempo de transação e/ou validade;

- consultas fazendo referência a uma determinada versão de esquema, por meio dos requisitos de versionamento de esquemas. Por exemplo, pelo nome da versão, pelo estágio de desenvolvimento, entre outros.

Obviamente, as consultas sobre os dados da extensão devem considerar todas as funcionalidades presentes na linguagem TVQL (MORO et al., 2002). Isso significa que, após selecionar a versão (ou versões) do esquema, a linguagem TVQL deve ser plenamente utilizada para a recuperação dos dados da extensão, satisfazendo os critérios da consulta. A adoção de tais requisitos é fundamental, uma vez que na linguagem TVQL são tratadas todas as peculiaridades do TVM com respeito à recuperação de informações considerando objetos temporais versionados.

O gerenciamento de consultas híbridas diz respeito às consultas que envolvem múltiplos esquemas juntamente com seus dados associados. Entre os requisitos principais estão consultas sobre:

- possibilidade de especificar se a consulta deve utilizar uma, várias ou todas as versões do esquema;
- existência de cláusulas para seleção da versão do esquema a ser utilizada na resposta;
- possibilidade dos dados serem recuperados sob qualquer versão de esquema;
- existência de cláusulas para informar quando a resposta deve contemplar múltiplos esquemas;
- acomodar qualquer tipo de consulta, utilizando todas as dimensões temporais, como por exemplo, "Encontrar todos os alunos que obtiveram conceito A, na disciplina de BD, em 2003, como armazenado em 1 de janeiro de 2003, mostrando os resultados utilizando o esquema válido em março de 2004, conforme armazenado em janeiro de 2004".

O objetivo desta seção não é propor uma linguagem de consulta para ambientes com evolução de esquemas, mas sim identificar alguns requisitos necessários para tal gerenciamento. Cabe salientar que alguns desses requisitos já estão cobertos pela linguagem de atualização de objetos proposta neste capítulo. Por exemplo, a recuperação de objetos e versões atualizados pela linguagem de atualização de objetos faz uso da linguagem TVQL, considerando a vigência de diversas versões de esquemas. A conclusão principal é a necessidade de um mecanismo completo de processamento de consulta considerando a evolução do banco de dados intencional e extensional. Devido à sua complexidade, essa tarefa é deixada para uma extensão futura do TVSE.

6.4 Considerações Finais

O principal objetivo deste capítulo foi apresentar a Linguagem de Atualização de Objetos proposta para o TVSE. Essa linguagem define o processo de modificação de dados (inserção, atualização e exclusão) durante a vigência de versões temporais do esquema conceitual. A linguagem apresenta o seu diferencial no tratamento de atualizações que envolvem mais de uma versão do esquema do banco de dados (denominadas atualizações multi-esquema). Esse tratamento difere das propostas analisadas no levantamento

bibliográfico apresentado no capítulo 2. Nessas propostas, uma versão específica do esquema é sempre selecionada antes da realização das operações de atualização. A principal aplicação das atualizações multi-esquemas do TVSE concentra-se em bancos de dados legados que freqüentemente sofrem evolução de esquemas, requerendo, conseqüentemente, a habilidade de manipular diversas versões do esquema juntamente com seus dados associados.

A linguagem de atualização de objetos não trata a relação dos rótulos temporais (tempo de transação e tempo de validade) com relação às propriedades ACID (***A**tomicidade*, ***C**onsistência*, ***I**solamento* e ***D**urabilidade*). Essas propriedades são os critérios de corretude para transações instantâneas rodando em um nível de isolamento serializável. Assumindo que o SGBD possua os mecanismos necessários para garantir as propriedades ACID para transações instantâneas, tais propriedades serão mantidas igualmente para as transações temporais. Porém, estudiosos da área de bancos de dados temporais têm apresentado controvérsias em decorrência da manipulação do tempo de transação (JENSEN, 1999). Com base nos problemas identificados, é elencado um conjunto de requisitos para uma semântica temporal de transações consistentes. Dentro desse contexto, está sendo definido um sistema de tipos para as linguagens do TVSE, bem como a prova de importantes propriedades de corretude (por exemplo, se a linguagem é determinística ou não), estando, entretanto, fora do escopo desta tese. Contudo, as regras temporais inerentes ao TVSE são cobertas pela semântica operacional apresentada no capítulo 7.

A linguagem de consulta para o TVSE não foi definida, apenas seus requisitos principais foram identificados. Essa linguagem deve permitir que sejam gerenciadas todas as características presentes na linguagem de especificação e versionamento de esquemas. Obviamente, a linguagem deve considerar o versionamento temporal tanto no banco de dados intencional quanto no extensional. A sugestão mais simples seria, além de utilizar a linguagem TVQL, acrescentar as mesmas palavras reservadas, com respectivos significados, a uma linguagem base como OQL e TSQL, por exemplo.

7 SEMÂNTICA OPERACIONAL PARA TVL/SE

Este capítulo enriquece o universo da tese definindo uma semântica operacional[1] para um núcleo da linguagem TVL/SE. Esse núcleo, denominado simplesmente de L, preserva as principais funcionalidades de TVL/SE e também é composto de uma linguagem para versionamento de esquema (L1), uma linguagem para modificação de esquemas (L2) e uma linguagem para atualização de dados (L3).

Um artigo apresentando uma semântica operacional para a especificação das operações de atualização de dados frente ao processo de evolução de esquemas foi publicado nos anais do *XVIII Simpósio Brasileiro de Banco de Dados (SBBD 2003)* (GALANTE et al., 2003). Além disso, um minicurso sobre semântica operacional e sistemas de tipos para linguagens de banco de dados foi apresentado no *VI Workshop de Métodos Formais* (WMF 2003) (MOREIRA et al., 2003a).

7.1 Semântica Operacional

A importância em adotar uma semântica precisa para especificação de linguagens de programação é reconhecida desde a década de 60, com o desenvolvimento das primeiras linguagens de programação de alto nível (BAKKER, 1969; STEEL, 1966). O uso de uma semântica operacional - que especifica o comportamento dinâmico de linguagens de programação, descrevendo passo a passo "como" os programas são executados - foi defendido por McCarthy em (MCCARTHY, 1962) e refinado em referências, como, por exemplo, (LUCAS, 1972). Exemplos clássicos de linguagens de programação definidas através de semântica operacional são Algol 60 (LAUNER, 1968), PL/I (PL/I Definition Group, 1986) e CSP (PLOTKIN, 1983). Atualmente, a semântica operacional é uma técnica popular na comunidade de linguagens de programação e tem sido utilizada com sucesso para especificar, por exemplo, a linguagem SML (MILNER et al., 1997) e um subconjunto da linguagem MSIL, *Microsoft´s bytecode language for .NET* (YU et al., 2004).

Semântica operacional e sistemas de tipos têm sido uma das abordagens mais utilizadas para definir e investigar propriedades em linguagens de programação. O interesse no uso dessas duas técnicas para a definição de linguagens de banco de dados tem crescido significativamente nos últimos anos. Prova disso são pesquisas recentes envolvendo semântica operacional e sistemas de tipos para uma rigorosa formalização de linguagens de bancos de dados. Bierman (BIERMAN, 2003) propõe uma semântica operacional e um sistema de tipos para uma linguagem de consulta para bancos de dados

[1]Agradecimentos a Álvaro Freitas Moreira pelo refinamento das regras semânticas apresentadas neste capítulo.

orientados a objetos, estando relacionada com a semântica operacional e o sistema de tipos definidos para a linguagem MJ, um núcleo imperativo para a linguagem JAVA (BIERMAN; PARKINSON; PITTS, 2003). O trabalho apresentado em (COLAZZO et al., 2002) propõe um sistema de tipos para uma linguagem de consulta, utilizando esse sistema de tipos para verificar a coerência das consultas com a estrutura definida para o esquema. Em (SIMÉON; WADLER, 2003) é apresentada uma semântica formal para *XML Schema*. Outro exemplo bem conhecido é a definição, pelo W3C, de uma semântica operacional e de um sistema de tipos para as linguagens *XQuery* e *XPath* (DRAPER et al., 2003).

Conforme mencionado na seção 2.3, a inclusão de facilidades de evolução de esquemas em sistemas de bancos de dados envolve a solução de dois problemas fundamentais: a semântica de modificação de esquemas e a semântica de propagação de mudanças. A necessidade de uma definição formal torna-se mais evidente quando a abordagem deve satisfazer restrições complexas, como é o caso, por exemplo, da abordagem TVSE que agrega os conceitos de versão e de tempo para gerenciar um ambiente com suporte a evolução de esquemas. Nesse sentido, esse capítulo especifica formalmente o comportamento dinâmico das linguagens de versionamento e modificação de esquemas, e atualização de dados propostas propostas para o TVSE. O objetivo é prover uma maior fundamentação para descrever as políticas de evolução de esquemas, fornecendo também uma base teórica e matemática que permite uma comparação segura com os demais modelos e abordagens propostos na literatura.

Uma semântica operacional para o núcleo da linguagem TVL/SE é apresentada nas próximas seções. O núcleo da linguagem, denominado simplesmente L, preserva as principais funcionalidades de TVL/SE, sendo composto por três sub-linguagens: L1, uma linguagem para versionamento de esquema; L2, uma linguagem para modificação de esquemas; e L3, uma linguagem para atualização de dados.

Intuitivamente, a semântica operacional utilizada para a linguagem L consiste de um conjunto de regras de inferência definidas da seguinte forma:

$$\frac{\textit{premissa} \;..\; \textit{premissa}}{\textit{conclusão}}$$

Um regra possui um número de premissas (escritas acima da linha) e uma conclusão (escrita abaixo da linha). Um regra pode conter também um número de condições (escritas do lado direito da linha), que devem ser satisfeitas sempre que a regra for aplicada. Regras com um conjunto vazio de premissas são denominadas axiomas, sendo que a linha pode ser omitida. A leitura para a regra é a seguinte "se as *premissa .. premissa* são verdadeiras então a *conclusão* é verdadeira".

Tipicamente, as premissas e a conclusão das regras semânticas são especificadas através de uma relação de transição (representada por $\rightarrow$), que descreve o relacionamento entre o estado inicial e o estado final da execução de um comando da linguagem. Por exemplo, uma transição é descrita da seguinte forma:

$$\textit{configuração} \;\rightarrow \textit{configuração}'$$

Uma relação de transição descreve, portanto, como a execução do comando da linguagem ocorre, ou seja, é um passo da execução que transforma uma configuração inicial em outra resultante. A relação $\rightarrow$ avalia a configuração presente no lado esquerdo da seta, produzindo a configuração posicionada ao lado direito da seta. As configurações

são interpretadas da seguinte forma: "a configuração *configuração′* resulta da avaliação da configuração *configuração*".

Exemplificando, a Figura 7.1 ilustra a construção de regras semânticas para o comando `IF`. A primeira regra, IF-true, expressa a avaliação do comando `IF` e `THEN` e_1 `ELSE` e_2, no qual e_1 é executado quando e é avaliado para verdadeiro. A segunda regra, IF-false, simplesmente mostra que e_2 é executado quando e é avaliado para falso.

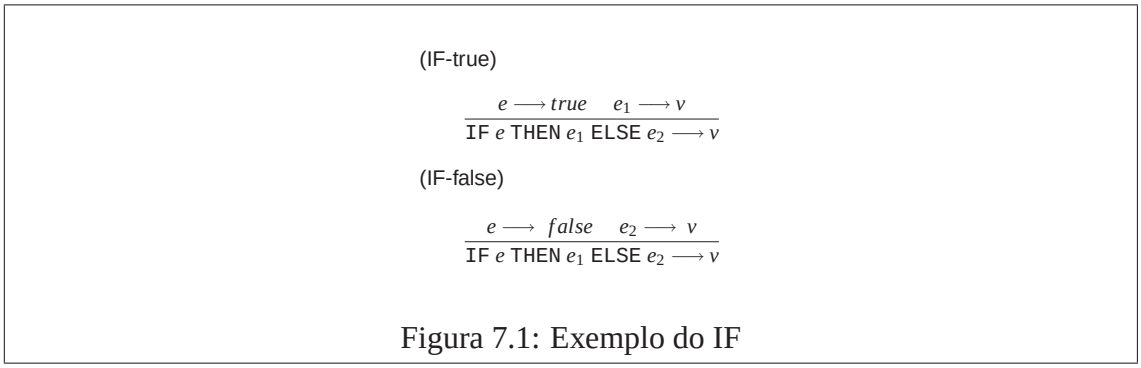


Figura 7.1: Exemplo do IF

7.1.1 A Linguagem L

A semântica operacional é definida para um subconjunto da linguagem TVL/SE, denominado linguagem L. Esta prática, tradicional na semântica de linguagens de programação, facilita o tratamento formal e foi usada, por exemplo, por Bierman (BIERMAN, 2003) na definição da semântica da linguagem de consulta OQL definida pelo ODMG

A linguagem L considerada para o tratamento formal inclui:

- um subconjunto da linguagem para especificação e versionamento de esquemas da linguagem TVL/SE, denominada L1;
- a linguagem de atualização de objetos da linguagem TVL/SE, denominada L3;

Cabe salientar que a linguagem L é essencialmente a mesma apresentada nas seções 5.3 e 6.1, respectivamente, preservando as características fundamentais. Entretanto, para tornar mais compacta a formalização a linguagem sofreu modificações de natureza sintática, identificadas ao longo do texto.

7.2 Semântica Operacional para Linguagem L

Esta seção apresenta a semântica operacional da linguagem L. Para cada porção da linguagem é apresentada uma gramática, seguida das regras semânticas referentes as principais operações.

As operações de L pertencem ao conjunto de operações produzido pela gramática a seguir, onde *op* representa todas as operações da linguagem L. Os índices identificam cada subconjunto da linguagem: L1 representa a linguagem de especificação e versionamento de esquemas (Seção 5.3.1.1), L2 a linguagem de modificação de esquemas (Seção 5.3.1.2) e L3 representa a linguagem de atualização de objetos (Seção 6.1). As operações da linguagem atuam sempre sobre uma versão específica de esquema, representada por *sv*. Operações de L1 também podem ser feitas sobre a versão corrente (*CurrentOf*()) e sobre o conjunto de versões válidas em um intervalo de tempo *intv*.

$$
\begin{array}{rcl}
op & ::= & op_{L1}\ sv \\
& | & op_{L2}\ sv \\
& | & op_{L3}\ (sv \mid \texttt{current} \mid intv) \\
& | & op\ op \\
& | & \texttt{nop}
\end{array}
$$

A Figura 7.2 apresenta as regras da semântica operacional para a linguagem L. As operações de L atuam sobre um banco de dados temporal versionado DB composto por um par (SCH, EE), onde $\mathsf{SCH} = (\mathsf{SV}, sv_{\text{fst}}, sv_{\text{lst}})$ representa o grafo de derivação de versões de um esquema. O componente SV contém os nodos do grafo e sv_{fst} e sv_{prd} identificam a primeira e a última versão. O componente EE representa um ambiente de extensões que mapeia os nomes dos esquemas para as suas extensões.

(L-Seq)

$$\frac{t, \mathsf{DB}, op_1 \underset{\mathsf{L}}{\longrightarrow} t', \mathsf{DB}', op_1'}{t, \mathsf{DB}, op_1\ op_2 \underset{\mathsf{L}}{\longrightarrow} t', \mathsf{DB}' op_1'\ op_2}$$

(L-nop)

$$t, \mathsf{DB}, \texttt{nop}\ op \underset{\mathsf{L}}{\longrightarrow} t', \mathsf{DB}, op$$

(L-L1)

$$\frac{t, (\mathsf{SCH}, \mathsf{EE}), op_{L1}\ sv \underset{\mathsf{L1}}{\longrightarrow} t', (\mathsf{SCH}', \mathsf{EE}'), \texttt{nop}}{t, (\mathsf{SCH}, \mathsf{EE}), op_{L1}\ sv \underset{\mathsf{L}}{\longrightarrow} t', (\mathsf{SCH}', \mathsf{EE}'), \texttt{nop}}$$

(L-L2)

$$\frac{t, \mathsf{SCH}, op_{L2}\ sv \underset{\mathsf{L2}}{\longrightarrow} t', \mathsf{SCH}', \texttt{nop}}{t, (\mathsf{SCH}, \mathsf{EE}), op_{L2}\ sv \underset{\mathsf{L}}{\longrightarrow} t', (\mathsf{SCH}', \mathsf{EE}), \texttt{nop}}$$

(L-L3)

$$\frac{\mathsf{SCH} \vdash t, \mathsf{EE}, op_{L3}\ sv \underset{\mathsf{L3}}{\longrightarrow} t', \mathsf{EE}', \texttt{nop}}{t, (\mathsf{SCH}, \mathsf{EE}), op_{L3}\ sv \underset{\mathsf{L}}{\longrightarrow} t', (\mathsf{SCH}', \mathsf{EE}'), \texttt{nop}}$$

(L-L3b)

$$\frac{\mathit{CurrentOf}(\mathsf{SCH}) = sv}{t, (\mathsf{SCH}, \mathsf{EE}), op_{L3}\ \texttt{current} \underset{\mathsf{L}}{\longrightarrow} t, (\mathsf{SCH}, \mathsf{EE}), op_{L3}\ sv}$$

(L-L3c)

$$\frac{\mathit{ValidOf}(\mathsf{SCH})\mathit{Between}(intv) = \{sv_1 \ldots sv_n\}}{t, (\mathsf{SCH}, \mathsf{EE}), op_{L3}\ intv \underset{\mathsf{L}}{\longrightarrow} t, (\mathsf{SCH}, \mathsf{EE}), op_{L3}\ sv_1 \ldots op_{L3}\ sv_n}$$

Figura 7.2: Regras para Linguagem L

A regra L-L1 mostra a avaliação da operação op_{L1} sobre a versão de esquema de nome *sv* e deve usar as regras que definem a relação $\underset{\mathsf{L1}}{\longrightarrow}$. A partir desta regra, observa-se que a operação pode alterar tanto o grafo de derivação de versões (SCH') quanto a base de dados extensional (EE'). Porém, a operação para ativar uma versão de esquema é a única operação da linguagem L1 que pode afetar a base de dados extensional (EE').

A regra L-L2 expressa a avaliação da operação op_{L2} sobre a versão de esquema de nome *sv* e deve usar as regras que definem a relação $\xrightarrow[\text{L2}]{}$. A avaliação das operações op_{L2} não afetam a base de dados extensional (EE) pois são aplicadas somente em versões de esquema que encontram-se no estado em trabalho, que, neste caso, não possuem uma extensão associada a elas.

A regra L-L3 expressa a avaliação da operação op_{L3} sobre a versão de esquema de nome *sv* e deve usar as regras que definem a relação da $\xrightarrow[\text{L3}]{}$. Observa-se, a partir desta regras, que a operação op_{L3} altera somente a base de dados extensional (EE'), mantendo inalterado o grafo de derivação de versões (EE). Entretanto, a premissa da regra é formada pelo grafo de derivação (SCH) pois as operações op_{L3} devem obrigatoriamente identificar pelo menos uma versão do esquema a partir do grafo.

7.2.1 Linguagem L1

A linguagem L1 é uma simplificação da linguagem para versionamento de esquemas apresentada na Seção5.3.1.1. Com L1 não é possível programar as operações de mudança de estado para um tempo futuro. As operações de L1 pertencem ao conjunto de operações produzidos pela seguinte gramática, onde *schema* é uma coleção de definições de classes.

$$
\begin{array}{rcl}
op_{L1} & ::= & \texttt{CREATE}\ \textit{schema}\ \textit{sv} \\
& | & \texttt{DERIVE}\ \textit{sv}\ \texttt{FROM}\ \textit{sv} \\
& | & \texttt{PROMOTE}\ \textit{sv} \\
& | & \texttt{ACTIVATE}\ \textit{sv} \\
& | & \texttt{BLOCK}\ \textit{sv} \\
& | & \texttt{FREEZE}\ \textit{sv} \\
& | & \texttt{RESTORE}\ \textit{sv}
\end{array}
$$

As regras da semântica operacional para a linguagem L1 estão nas figuras 7.3 e 7.4. A regra Create expressa a avaliação da operação para criar um esquema, que será raiz no processo de derivação. A operação é realizada em um tempo *t*, sobre um esquema versionado SCH e uma extensão EE. A avaliação da operação não afeta a base de dados extensional (EE). A segunda premissa indica que o nome *sv* da versão criada não pertence ao domínio do esquema versionado SCH. A partir desta regra, observa-se que o nodo criado não possui tempos associados, o estado é em trabalho (W), não possui predecessor e o conjunto de sucessores é vazio. A única alteração de SCH' indica que *sv* é definido como corrente.

A regra Derive expressa a avaliação da operação para derivar uma nova versão de esquema. SV é o grafo atual e SV(sv2) é o nodo versionado. Observa-se, a partir dessa regra, que o grafo SV' é modificado, onde SV_1 é o nodo recém criado, sendo que a única alteração em SV_2 é a inclusão do nodo filho a lista de sucessores do nodo pai.

A regra Promote simplesmente promove uma versão *sv* do estado em trabalho (W) para o estado estável (S) e estabelece o seu tempo de transação inicial *tti*.

A regra Block expressa a avaliação da operação para bloquear uma versão do esquema. O único efeito dessa regra sobre o esquema versionado *sv* é modificar o estado de A (ativo) para B (bloqueado).

A regra Freeze1 expressa a operação para congelar uma versão de esquema que encontra-se no estado estável. O efeito dessa operação é alterar o estado para F (congelado - *freeze*), finalizando a validade com o tempo em que a operação é executada adicionado de um instante (t_{+1}).

A regra Freeze2 é avaliada para os casos em que a versão de esquema encontra-se no estado A (ativo) ou C (consolidado). Neste caso, a versão de esquema possui uma extensão de dados associada a ela. Além de alterar o estado de *sv* para F (congelado), o efeito dessa regra é encerrar o tempo de validade da extensão associada a *sv*. Cabe salientar que *Finalize* é uma função auxiliar da semântica que encerra o tempo de validade da extensão, percorrendo recursivamente os objetos, suas versões, e os atributos e os relacionamentos temporais.

Apesar da aparente complexidade da regra devido aos subscritos, a regra Freeze3 simplesmente congela as versões do grafo a partir do nodo *sv* até as folhas.

A regra Restore1 muda o estado de uma versão *sv* sem dados de congelado (F) para estável (S). A regra Restore2 faz praticamente o mesmo, mas usa a função auxiliar *Restore* sobre as extensões temporais de *sv*. Essa função é análoga a função auxiliar *Finalize* da regra Freeze2. As regras Restore3 e Restore4 restauram a versão *sv* assim como todas as versões que a precedem no grafo de versões.

As regras Activate1 e Activate2 tornam ativa (A) uma versão estável *sv*. Caso essa versão estável seja o resultado da evolução de uma versão predecessora sv_{prd} ativa ou consolidada, a extensão ext_{prd} de sv_{prd} deve ser convertida de acordo com o esquema de *sv*. Essa conversão de dados, definida em 5.4, é aqui representada pela função auxiliar *Convert*.

7.2.2 Linguagem L3

A linguagem L3 é uma simplificação da linguagem para atualização de objetos apresentada na Seção 6.1. As operações de L3 pertencem ao conjunto de operações produzidos pela seguinte gramática, onde C é uma classe, $\overline{a = e\ intv}$ representa um conjunto de atributos temporais e $\overline{a = e}$ representa um conjunto de atributos não temporais.

$$
\begin{array}{rcl}
op_{L3} & ::= & \texttt{INSERT}\ C(\overline{a = e\ intv}, \overline{a = e})\ intv \\
& | & \texttt{INSERT INTO}\ C(\overline{a = e}) \\
& | & \texttt{UPDATE}\ C\ \texttt{SET}\ a = e\ intv\ \texttt{WHERE}\ q \\
& | & \texttt{UPDATE}\ C\ \texttt{SET}\ a = e\ \texttt{WHERE}\ q \\
& | & \texttt{DELETE FROM}\ C\ \texttt{WHERE}\ q\ t
\end{array}
$$

As regras da semântica operacional para a linguagem L3 estão na Figura 7.5. A regra Insere1 adiciona um novo objeto identificado por *oid* a extensão da classe C na versão *sv*. Os atributos do objeto são inicializados e para isso as expressões devem ser avaliadas por um conjunto de regras que definem a relação $\xrightarrow[e]{}$. A avaliação dessas expressões pode, por sua vez criar outros objetos, modificando extensões.

As regras Update1 e Update2 atualizam um objeto da classe C da versão *sv*. Se o objeto for normal ele é simplesmente modificado. Se ele for temporal, mas o atributo a ser modificado for normal, uma nova versão do objeto é criada.

Pela regra Update1, a atualização de atributo temporal de objeto temporal adiciona essa modificação de valor no histórico do atributo.

7.3 Considerações Finais

Este capítulo apresentou a semântica operacional para um subconjunto da linguagem TVL/SE.

Uma das dificuldades relacionada à semântica operacional para linguagens de bancos de dados em geral é a inexistência de um consenso que regularize a terminologia e notação adotadas. Na prática, trabalhos desenvolvidos nesta área têm utilizado notações e terminologias próprias (BRUCE, 2002). O estilo da semântica apresentada neste capítulo segue a semântica operacional padrão. Entretanto, termos relacionados a linguagens de programação orientadas a objetos e de banco de dados foram especificados especialmente para este trabalho. Por exemplo, a definição de um banco de dados foi intuitivamente especificada através de um par, representado por $(\mathsf{SCH}, \mathsf{EE})$, no qual SCH representa o grafo de derivação de versões e EE um conjunto com a a extensão associada a cada versão de esquema.

Por fim, é importante destacar que o trabalho nessa área é relativamente recente, não representando o núcleo de atuação desta tese, de modo que a semântica operacional é apresentada com um certo grau de abstração. Sendo assim, por restrições de tempo e de espaço, resultados concretos com respeito à especificação de um sistema de tipos para a linguagem TVL/SE está sendo desenvolvido paralelamente a esta tese, devendo aparecer em publicações futuras (MOREIRA et al., 2003b). A linguagem para modificação de esquemas, denominada $\mathsf{L2}$ não foi tratada neste capítulo, mas está definida também em (MOREIRA et al., 2003b). Cabe salientar que a formalização completa da linguagem TVL/SE não é uma tarefa extremamente complexa, porém a semântica formal especificada para o núcleo da linguagem é suficiente, uma vez que as restrições fundamentais relativas aos conceitos de tempo e versão foram tratadas.

(Create)

$$\frac{\mathsf{SCH} = (\mathsf{SV}, sv_{\mathrm{fst}}, sv_{\mathrm{lst}}) \qquad sv \notin dom(\mathsf{SV})}{t, \mathsf{SCH}, \mathsf{EE}, \texttt{CREATE}\ schema\ sv \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}', EE}$$

$$\begin{array}{l} \mathsf{SV}' = \mathsf{SV}[sv \mapsto (schema, \_, \_, \mathtt{W}, \_, \{\})] \\ \mathsf{SCH}' = (\mathsf{SV}', sv_{\mathrm{fst}}, sv) \end{array}$$

(Derive)

$$\frac{\begin{array}{l}\mathsf{SCH} = (\mathsf{SV}, sv_{\mathrm{fst}}, sv_{\mathrm{lst}}) \\ sv_1 \notin dom(\mathsf{SV}) \qquad\qquad \mathsf{SV}(sv_2) = (schema, tti, time, \mathtt{st}, sv, \mathtt{Succ})\end{array}}{t, \mathsf{SCH}, \mathsf{EE}, \texttt{DERIVE}\ sv_1\ \texttt{FROM}\ sv_2 \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}', \mathsf{EE}}$$

$$\begin{array}{l} \mathsf{SV}' = \mathsf{SV}[sv_1 \mapsto (schema, \_, \_, \mathtt{W}, sv_2, \{\})] \\ \qquad [sv_2 \mapsto (schema, tti, time, \mathtt{st}, sv, \mathtt{Succ} \cup \{sv_1\})] \\ \mathsf{SCH}' = (\mathsf{SV}', sv_{\mathrm{fst}}, sv_1) \end{array}$$

(Promote)

$$\frac{\mathsf{SCH} = (\mathsf{SV}, \overline{sv}) \qquad \mathsf{SV}(sv) = (schema, \_, \_, \mathtt{W}, sv_{\mathrm{prd}}, \{\ \})}{t, \mathsf{SCH}, \mathsf{EE}, \texttt{PROMOTE}\ sv \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}', \mathsf{EE}}$$

$$\begin{array}{l} \mathsf{SV}' = \mathsf{SV}[sv \mapsto (schema, t, \_, \mathtt{S}, sv_{\mathrm{prd}}, \{\ \})] \\ \mathsf{SCH}' = (\mathsf{SV}', \overline{sv}) \end{array}$$

(Block)

$$\frac{\mathsf{SCH} = (\mathsf{SV}, \overline{sv}) \qquad \mathsf{SV}(sv) = (schema, tti, \_, \mathtt{A}, sv_{\mathrm{prd}}, \mathtt{Succ})}{t, \mathsf{SCH}, \mathsf{EE}, \texttt{BLOCK}\ sv \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}', \mathsf{EE}}$$

$$\begin{array}{l} \mathsf{SV}' = \mathsf{SV}[sv \mapsto (schema, tti, \_, \mathtt{B}, sv_{\mathrm{prd}}, \mathtt{Succ})] \\ \mathsf{SCH}' = (\mathsf{SV}', \overline{sv}) \end{array}$$

(Freeze1)

$$\frac{\mathsf{SCH} = (\mathsf{SV}, \overline{sv}) \qquad \mathsf{SV}(sv) = (schema, tti, \_, \mathtt{S}, sv_{\mathrm{prd}}, \mathtt{Succ})}{t, \mathsf{SCH}, \mathsf{EE}, \texttt{FREEZE}\ sv \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}', \mathsf{EE}}$$

$$\begin{array}{l} \mathsf{SV}' = \mathsf{SV}[sv \mapsto (schema, tti, t_{+1}, \mathtt{F}, sv_{\mathrm{prd}}, \mathtt{Succ})] \\ \mathsf{SCH}' = (\mathsf{SV}', \overline{sv}) \end{array}$$

(Freeze2)

$$\frac{\begin{array}{l}\mathsf{SCH} = (\mathsf{SV}, \overline{sv}) \\ \mathsf{SV}(sv) = (schema, tti, \_, \mathtt{st}, sv_{\mathrm{prd}}, \mathtt{Succ})\ \ \mathtt{st} = \mathtt{A} \vee \mathtt{st} = \mathtt{C} \\ tempExt = \{C \mapsto ext \in \mathsf{EE}(sv) \mid TyCls(C, schema) = \mathtt{HasVersions}\} \\ normExt = \{C \mapsto ext \in \mathsf{EE}(sv) \mid TyCls(C, schema) = \mathtt{Normal}\}\end{array}}{t, \mathsf{SCH}, \mathsf{EE}, \texttt{FREEZE}\ sv \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}', \mathsf{EE}'}$$

$$\begin{array}{l} \mathsf{SV}' = \mathsf{SV}[sv \mapsto (schema, tti, t_{+1}, \mathtt{F}, sv_{\mathrm{prd}}, \mathtt{Succ})] \\ \mathsf{SCH}' = (\mathsf{SV}', \overline{sv}) \\ \mathsf{EE}' = \mathsf{EE}[sv \mapsto normExt\ \ cup\ Finalize(tempExt)] \end{array}$$

(Freeze3)

$$\frac{\begin{array}{l}\mathsf{SCH} = (\mathsf{SV}, \overline{sv}) \\ \mathsf{SV}(sv) = (schema, tti, \_, \mathtt{st}, sv_{\mathrm{prd}}, \{sv_i \mid i = 1..n\}) \\ t, \mathsf{SCH}, \mathsf{EE}, \texttt{FREEZE}\ sv_1\ \texttt{CASCADE} \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}_1, \mathsf{EE}_1 \\ \ldots \\ t_{+n-1}, \mathsf{SCH}_{n-1}, \mathsf{EE}_{n-1}, \texttt{FREEZE}\ sv_n\ \texttt{CASCADE} \xrightarrow[\mathsf{L1}]{} t_{+n}, \mathsf{SCH}_n, \mathsf{EE}_n \\ t_{+n}, \mathsf{SCH}_n, \mathsf{EE}_n, \texttt{FREEZE}\ sv \xrightarrow[\mathsf{L1}]{} t_{+n+1}, \mathsf{SCH}', \mathsf{EE}'\end{array}}{t, \mathsf{SCH}, \mathsf{EE}, \texttt{FREEZE}\ sv\ \texttt{CASCADE} \xrightarrow[\mathsf{L1}]{} t_{+n+1}, \mathsf{SCH}', \mathsf{EE}'}$$

Figura 7.3: Regras para a Linguagem L1 (I)

(Restore1)

$$\frac{\mathsf{SCH} = (\mathsf{SV}, \overline{sv}) \qquad \mathsf{SV}(sv) = (schema, tti, tvf, \mathsf{F}, sv_{\text{prd}}, \mathtt{Succ}) \qquad \mathsf{EE}(sv) = \{\,\}}{t, \mathsf{SCH}, \mathsf{EE}, \mathtt{RESTORE}\ sv \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}', \mathsf{EE}}$$

$$\begin{array}{l} \mathsf{SV}' = \mathsf{SV}[sv \mapsto (schema, tti, \_, \mathsf{S}, sv_{\text{prd}}, \mathtt{Succ})] \\ \mathsf{SCH}' = (\mathsf{SV}', \overline{sv}) \end{array}$$

(Restore2)

$$\frac{\begin{array}{l} \mathsf{SCH} = (\mathsf{SV}, \overline{sv}) \\ \mathsf{SV}(sv) = (schema, tti, tvf, \mathsf{F}, sv_{\text{prd}}, \mathtt{Succ}) \quad \mathtt{st} = \mathsf{A} \vee \mathtt{st} = \mathsf{C} \\ tempExt = \{C \mapsto ext \in \mathsf{EE}(sv) \mid TyCls(C, schema) = \mathtt{HasVersions}\} \\ normExt = \{C \mapsto ext \in \mathsf{EE}(sv) \mid TyCls(C, schema) = \mathtt{Normal}\} \\ tempExt \neq \{\,\} \end{array}}{t, \mathsf{SCH}, \mathsf{EE}, \mathtt{RESTORE}\ sv \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}', \mathsf{EE}'}$$

$$\begin{array}{l} \mathsf{SV}' = \mathsf{SV}[sv \mapsto (schema, tti, \_, \mathsf{C}, sv_{\text{prd}}, \mathtt{Succ})] \\ \mathsf{SCH}' = (\mathsf{SV}', \overline{sv}) \\ \mathsf{EE}' = \mathsf{EE}[sv \mapsto normExt \cup Restore(tempExt)] \end{array}$$

(Restore3)

$$\frac{Root(\mathsf{SCH}) = sv \qquad t, \mathsf{SCH}, \mathsf{EE}, \mathtt{RESTORE}\ sv \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}', \mathsf{EE}'}{t, \mathsf{SCH}, \mathsf{EE}, \mathtt{RESTORE}\ sv\ \mathtt{TOP} \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}', \mathsf{EE}'}$$

(Restore4)

$$\frac{\begin{array}{l} Root(\mathsf{SCH}) \neq sv \\ sv_{\text{prd}} = Pred(\mathsf{SV}(sv)) \\ t, \mathsf{SCH}, \mathsf{EE}, \mathtt{RESTORE}\ sv \xrightarrow[\mathsf{L1}]{} t_{+1}, \mathsf{SCH}'', \mathsf{EE}'' \\ t_{+1}, \mathsf{SCH}'', \mathsf{EE}'', \mathtt{RESTORE}\ sv_{\text{prd}}\ \mathtt{TOP} \xrightarrow[\mathsf{L1}]{} t', \mathsf{SCH}', \mathsf{EE}' \end{array}}{t, \mathsf{SCH}, \mathsf{EE}, \mathtt{RESTORE}\ sv\ \mathtt{TOP} \xrightarrow[\mathsf{L1}]{} t', \mathsf{SCH}', \mathsf{EE}'}$$

(Activate1)

$$\frac{\begin{array}{l} \mathsf{SCH} = (\mathsf{SV}, \overline{sv}) \\ \mathsf{SV}(sv) = (schema, tti, \_, \mathsf{S}, sv_{\text{prd}}, \mathtt{Succ}) \\ Status(\mathsf{SCH}, sv_{\text{prd}}) \in \{\mathsf{A}, \mathsf{C}\} \\ ext_{\text{prd}} = \mathsf{EE}(sv_{\text{prd}}) \end{array}}{t, \mathsf{SCH}, \mathsf{EE}, \mathtt{ACTIVATE}\ sv \xrightarrow[\mathsf{L1}]{} t', \mathsf{SCH}', \mathsf{EE}'}$$

$$\begin{array}{l} \mathsf{SV}' = \mathsf{SV}[sv \mapsto (schema, tti, \_, \mathsf{A}, sv_{\text{prd}}, \mathtt{Succ})] \\ \mathsf{SCH}' = (\mathsf{SV}', \overline{sv}) \\ \mathsf{EE}' = \mathsf{EE}[sv \mapsto Convert(ext_{\text{prd}})] \end{array}$$

(Activate2)

$$\frac{\begin{array}{l} \mathsf{SCH} = (\mathsf{SV}, \overline{sv}) \\ \mathsf{SV}(sv) = (schema, tti, \_, \mathsf{S}, sv_{\text{prd}}, \mathtt{Succ}) \\ Status(\mathsf{SCH}, sv_{\text{prd}}) = \mathsf{S} \end{array}}{t, \mathsf{SCH}, \mathsf{EE}, \mathtt{ACTIVATE}\ sv \xrightarrow[\mathsf{L1}]{} t', \mathsf{SCH}', \mathsf{EE}}$$

$$\begin{array}{l} \mathsf{SV}' = \mathsf{SV}[sv \mapsto (schema, tti, \_, \mathsf{A}, sv_{\text{prd}}, \mathtt{Succ})] \\ \mathsf{SCH}' = (\mathsf{SV}', \overline{sv}) \end{array}$$

Figura 7.4: Regras para a Linguagem L1 (II)

(Insert1)

$$\frac{\begin{array}{ll}\mathsf{SCH} \vdash t_{+(i-1)}, \mathsf{EE}_{i-1}, e_n \xrightarrow[\mathsf{e}]{} t_{+i}, \mathsf{EE}_i, v_i & (C, ext) = \mathsf{EE}_{i+k}(sv) \\ \mathsf{SCH} \vdash t_{+(i+k-1)}, \mathsf{EE}_{i+k-1}, f_k \xrightarrow[\mathsf{e}]{} t_{+(i+k+1)}, \mathsf{EE}_{i+k}, v_{i+k} & oid = \mathit{fresh}(\mathsf{EE}_{i+k})\end{array}}{\mathsf{SCH} \vdash t, \mathsf{EE}, \texttt{INSERT}\ C(a_i = e_i\ \mathit{intv}_i{}^{\,i=1..n}, b_k = f_k{}^{\,k=1..m})\ \mathit{intv}\ sv \xrightarrow[\mathsf{L3}]{} t_{+m}, \mathsf{EE}', oid}$$

$$\begin{array}{l} ext' = ext[oid \mapsto (intv, \{a_i \mapsto \{(v_i, ?, ?, intv_i)\}, b_k \mapsto v_{i+k}\})] \\ \mathsf{EE}' = \mathsf{EE}_{i+k}[(sv, C) \mapsto ext'] \end{array}$$

(Insert2)

$$\frac{\mathsf{SCH} \vdash t_{+(i-1)}, \mathsf{EE}_{i-1}, e_i \xrightarrow[\mathsf{e}]{} t_{+i}, \mathsf{EE}_i, v_i \qquad \begin{array}{ll}(C, ext) = \mathsf{EE}_i(sv) \\ oid = \mathit{fresh}(\mathsf{EE}_i)\end{array}}{\mathsf{SCH} \vdash t, \mathsf{EE}, \texttt{INSERT INTO}\ C(a_i = e_i{}^{\,i=1..n})\ sv \xrightarrow[\mathsf{L3}]{} t_{+i+1}, \mathsf{EE}', oid}$$

$$\begin{array}{l} ext' = ext[oid \mapsto \{a_i \mapsto v_i\}] \\ EE' = \mathsf{EE}_{i+1}[(sv, C) \mapsto ext'] \end{array}$$

(Update1)

$$\frac{\begin{array}{ll}\mathsf{SCH} \vdash t, \mathsf{EE}, e_2 \xrightarrow[\mathsf{e}]{} t''', \{oid_1 \ldots oid_n\} & (C, ext) = \mathsf{EE}''(sv) \\ \mathsf{SCH} \vdash t''', \mathsf{EE}, e_1 \xrightarrow[\mathsf{e}]{} t'', \mathsf{EE}'', v & (intv_i, attrs_i) = ext(oid_i) \\ & his_i^a = attrs_i(a)\end{array}}{\mathsf{SCH} \vdash t, \mathsf{EE}, \texttt{UPDATE}\ C\ \texttt{SET}\ a = e_1\ intv_a\ \texttt{WHERE}\ e_2 \xrightarrow[\mathsf{L3}]{} t', \mathsf{EE}'}$$

$$\begin{array}{l} attrs_i' = attrs_i[a_i \mapsto his_i^a \cup \{v, t', t'_{+1}, intv_a\}] \\ ext' = ext[oid_i \mapsto (intv_i, attrs_i')] \\ \mathsf{EE}' = \mathsf{EE}[(sv, C) \mapsto ext'] \end{array}$$

(Update2)

$$\frac{\begin{array}{ll}\mathsf{SCH} \vdash t, \mathsf{EE}, e_2 \xrightarrow[\mathsf{e}]{} t'', \mathsf{EE}''', \{oid_1 \ldots oid_n\} & (C, ext) = \mathsf{EE}''(sv) \\ \mathsf{SCH} \vdash t'', \mathsf{EE}''', e_1 \xrightarrow[\mathsf{e}]{} t', \mathsf{EE}'', v & obj_i = ext(oid_i)\end{array}}{\mathsf{SCH} \vdash t, \mathsf{EE}, \texttt{UPDATE}\ C\ \texttt{SET}\ a = e_1\ \texttt{WHERE}\ e_2 \xrightarrow[\mathsf{L3}]{} T', \mathsf{EE}'}$$

$$\begin{array}{l} ext' = ext[oid_i \mapsto obj_i[a_i \mapsto v]] \\ \mathsf{EE}' = \mathsf{EE}_{i+1}[(sv, C) \mapsto ext'] \end{array}$$

(Delete)

$$\frac{\mathsf{SCH} \vdash t, \mathsf{EE}, \texttt{SELECT FROM}\ e_1\ \texttt{WHERE}\ e_2 \xrightarrow[\mathsf{e}]{} t'', \{oid_1 \ldots oid_n\}}{\mathsf{SCH} \vdash t, \mathsf{EE}, \texttt{DELETE FROM}\ e_1\ \texttt{WHERE}\ e_2\ t_f \xrightarrow[\mathsf{L3}]{} t', \mathsf{SCH}, \mathsf{EE}'}$$

$$\mathsf{EE} = \mathit{Finalize}(t_f, \{oid_1 \ldots oid_n\})$$

Figura 7.5: Regras para a Linguagem L3

8 AMBIENTE TEMPORAL DE EVOLUÇÃO DE ESQUEMAS

Este capítulo apresenta um ambiente que presta suporte ao desenvolvimento de aplicações com gerência de versionamento temporal de esquemas através do TVSE. O ambiente é denominado Ambiente Temporal de Evolução de Esquemas. Um conjunto das funcionalidades do TVSE foi implementado neste ambiente, utilizando um banco de dados convencional. O ambiente incorpora a Ferramenta de Especificação de Classes (PEIXOTO et al., 2002) implementada para o TVM. O ambiente está sendo implementado no âmbito do projeto Tempo-Versões (MORO et al., 2003).

Dois diferentes usuários são identificados no ambiente: o administrador de banco de dados e o usuário de banco de dados. O administrador utiliza o ambiente como apoio para a construção da especificação das aplicações. O usuário pode especificar a aplicação com todos os conceitos definidos no TVM e no TVSE. As informações a respeito do histórico da evolução dos esquemas são armazenadas em uma estrutura de metadados. O próprio ambiente guia a especificação das aplicações, mantendo todo o histórico da evolução do esquema através de versionamento temporal. O usuário do banco de dados pode manipular um ou mais esquemas específicos para popular o banco de dados, realizando atualizações e consultas. O ambiente agrega funcionalidade para a coexistência de dados temporais versionados e convencionais.

8.1 Visão Geral

A especificação de uma arquitetura de SGBD integrada para um modelo de dados com suporte para tempo e para versões consiste em uma tarefa de alto custo, que somente os maiores fabricantes de banco de dados poderiam financiar (JENSEN, 1999). O fato de já existir um SGBD que gerencie grande volume de dados sugere uma alternativa mais adequada: prover suporte para aplicações de tempo e de versões embutido por meio de uma camada entre um SGBD existente e as aplicações dos usuários.

Seguindo essa idéia, foi proposto um Ambiente Temporal de Evolução de Esquemas para implementar o TVSE de forma integrada com o ambiente de Especificação de Classes proposto para o TVM. O ambiente dispõe de um conjunto de interfaces para a especificação de esquemas, gerenciamento de versões e manipulação de dados. O gerenciamento da evolução temporal de esquemas e dados é modelado através de uma camada intermediária que controla as aplicações dos usuários e o banco de dados propriamente dito. Conforme ilustrado na Figura 8.1, os esquemas e suas respectivas bases de dados são armazenados em um banco de dados (Banco de Dados Intenção e Extensão). Paralelamente, é especificada uma estrutura de metadados, cujo objetivo é

manter informações a respeito da evolução ocorrida.

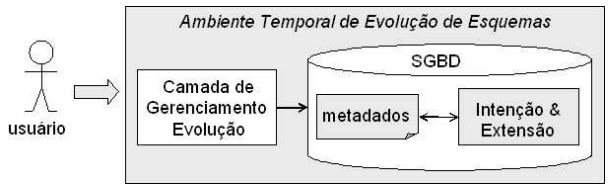


Figura 8.1: Ambiente Temporal de Evolução de Esquemas

O gerenciamento de evolução pode ser dividido em três partes distintas: gerenciamento da Intenção, gerenciamento da Extensão, e acesso e manipulação aos dados, através da camada de gerenciamento da evolução, conforme ilustrado pela Figura 8.2. O gerenciamento da evolução dos esquemas é realizado no nível da intenção, sendo que cada esquema possui seus dados associados. O gerenciamento da extensão é responsável pela evolução histórica dos dados em cada esquema. O gerenciamento da evolução permite o acesso aos dados considerando uma versão de esquema em particular, ou um esquema em evolução (com todas as suas versões), como, por exemplo, um Sistema Legado. Cabe salientar que os dados da extensão podem estar armazenados em um repositório único ou em múltiplos repositórios, dependendo da estratégia de implementação adotada (vide seção 8.2).

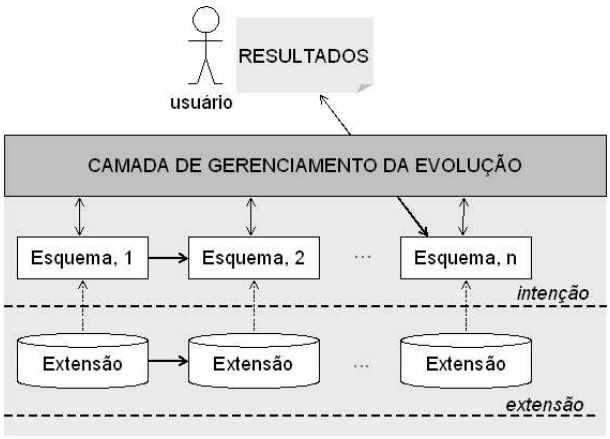


Figura 8.2: Gerenciamento da Evolução de Esquemas

8.1.1 Arquitetura

Detalhando essa idéia, a Figura 8.3 ilustra como é a interação da interface do ambiente com a camada e com o banco de dados suporte. Os componentes listados na parte superior da figura: especificação de esquemas, versionamento, evolução de esquemas, gerenciamento de transações e gerenciamento de consultas, representam o Ambiente de Versionamento Temporal de Esquemas. O segundo nível da figura apresenta os componentes que devem ser fornecidos pelo SGBD (processador de consulta e a estrutura de metadados) e os inter-relacionamentos de tais módulos com o processo de evolução de esquemas e atualização de objetos. O processador de consultas é utilizado para acessar a estrutura de metadados e manipular a base de dados extensional durante a execução das aplicações. A linguagem TVQL é utilizada na recuperação dos objetos convencionais e

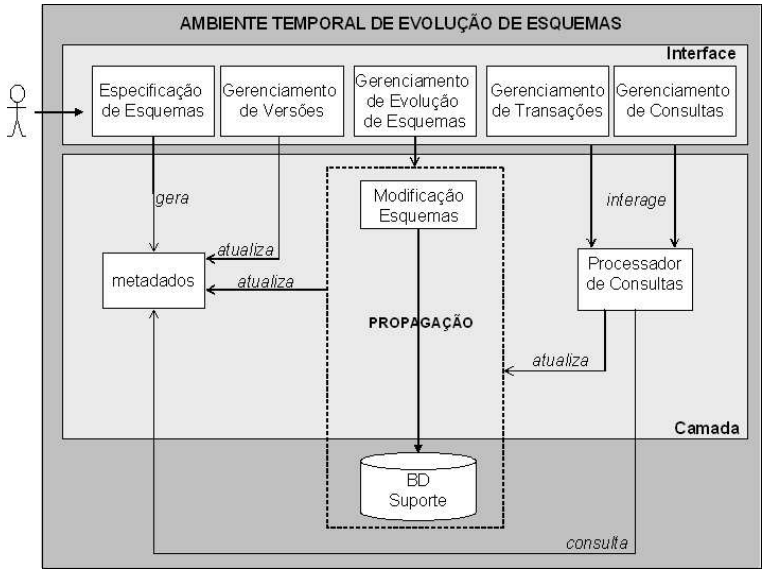


Figura 8.3: Ambiente Temporal para Evolução de Esquemas

temporais versionados. Do ponto de vista do usuário, a camada encapsula o SGBD, que realiza as funções na base de dados extensional.

O Ambiente Temporal de Evolução de Esquemas proporciona ao usuário as funcionalidades básicas de um SGBD, divididas nos seguintes módulos:

- Especificação de esquemas - permite a especificação de esquemas que servirão de base para o processo de evolução, gerando os metadados necessários para armazenar as informações a respeito da evolução dos esquemas. O esquema pode ser especificado utilizando todas as funcionalidades propostas para especificação de classes propostas pelo TVM;
- Gerenciador de Versionamento - permite a especificação de um esquema, bem como a realização de operações de transição entre os estados permitidos às versões, por exemplo: derivar, excluir, ativar, bloquear, congelar;
- Gerenciador de Evolução de Esquemas - permite a realização de operações de modificação de esquemas, realizando a propagação das mudanças nos dados associados. O gerenciamento das versões de esquema é feito implicitamente pelo sistema;
- Gerenciador de Transações - permite a atualização dos dados associados às versões de esquema através de operações de inserção, alteração e exclusão;
- Gerenciador de Consultas - permite a execução de consultas aos dados associados às versões de esquema.

Cabe salientar que a Ferramenta de Especificação de Classes do TVM é utilizada na sua plenitude pelo componente Especificação de Esquemas. O processador de consulta é utilizado para acessar a estrutura de metadados e manipular a base de dados extensional durante a execução das aplicações. A linguagem TVQL é utilizada na recuperação dos objetos convencionais e temporais versionados, tanto nas consultas quando durante o gerenciamento de transações.

8.2 Armazenamento Físico da Intenção e Extensão

Esta seção propõe um mecanismo híbrido para o armazenamento físico da base de dados extensional através de duas alternativas de implementação: repositório único e múltiplos repositórios. As duas abordagens podem coexistir no mesmo domínio de aplicação, dependendo da operação de modificação de esquema realizada.

Um artigo apresentando um estudo preliminar sobre a viabilidade de gerenciar a evolução de esquemas conceituais, armazenando a extensão em repositório único e/ou em múltiplos repositórios foi publicado nos anais do *XVI Simpósio Brasileiro de Banco de Dados (SBBD 2001)* (ROMA et al., 2001). Outro artigo especificando detalhadamente o método de armazenamento híbrido (proposto para o TVSE) foi publicado nos anais do *13th International Conference on Database and Expert Systems Applications (DEXA 2002)* (GALANTE et al., 2002).

8.2.1 Proposta de um Mecanismo Híbrido para o Armazenamento da Extensão

Segundo Cristina de Castro (CASTRO; GRANDI; SCALAS, 1995b, 1997), existem duas soluções para o armazenamento de vários esquemas e de seus dados associados: repositório único e múltiplos repositórios, conforme detalhado na seção 2.4.

Uma alternativa de implementação proposta para o TVSE consiste em utilizar as duas abordagens de armazenamento de forma integrada, de acordo com o tipo de alteração de esquema realizado. A solução múltiplos repositórios deve ser empregada nos casos em que a primitiva de atualização modifica a estrutura do esquema. Por exemplo, incluir/excluir uma classe ou um relacionamento. Por outro lado, a solução repositório único deve ser adotada nos casos em que a estrutura de uma classe é alterada, como por exemplo, a inclusão de um atributo.

A Figura 8.4 apresenta a classificação completa das operações que acarretam o armazenamento da extensão em um único repositório ou em múltiplos repositórios. Assim, a abordagem de armazenamento em múltiplos repositórios é adotada na presença de alterações na estrutura do esquema, nos relacionamentos ou quando uma operação composta é definida pelo usuário. A abordagem de armazenamento em repositório único é adotada nos casos em que a estrutura de uma classe ou a definição de um método é alterada. É importante ressaltar que as duas abordagens podem coexistir em um mesmo domínio de aplicação, dependendo da operação de modificação de esquema realizada.

Com o intuito de ilustrar o mecanismo proposto, as subseções a seguir exemplificam o armazenamento da base de dados extensional em repositório único e em múltiplos repositórios, seguindo o mecanismo proposto para o TVSE.

8.2.1.1 Armazenamento em Múltiplos Repositórios

Com o objetivo de armazenar todo histórico das modificações, uma alternativa para a implementação do gerenciamento temporal de esquemas é a utilização de versionamento de esquemas para a intenção e múltiplos repositórios para extensão. Nesse caso, qualquer operação de modificação acarreta a derivação de uma nova versão de esquema e, conseqüentemente, um novo repositório de dados deve ser criado. Quando uma nova versão de esquema é derivada, as instâncias são adaptadas de acordo com o novo esquema e copiadas para o novo repositório.

Tomando como base o exemplo apresentado no capítulo anterior (Figura 6.2), a Figura 8.5 ilustra a inclusão da classe `Vinculo`, formando uma relacionamento de associação entre as classes `Aluno` e `Curso`. Conseqüentemente, uma nova versão de

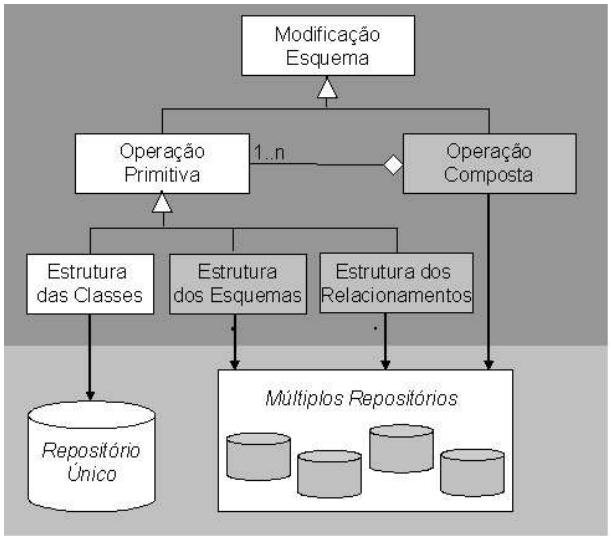


Figura 8.4: Armazenamento híbrido da base de dados extensional

esquema é derivada (`SV3`), repercutindo a modificação realizada na primeira versão do esquema (`SV1`), cujas informações são armazenadas no banco de dados da intenção. Para cada versão de esquema, um novo repositório de dados é criado (`Repositório1` e `Repositório2`), armazenando as modificações realizadas.

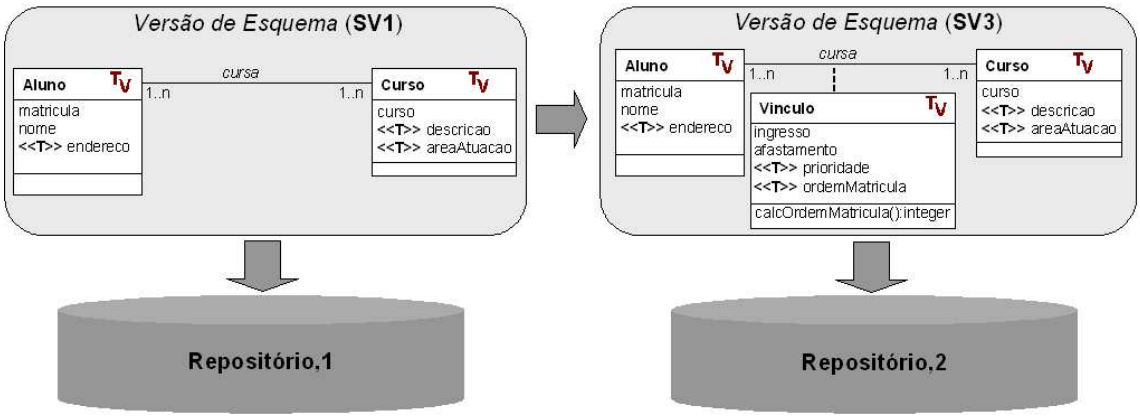


Figura 8.5: Armazenamento da extensão em múltiplos repositórios

Cabe salientar que, no nível de representação física, os vários repositórios podem compartilhar estruturas, sendo evitada a replicação de dados não afetados pela evolução do esquema.

8.2.1.2 Armazenamento em Repositório Único

Como o simples processo de derivação de esquemas e a criação de repositórios de dados pode implicar uma excessiva proliferação de versões, uma outra alternativa para implementação é definida, visando a melhorar o desempenho do sistema, evitando interrupções freqüentes e garantindo a integridade das informações armazenadas.

Para tanto, propõe-se a derivação de versões de esquema para a intenção nos casos em que a operação de modificação não altera a estrutura do esquema, acarretando a atualização do mesmo repositório para o armazenamento da extensão, ou seja, adotando

a abordagem repositório único. Nesse caso, todo o histórico da evolução é igualmente mantido.

A Figura 8.6 ilustra a operação de inclusão do atributo `email`, na classe `Aluno`, a partir da primeira versão do esquema (`SV1`), e, por conseguinte, a derivação de uma nova versão do esquema (`SV2`), sendo a alteração repercutida no banco de dados da extensão. Entretanto, a modificação realizada é propagada no repositório de dados existente (Repositório,1).

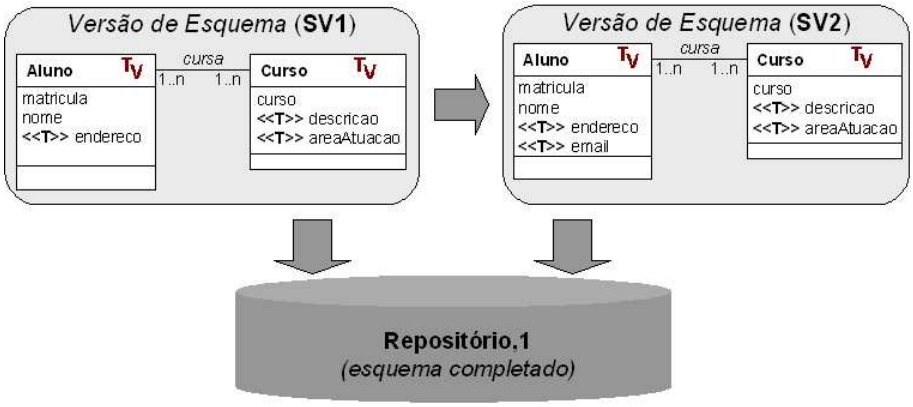


Figura 8.6: Armazenamento da extensão em repositório único

8.3 Ferramenta de Apoio ao Gerenciamento de Evolução de Esquemas

Esta seção apresenta em linhas gerais o protótipo implementado para o TVSE. O seu desenvolvimento levou em consideração as características do TVSE propostas nos capítulos anteriores, buscando comprovar que a solução indicada é possível de ser implementada, e gerando os resultados esperados quanto à integração do versionamento de esquemas com bancos de dados temporais, implementados sobre um banco de dados convencional.

Obviamente que esta implementação não prova a validade do TVSE, mas serve de indicativo de que os objetivos propostos podem ser plenamente atingidos. Por exemplo, é possível mostrar a viabilidade de armazenamento do histórico da evolução de um banco de dados tanto no nível intencional quanto extensional. A desvantagem, por conseguinte, é a grande quantidade de informações extras armazenadas, a qual pode ser amenizada através da escolha, pelo usuário, de quais classes e dados serão temporais.

Esta seção apresenta resumidamente o protótipo implementado. A ferramenta é denominada Ambiente Temporal de Evolução de Esquemas. A especificação completa está documentada em (PEIXOTO, 2003), que consiste em um trabalho de diplomação apresentado na UFRGS para obtenção do grau de bacharel em Ciência da Computação. Cabe salientar que o ambiente implementado incorpora a Ferramenta de Especificação de Classes (PEIXOTO et al., 2002) implementada para o TVM.

8.3.1 Arquitetura

A Figura 8.7 apresenta a arquitetura proposta para a Ferramenta de Evolução de Esquemas, sendo composta dos seguintes módulos: Gerente de Especificação de Esquemas, Gerente de Versionamento e Gerente de Transações.

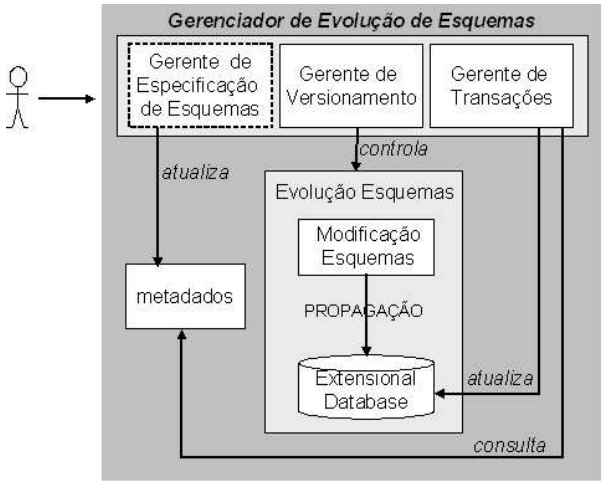


Figura 8.7: Arquitetura da Ferramenta de Suporte à Evolução Temporal de Esquemas

A implementação utilizou como banco de dados suporte o IBM DB2 *Universal Database*. Os fatores que levaram à escolha do DB2 como banco de dados foram:

- proximidade com tipos de dados do SQL-92;
- o fato de já existir um mapeamento do TVM (MORO, 2001) para esse banco de dados, bem como da linguagem de consulta TVQL (ZAUPA, 2002); e
- a facilidade de utilização de tipos de dados temporais fornecidos pelo SGBD.

É importante lembrar que a implementação dessa ferramenta independe da linguagem ou do banco de dados, sendo necessário somente que eles tenham os recursos que permitam o desenvolvimento desse tipo de aplicação.

As subseções seguintes apresentam sucintamente as interfaces e as principais especificações dos diversos módulos que compõem a ferramenta.

8.3.1.1 Painel de Controle

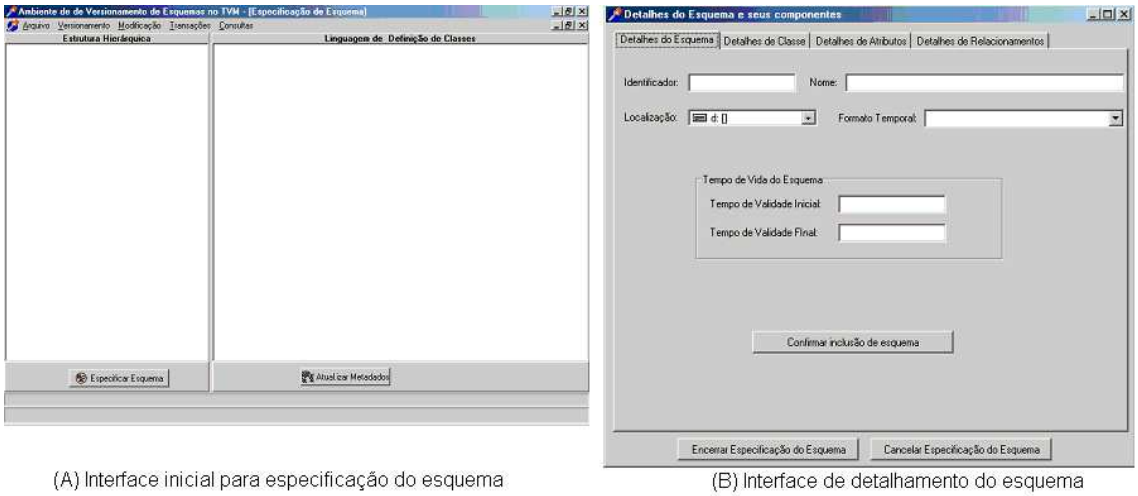
A Ferramenta possui uma interface de controle através da qual o usuário pode obter acesso aos diversos módulos que compõem o sistema: Versionamento de Esquemas, Modificação de Esquemas, Transações e Consultas, conforme detalhado nas seções subseqüentes.

8.3.1.1.1 Gerente de Versionamento

O Gerente de Versionamento tem por objetivo permitir a especificação de um esquema e o gerenciamento das operações de transição de estados permitidas às versões de esquema. Além disso, realiza o gerenciamento dos metadados (informações sobre os esquemas, classes, atributos e relacionamentos), provendo os recursos necessários aos outros módulos quanto às informações sobre as versões do esquema conceitual. O Gerenciador de Versões possui os seguintes componentes:

- Especificar - permite criar o primeiro esquema que servirá de base para o processo de derivação de versões;
- Derivar - permite derivar uma versão de esquema a partir de outra existente;
- Promover - permite promover uma versão em trabalho para estável;

- Ativar - permite ativar uma versão de esquema, resultando na propagação das modificações no dados;
- Congelar - permite excluir logicamente uma versão de esquema e seus dados associados.



(A) Interface inicial para especificação do esquema

(B) Interface de detalhamento do esquema

Figura 8.8: Interface do Gerente de Versionamento

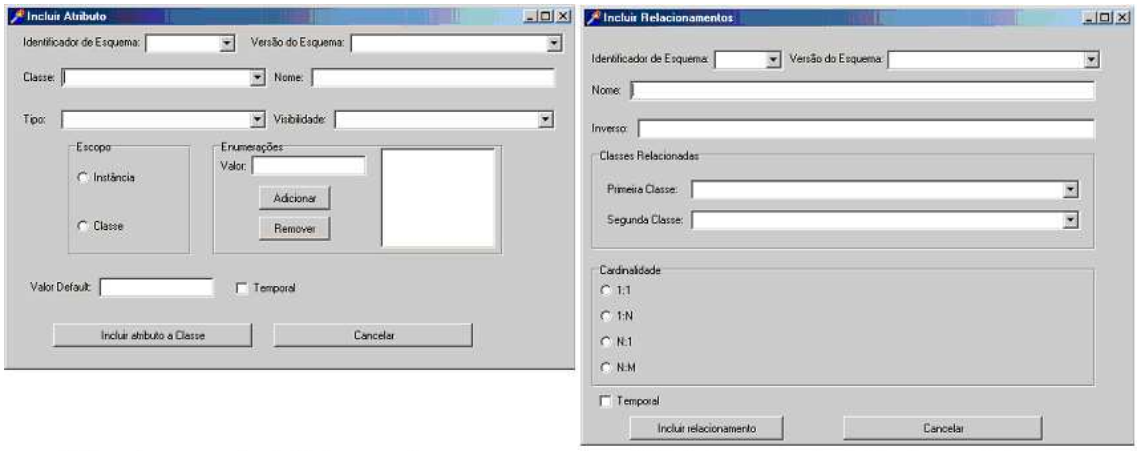
A Figura 8.8-(a) ilustra a interface para especificação de um esquema. A Figura 8.8-(b) mostra a interface para o detalhamento do esquema, no qual o usuário pode especificar as classes, atributos e relacionamentos. Todas as funcionalidades da Ferramenta de Especificação de Classes do TVM são mantidas, ou seja, os elementos do esquema podem ser temporais ou não temporais. Além disso, as operações de transição de estados são também permitidas, como, por exemplo, derivar, ativar e congelar, bastando somente que o usuário selecione previamente a versão do esquema na qual deseja aplicar a operação.

8.3.1.1.2 Gerente de Modificação

Através do Gerente de Modificação o usuário pode realizar modificações nas versões de esquemas. Realizar modificação em uma versão de esquema que esteja em um estado evolutivo diferente do estado em trabalho ocasiona a derivação de uma nova versão de esquema. Se a versão estiver no estado em trabalho, a utilização de uma operação de modificação acarreta uma correção no esquema para qual o histórico não é armazenado, uma vez que a versão não possui tempos associados.

O Gerente de Modificação apresenta três opções para modificação: atributo, classe e relacionamento. As operações de modificação são separadas em três grupos: inclusão, exclusão e alteração. Como cada operação de modificação traz consigo algumas peculiaridades, foi definida uma interface para cada operação, contendo os requisitos necessários para o cumprimento das funcionalidades inerentes ao TVSE. Para simplificar a apresentação do texto, e não o tornar exaustivo, são exemplificadas duas operações do Gerente de Modificação. A Figura 8.9-(A) ilustra a interface para inclusão de um atributo. A inclusão de um atributo implica a definição do nome do atributo, a classe a que pertence, o tipo, seu escopo, sua visibilidade e a lista de valores possíveis. A Figura 8.9-(B) ilustra a interface para a inclusão de um relacionamento. Incluir um relacionamento implica a definição de um nome para o relacionamento e seu relacionamento inverso, caso exista,

na definição da cardinalidade, da classe a qual o relacionamento pertence e da classe com a qual está relacionado, além da definição da temporalidade do relacionamento.



(A) Interface para a inclusão de um atributo (B) Interface para a inclusão de um relacionamento

Figura 8.9: Interface do Gerente de Modificação

8.3.1.1.3 Gerente de Transações

O Gerente de Transações é o módulo responsável por realizar instanciações em versões de esquemas ativas. Através do gerenciador é possível incluir, alterar ou excluir instâncias de um objeto existente na base de dados. As operações de atualização são realizadas selecionando uma versão específica de esquema, identificadas pelo nome da versão. No momento da instanciação, as operações de atualização da base se comportam como operações de atualização comuns em banco de dados para atributos sem tempo e versão. Para os atributos temporais, o gerenciamento dos rótulos de tempo de transação e tempo de validade são providos pela ferramenta. O tempo de transação é gerenciado pela ferramenta, sendo que o tempo de validade pode ser fornecido pelo usuário.

A Figura 8.10 apresenta a interface da ferramenta responsável pelas operações do gerente de transações. Para realizar a operação, o usuário deve selecionar o esquema, a classe a ser instanciada e a operação a ser realizada (inserção, exclusão ou alteração). Após a seleção da versão do esquema e da classe a ser alterada, são exibidas as informações disponíveis.

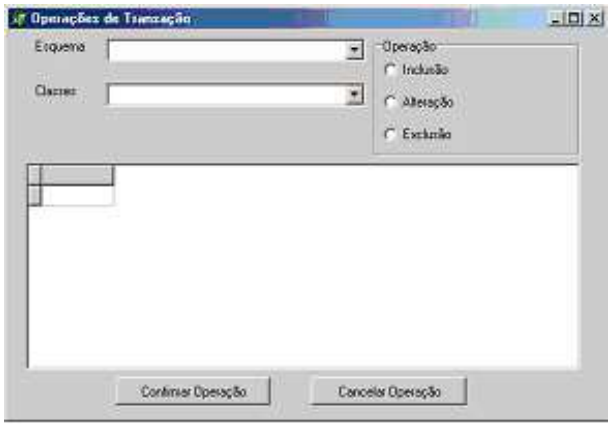


Figura 8.10: Interface do Gerente de Transações

8.4 Considerações Finais

Este capítulo apresentou uma arquitetura para desenvolver o funcionamento do TVSE sobre um SGBD convencional. Um característica ímpar do ambiente é a completa integração com o gerenciamento de evolução dos dados, tendo como base o TVM. Uma proposta para a implementação de um mecanismo híbrido de armazenamento da extensão do banco de dados foi também apresentada. Finalizando o capítulo, foi apresentada uma ferramenta para o suporte à evolução temporal de esquemas. Nessa interface, o usuário pode gerenciar o processo de modificação de esquemas e propagação de mudanças, bem como realizar consultas e atualizações na base de dados extensional.

A implementação do protótipo contribuiu significativamente para mostrar a viabilidade de gerenciamento de versões tanto no nível intencional quanto no extensional do banco de dados, mantendo todo o histórico da evolução de um banco de dados. De forma prática, foram implementados os conceitos que envolvem estas duas áreas de pesquisa, comprovando assim os resultados obtidos. Por ser um trabalho bastante amplo, diversas limitações foram identificadas. Por exemplo, a limitação do armazenamento em múltiplos repositórios e aplicação de uma primitiva de atualização por vez, ou ainda a implementação somente das construções mais usuais dos comandos de manipulação de dados. O ideal seria reduzir o foco, tratando independentemente cada módulo implementado, de forma que pudessem ser explorados na sua totalidade.

Pretende-se estender as facilidades da ferramenta, incorporando uma linguagem de consulta completa que envolva consultas com respeito à evolução da intenção, à evolução da extensão, bem como consultas envolvendo versões de esquemas e seus dados associados, tratando homogeneamente os conceitos de tempo e de versão. Outras sugestões são gerar especificações em XML, para o processo de integração entre as versões de esquemas.

9 CONCLUSÕES

Esta tese apresenta um modelo para evolução dinâmica de esquemas em bancos de dados temporais orientados a objetos denominado TVSE. TVSE foi proposto como forma de guiar modificações em esquemas, classes, métodos e objetos com a finalidade de manter todo o histórico das alterações realizadas, tendo como base o Modelo Temporal de Versões definido em (MORO, 2001). O gerenciamento de evolução de esquemas é um tópico atual de pesquisa na comunidade de bancos de dados temporais. Esse fato justifica a relevância do tema da tese.

Este capítulo apresenta as conclusões e alguns comparativos com relação ao modelo proposto nesta tese. Esses aspectos têm por objetivo apresentar uma revisão da tese, colocando em evidência as principais contribuições e os aspectos inovadores, bem como identificar os desdobramentos mais significativos para trabalhos futuros.

9.1 Contribuições

O TVSE propõe uma solução para evolução dinâmica de esquemas, contemplando também o processamento de transações e o armazenamento físico das versões de esquemas e seus dados associados. A principal contribuição é a definição de estratégias que regem o processo de evolução de esquemas em todos os seus aspectos: modificação de esquemas, propagação de mudanças e manipulação de dados. Além disso, o TVSE representa um diferencial em relação às abordagens de evolução de esquemas existentes, pois incorpora homogeneamente os conceitos de versão e de tempo tanto no nível de esquemas quanto no nível de objetos, permitindo a representação completa do histórico da evolução de esquemas e de instâncias. A união dos conceitos de tempo e de versão é utilizada no tratamento uniforme de esquemas e objetos, permitindo armazenar todo o histórico da evolução ao longo do tempo.

O TVM foi selecionado por apresentar características peculiares que auxiliam o processo de evolução de esquemas. Entre as vantagens, destacam-se as características de relacionamento de herança por extensão, ordem temporal ramificada e a possibilidade da modelagem de classes sem tempo e versão entre as classes temporais versionadas.

A utilização dos conceitos de tempo e de versão no tratamento de evolução de esquemas permite o desenvolvimento de esquemas conceituais onde refinamentos são acrescentados gradualmente ao esquema sem causar danos às aplicações em execução, sendo de fundamental importância no desenvolvimento de projetos de bancos de dados. A utilização de versões para esquemas e objetos permite manter o histórico das modificações e garantir a manipulação dos objetos em qualquer perspectiva de versões sob a qual são definidos. A transparência frente às alterações é mantida, pois aplicações antigas podem continuar funcionando e novas aplicações podem conhecer versões antigas de objetos. Por

outro lado, os conceitos de bancos de dados bitemporais, que incorporam tanto o tempo de transação quanto o tempo de validade, provêm flexibilidade ao mecanismo de evolução de esquemas, não somente por permitir o acesso a informações presentes, passadas e futuras, mas também por permitir atualizações e consultas entre as diversas versões de esquemas existentes.

Inicialmente, foi realizado um estudo de caso, considerando o Sistema Discente da UFRGS. Esse estudo possibilitou a identificação de funcionalidades reais e práticas no contexto de evolução de esquemas. A análise da evolução de um sistema em execução teve o intuito de investigar problemas significativos, com base em dimensões reais, permitindo uma melhor aplicabilidade e adequação do modelo proposto. Constatou-se que uma das grandes dificuldades encontradas foi a substituição de um banco de dados (esquemas e aplicativos) por outro, inviabilizando a possibilidade de retornar a estados anteriores às transições.

Dentro desse contexto, foi definido um modelo para evolução dinâmica de esquemas em bancos de dados temporais orientados a objetos, que busca identificar e tratar os diversos aspectos relacionados à evolução de esquemas com tempo e versões. Em particular, o TVSE traz consigo as seguintes contribuições:

- Modificação de Esquemas e Propagação de Mudanças - especifica uma linguagem para versionamento e modificação de esquemas, bem como o mapeamento de tal linguagem para ODMG. Define um mecanismo de propagação de mudanças através da especificação de função de conversão de objetos para cada operação de modificação de esquemas;

- Armazenamento da Extensão e Intenção - avalia as duas principais alternativas de armazenamento da extensão e intenção (repositório único e múltiplos repositórios) tendo como base o TVSE. Além disso, é proposto um método híbrido que contempla características das duas propostas, reduzindo o espaço de armazenamento e a excessiva proliferação de versões;

- Gerenciamento de Dados - apresenta uma linguagem para modificação de dados durante o gerenciamento da evolução de esquemas. O diferencial consiste em prover mecanismo para execução de consultas que envolvem mais de um esquema simultaneamente;

- Semântica para Evolução de Esquemas - define precisamente o modelo através de semântica operacional.

A utilização do TVSE não implica, necessariamente, a utilização de um SGBD específico. Sendo assim, é proposta uma arquitetura de uma camada intermediária para a execução do TVSE sobre um SGBD convencional. Essa camada, denominada Ambiente de Especificação e Evolução de Esquemas, é responsável pelo gerenciamento das características específicas do TVSE. O ambiente projetado agrega a camada intermediária e o SGBD, tornando o processo de evolução de esquemas transparente para o usuário. Cabe salientar que, para que o ambiente funcione adequadamente, a estrutura de metadados (seção 5.1.4) deve estar previamente criada no SGBD. O Ambiente agrega à Ferramenta de Especificação de Classes (PEIXOTO et al., 2002), proposta para o TVM, um mecanismo para gerenciamento de evolução de esquemas. Através de sua interface, o usuário pode especificar esquemas (com classes normais e/ou temporais versionadas), gerenciar o versionamento de esquemas, aplicar modificações de esquemas (primitivas

ou compostas), consultar esquemas e instâncias, e atualizar instâncias considerando as diversas versões de esquemas. Além disso, o usuário convive com o versionamento e evolução de esquemas sem a necessidade do conhecimento prévio de todas as peculiaridades inerentes ao modelo. Por fim, para mostrar a viabilidade do modelo proposto, o ambiente é implementado sobre o banco de dados DB2. O DB2 foi escolhido pela proximidade dos tipos temporais definidos na linguagem SQL2 e pela implementação de tipos estruturados com métodos próprios de acesso e atualização.

Em resumo, o modelo é apresentado com riqueza de detalhes que incluem: mecanismo temporal para versionamento de esquemas, gerenciamento de modificação de esquemas e propagação de mudanças, mapeamento para o padrão ODMG, mecanismo híbrido para armazenamento de esquemas e dados associados, gerenciamento de transações de dados e de esquemas e uma semântica operacional que define precisamente o modelo. Também é apresentado um estudo que compara o modelo proposto com os modelos e sistemas mais significativos presentes na literatura.

Como produção científica estão publicados os seguintes artigos:

1. Renata de Matos Galante, Clesio Saraiva do Santos, Nina Edelweiss. **Temporal and Versioning Model to Schema Evolution in Object-Oriented Databases**. Data Knowledge & Engineering, primeira revisão: setembro 2003 (GALANTE; SANTOS; EDELWEISS, 2003);

2. Renata de Matos Galante, Nina Edelweiss, Clesio Saraiva do Santos, Álvaro Freitas Moreira. **Data Modification Language for Full Support of Temporal Schema Versioning**. In: *XVIII Simpósio Brasileiro de Banco de Dados* - SBBD 2003, Outubro de 2003, Manaus, Amazonas, Brasil (GALANTE et al., 2003);

3. Renata de Matos Galante, Nina Edelweiss, Clesio Saraiva do Santos. **TVL/SE - Temporal and Versioning Language for Schema Evolution in Object-Oriented Databases**. In: *14th International Conference on Database and Expert Systems Applications* - DEXA 2003, September 2003, Prague, Czech Republic, LNCS (GALANTE; EDELWEISS; SANTOS, 2003);

4. Renata de Matos Galante, Adriana B. S. Roma, Anelise Jantsch, Nina Edelweiss, Clesio Saraiva do Santos. **Dynamic Schema Evolution Management using Version in Temporal Object-Oriented Databases**. In: *13th International Conference on Database and Expert Systems Applications* - DEXA 2002, September 2002, Aix en Provence, France, LNCS 2453 (GALANTE et al., 2002);

5. Renata de Matos Galante, Nina Edelweiss, Clesio Saraiva dos Santos. **Change Management for a Temporal Versioned Object-Oriented Database**. In: *Second International Workshop on Evolution and Change in Data Management* - ECDM 2002 (in conjunction with ER 2002), October 2002, Tampere, Finland, LNCS (GALANTE; EDELWEISS; SANTOS, 2002b);

6. Renata de Matos Galante, Nina Edelweiss, Clesio Saraiva dos Santos. **Gerenciamento Dinâmico de Evolução de Esquemas usando o Modelo Temporal de Versões**. In: *I Workshop de Teses e Dissertações em Bancos de Dados* - SBBD 2002, Outubro de 2002, Gramado, RS, Brasil (GALANTE; SANTOS; EDELWEISS, 2002);

7. Renata de Matos Galante, Nina Edelweiss, Clesio Saraiva dos Santos. **Evolução de Esquemas e Propagação de Mudanças usando o Modelo Temporal de Versões**. In: *XXVIII Conferência Latinoamericana de Informática* - CLEI 2002, Novembro de 2002, Montevideo, Uruguay (GALANTE; EDELWEISS; SANTOS, 2002a);

8. Adriana Bueno da Silva Roma, Renata de Matos Galante, Anelise Jantsch, Nina Edelweiss, Clesio Saraiva dos Santos. **Gerenciamento Temporal de Versões para Evolução de Esquemas em BDOO**. In: *XVI Simpósio Brasileiro de Banco de Dados* - SBBD 2001, Outubro de 2001, Rio de Janeiro, RJ, Brasil (ROMA et al., 2001).

Ainda como produção científica o seguinte minicurso foi ministrado:

1. Álvaro Freitas Moreira, Renata de Matos Galante, Nina Edelweiss, Clesio Saraiva do Santos. **Semântica Operacional e Sistemas de Tipos para Linguagens de Banco de Dados**. In: VI Workshop de Métodos Formais, Outubro 2003, Campina Grande, Paraíba, Brasil (MOREIRA et al., 2003a);

Alguns trabalhos foram desenvolvidos em paralelo ao desenvolvimento desta tese, entre eles:

- TVMSE - Uma Implementação do Versionamento de Esquemas segundo o TVM (JANTSCH, 2003) - é um protótipo que simula um ambiente de versionamento de esquemas, no qual a intenção é implementada através de tempo de transação, a extensão é bitemporal (agregando as principais características do TVM) e o gerenciamento é síncrono;
- Implementação de Bancos de Dados Temporais com Versionamento de Esquemas: um estudo comparativo (SANTOS, 2003) - é uma implementação de versionamento de esquemas que avalia duas alternativas para o armazenamento da extensão de dados: repositório único e múltiplos repositórios. Os critérios de avaliação são: o espaço de armazenamento, o tempo para atualização dos dados e o tempo de resposta às consultas;
- Ambiente de Especificação e Evolução de Esquemas no TVM (PEIXOTO, 2003) - é um protótipo do ambiente de especificação, versionamento e modificação de esquemas proposto nesta tese.

Outros trabalhos estão em desenvolvimento tendo como base no TVSE, entre eles:

- uma tese de doutorado propõe um mecanismo de processamento de consulta para o TVSE, considerando o banco de dados intencional e extensional. Entre os objetivos, estão a analise de consultas através de outras versões de esquemas além da versão corrente, bem como consultas que envolvem mais de um esquema, denominadas consultas multi-esquemas. Além disso, objetiva-se especificar um método de otimização, tanto para as operações de evolução de esquemas quanto para as operações de consulta, que reduza a seqüência de operações realizadas, garantindo que o resultado produzido esteja de acordo com o original;
- como extensão desta tese, está sendo definido um sistema de tipos que prova a correção de propriedades do TVSE através de semântica operacional. Por exemplo, provar se a avaliação de uma operação de consulta é determinística ou não;
- uma dissertação de mestrado propõe o mapeamento do TVSE para gerenciar modificação e evolução de esquemas em dados semi-estruturados.

9.1.1 Comparativo com outros Modelos

O capítulo 2 apresenta o contexto no qual a tese está inserida, identificando as principais necessidades e desafios quanto ao gerenciamento de evolução de esquemas com características de tempo e de versão. O atendimento desses critérios por TVSE é ilustrado nas Tabelas 9.1 e 9.2.

Tabela 9.1: Comparativo do modelo proposto com os demais analisados

Itens / Propostas	**STV**	**MTV**	**PMTV**	**ORION**	**COAST**	**DBEvolution**
Modelo de Dados	relacional	relacional	relacional	OO	OO	OO
Dimensão Temporal	TT, TV, bitemporal	TT, TV, bitemporal	bitemporal	×	×	×
Gerenciamento da Intenção	TT, TV, bitemporal	TT, TV, bitemporal	TT, TV, bitemporal	×	×	×
Gerenciamento da Extensão	TT, TV, bitemporal	TT, TV, bitemporal	TT, TV, bitemporal	×	×	×
Armazenamento da Extensão	mono	multi	mono, multi	−−	−−	−−
Sincronismo entre Intenção e Extensão	síncrono, assíncrono	síncrono, assíncrono	síncrono, assíncrono	−−	−−	−−
Granularidade do Versionamento	tabelas	tabelas	tabelas	objetos	esquemas	esquemas, classes, objetos
Método para evolução de esquemas	versões	versões	versões	conversão	versões	versões, conversão
Taxonomia de mudanças	√	√	×	√	√	√
Restrições para modificação	×	×	×	√	√	√
Propagação de mudanças	implícita	implícita	implícita	implícita	implícita	implícita
Métodos de Propagação	conversão imediata	conversão imediata	conversão imediata	conversão imediata	híbrido	híbrido (versões)
Atualização de dados	√	√	√	×	×	×
Processamento de Consultas	√	√	√	×	×	×
Compatibilidade com ODMG	⋆	⋆	⋆	−−	−−	−−
Semântica para Evolução de Esquemas	×	×	×	invariantes	×	×

Legenda: √ Possui × Não possui −− Não conhecido ⋆ Não se aplica

Tabela 9.2: Comparativo do modelo proposto com os demais analisados

Itens / Propostas	**Farandole 2**	**Sades**	**TIGUKAT**	**TVOO**	**OODM$\|_{SV}$**	**SERF**	**TVSE**
Modelo de Dados	×	×	−−	OO	OO	×	OO
Dimensão Temporal	×	×	−−	TT	bitemporal	×	bitemporal
Gerenciamento da Intenção	×	×	−−	−−	bitemporal	×	bitemporal
Gerenciamento da Extensão	×	×	−−	−−	bitemporal	×	bitemporal
Armazenamento da Extensão	−−	−−	−−	−−	bitemporal	−−	mono, multi
Sincronismo entre Intenção e Extensão	×	×	−−	assíncrono	síncrono	×	síncrono, assíncrono
Granularidade do Versionamento	esquemas	classes	esquemas, tipos, objetos	esquemas, classes, objetos	esquemas	×	esquemas, objetos
Método para evolução de esquemas	versões	versões	versões	versões	versões	conversão	versões temporais
Taxonomia de mudanças	√	−−	√	√	√	√	√
Restrições para modificação	√	−−	√	√	√	√	√
Propagação de mudanças	implícita, explícita	implícita, explícita	implícita	implícita, explícita	implícita, explícita	implícita	implícita, explícita
Métodos de Propagação	multi-objetos	conversão adiada	conversão adiada	conversão adiada, imediata (versões)	conversão imediata	conversão imediata	híbrido (versões)
Atualização de dados	×	×	√	×	√	×	√
Processamento de Consultas	×	×	√	×	√	×	×
Compatibilidade com ODMG	−−	−−	√	−−	√	√	√
Semântica para Evolução de Esquemas	×	×	modelo axiomático	modelo axiomático	semântica operacional	×	semântica operacional

Legenda: √ Possui × Não possui −− Não conhecido ⋆ Não se aplica

Comparando com os trabalhos relacionados, constata-se que o mecanismo de evolução de esquemas proposto pelo TVSE é o mais completo com relação ao atendimento das estratégias de modificação de esquemas e propagação de mudanças. Além disso, TVSE é o único modelo que utiliza o conceito de tempo e de versão homogeneamente, tanto no nível intencional quanto no nível extensional. Aparentemente, um dos inconvenientes consiste na excessiva proliferação de versões de esquemas e de objetos. Como conseqüência, há a dificuldade na navegação pela estrutura de versões, complexidade no controle de compatibilidade entre esquemas e objetos e o acréscimo

no custo de armazenamento do histórico das versões. Entretanto, o cuidado com a excessiva proliferação de versões é tratado durante o gerenciamento das versões de esquemas e também nas estratégias de propagação de mudanças, conforme salientado nas contribuições.

Com relação ao modelo de objetos adotado, nenhum trabalho relacionado usufrui de todas as particularidades de um modelo de objetos como o TVSE. Uma categoria de trabalhos, como TIGUKAT, OODM$|_{SV}$ e TVOO utiliza um modelo temporal de objetos para gerenciar a evolução de esquemas. Contudo, o mecanismo de evolução de esquemas e o modelo de objetos são considerados ortogonais, ou seja, são manipulados independentemente. Em particular, o TVSE é a única abordagem que utiliza todas as funcionalidades de um modelo de objetos no tratamento da evolução de esquemas. Além disso, uma característica ímpar presente no TVM e conseqüentemente no TVSE é a união dos conceitos de tempo e de versão. Enquanto as propostas tratam isoladamente versões e dados temporais, o TVM une esses dois conceitos tratando-os homogeneamente. Assim, a adoção do TVM para evolução de esquemas traz consigo duas vantagens singulares. O versionamento permite tratar aplicações de projeto de banco de dados, por exemplo, quando há necessidade de alternativas. Aliado a isso, a temporalidade armazena a história da evolução através do tempo de validade, permitindo reconstruir o estado do esquema em qualquer data passada sem usar operações de recuperação (por exemplo, *backup* e *recovery*) por meio do tempo de transação.

Os dois modelos mais completos de evolução de esquemas são OODM$|_{SV}$ e TIGUKAT. Mesmo assim, o TVSE supre as suas limitações no que diz respeito à manipulação de dados durante a modificação de esquemas e propagação de mudanças. Através de uma linguagem de manipulação de dados, o TVSE permite, naturalmente, gerenciar e manter a atualização e a consulta dos dados durante a evolução de esquemas, tornando o processo transparente ao usuário.

9.2 Trabalhos Futuros

Existem ainda diversos aspectos que foram identificados e devem ser explorados e aprimorados futuramente, fora do âmbito dessa tese:

- extensão do modelo para o tratamento de diretivas de indexação. O TVSE não provê mecanismos para gerenciar o processo de indexação que pode ser afetado pelas modificações de esquemas. Por exemplo, a exclusão de uma classe que possui um índice ou a exclusão de um atributo componente de um índice pode implicar a recompilação ou reconstrução do índice durante o mecanismo de propagação de mudanças;
- desenvolvimento de uma arquitetura espaço-temporal para o versionamento de esquemas. O TVSE considera como rótulo temporal o tempo de transação e de validade para as versões de esquemas. O TVSE pode ser estendido a fim de agregar a dimensão de localização, permitindo especificar, além do tempo, a área de utilização das versões de esquemas. Uma aplicação concreta para essa arquitetura é o tratamento de versões de esquemas oriundas de fontes heterogêneas de dados;
- métricas para evolução de esquemas. Especificação de métricas para avaliar a estabilidade da evolução do esquema conceitual do ponto de vista operacional. Por exemplo, o requisito extensibilidade pode ser uma métrica para avaliar o mínimo impacto que uma operação de modificação de esquemas poderia causar

nas aplicações. Assim, além de melhorar a prática de evolução de esquemas, tais métricas poderiam ser utilizadas para comparar diversos modelos e propostas presentes na literatura;

- desenvolvimento de um sistema (componente) para o processo de evolução de esquemas. Atualmente o ambiente implementado gerencia a evolução de esquemas, armazenando a extensão em repositórios múltiplos. Algumas limitações foram realizadas durante a implementação, pois o intuito foi mostrar a viabilidade do modelo como um todo, em detrimento de aspectos peculiares do TVSE. Assim, a implementação deve ser continuada para suportar todo processo de evolução e versionamento proposto pelo TVSE. Uma interface *Web* deve ser também desenvolvida a fim de tornar o TVSE disponível na Internet;
- análise comparativa do desempenho da implementação de evolução de esquemas com armazenamento em um único repositório, em múltiplos repositórios e com o modelo híbrido proposto nesta tese. Um estudo nesse estilo, porém no contexto de bancos de dados temporais relacionais, foi realizado em (WEI; ELMASRI, 1999), mostrando o tempo gasto no processamento de consultas temporais, o espaço requerido para as novas versões de objetos e o tempo necessário para a localização da versão corrente dos objetos antes da propagação das mudanças na base de dados.

9.3 Considerações Finais

Esta tese é uma proposta de solução para evolução dinâmica de esquemas em bancos de dados temporais orientados a objetos. É proposto um modelo flexível de suporte a evolução de esquemas, bem como estratégias de propagação das mudanças nas instâncias vigentes na base de dados e a atualização de dados durante a evolução. Esta tese representa uma contribuição para a pesquisa e o desenvolvimento de mecanismos para gerenciamento o de evolução de bancos de dados temporais orientados a objetos, tanto no campo conceitual quanto no campo operacional.

Uma vasta gama de trabalhos encontrados na literatura tem buscado propor soluções para a questão de evolução de esquemas, entretanto ainda não existe um consenso entre as propostas apresentadas nem um modelo completamente definido. O diferencial desta tese consiste em prover uma generalidade frente ao processo de evolução de esquemas conceituais, ao invés de dar ênfase a conceitos esparsos, investigando, explorando e tratando os vários aspectos envolvidos no processo de evolução de esquemas em bancos de dados temporais orientados a objetos. Além disso, esta tese leva em conta todas as particularidades do processo de evolução de esquemas, considerando de forma mais precisa cada etapa envolvida no processo.

Além disso, o TVSE adota um modelo de objeto com características de tempo e de versão para gerenciar a evolução dos esquemas e objetos. A derivação de versões abrange não somente os objetos da base de dados, mas também esquemas e aplicativos. Entretanto, todos esses conceitos são tratados uniformemente como os objetos, de forma que o modelo TVM adotado pode ser perfeitamente utilizado no processo de retenção do histórico da evolução de esquemas.

Enfim, foi mostrado que o uso dos conceitos de tempo e de versão no processo de evolução de esquemas apresenta uma vantagem significativa por permitir a definição de diferentes organizações conceituais para um mesmo domínio da aplicação, mantendo a funcionalidade das instâncias armazenadas frente às alterações, evitando a perda de informações e garantindo a compatibilidade dos programas aplicativos. Não menos

importante é a possibilidade do armazenamento do histórico das modificações, permitindo a manipulação e a consulta às versões que habitam simultaneamente o esquema conceitual e a base de dados.

REFERÊNCIAS

AGRAWAL, R.; BUROFF, S.; GEHANI, N. H.; SHASHA, D. Object Versioning in Ode. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ICDE, 1991, Kobe, Japan. **Proceedings...** Los Alamitos: IEEE Computer Society, 1991. p.446–455.

AL-JADIR, L.; ESTIER, T.; FALQUET, G.; LÉONARD, M. Evolution Features of the F2 OODBMS. In: INTERNATIONAL CONFERENCE ON DATABASE SYSTEMS FOR ADVANCED APPLICATIONS, DASFAA, 4., 1995, Singapore. **Proceedings...** [S.l.]: World Scientific, 1995. p.284–291. (Advanced Database Research and Development Series, v.5).

AL-JADIR, L.; LÉONARD, M. Multiobjects to Ease Schema Evolution in an OODBMS. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER, 1998, Singapore. **Proceedings...** Berlin: Springer-Verlag, 1998. p.316–333. (Lecture Notes in Computer Science, v.1507).

ANGONESE, S. F. **Gerenciamento Temporal de Versões de Esquemas**. 2000. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

ANTUNES, D. C. **Modelagem Temporal de Sistemas**: uma abordagem fundamentada em redes de petri. 1997. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

ARDENT SOFTWARE. **The O2 System - Release 5.0 Documentation**. [S.l.], 1998.

ARIAV, G. A Temporally Oriented Data Model. **ACM Transactions on Database Systems, TODS**, [S.l.], v.11, n.4, p.499–527, Dec. 1986.

BAKKER, J. D. Semantics of programming languages. **Advances in Information Systems Science**, [S.l.], n.2, p.173–227, 1969.

BANERJEE, J.; KIM, W.; KIM, H.-J.; KORTH, H. F. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1987, San Francisco, California. **Proceedings...** New York: ACM Press, 1987. p.311–322.

BEECH, D.; MAHBOD, B. Generalized Version Control in an Object-Oriented Database. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ICDE, 1988, Los Angeles, California, USA. **Proceedings...** Los Alamitos: IEEE Computer Society, 1988. p.14–22.

BENATALLAH, B. A Unified Framework for Supporting Dynamic Schema Evolution in Object Databases. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER, 18., 1999, Paris, France. **Proceedings...** Berlin: Springer-Verlag, 1999. p.16–30. (Lecture Notes in Computer Science, v.1728).

BENATALLAH, B.; TARI, Z. Dealing with Version Pertinence to Design an Efficient Schema Evolution Framework. In: INTERNATIONAL DATABASE ENGINEERING AND APPLICATIONS SYMPOSIUM, IDEAS, 1998, Cardiff, Wales, UK. **Proceedings...** Los Alamitos: IEEE Computer Society, 1998. p.24–33.

BIERMAN, G. M. Formal semantics and analysis of object queries. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2003, San Diego, California, USA. **Proceedings...** New York: ACM Press, 2003. p.407–418.

BIERMAN, G. M.; PARKINSON, M. J.; PITTS, A. M. **MJ**: an imperative core calculos for java and java with effects. [S.l.]: University of Cambridge, 2003.

BILIRIS, A. Modeling Design Object Relationships in PEGASUS. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ICDE, 1990, Los Angeles, California, USA. **Proceedings...** Los Alamitos: IEEE Computer Society, 1990. p.228–236.

BJÖRNERSTEDT, A.; HULTÉN, C. Version Control in an Object-Oriented Architecture. In: KIM, W.; LOCHOVSKY, F. H. (Ed.). **Object-Oriented Concepts, Databases, and Applications**. [S.l.]: ACM Press and Addison-Wesley, 1989. p.451–485.

BRUCE, K. B. **Foundations of Object-Oriented Languages - Types and Semantics**. [S.l.]: The Mit Press, 2002.

BUDDRUS, F.; GÄRTNER, H.; LAUTEMANN, S.-E. First Steps to a Formal Framework for Multilevel Database Modifications. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 8., 1997, Toulouse, France. **Proceedings...** Berlin: Springer-Verlag, 1997. p.240–251. (Lecture Notes in Computer Science, v.1308).

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

CASTRO, C. de; GRANDI, F.; SCALAS, M. R. On Schema Versioning in Temporal Databases. **Recent Advances in Temporal Databases**, [S.l.], p.272–291, Sept. 1995. Trabalho apresentado no International Workshop on Temporal Databases, 1995.

CASTRO, C. de; GRANDI, F.; SCALAS, M. R. Extensional Data Management in Multitemporal Relational Databases Supporting Schema Versioning. In: CONVEGNO NAZIONALE SU SISTEMI EVOLUTTI PER BASI DI DATI, SEBD, 1995, Ravello. **Proceedings...** [S.l.: s.n.], 1995.

CASTRO, C. de; GRANDI, F.; SCALAS, M. R. Schema Versioning for Multitemporal Relational Databases. **Information Systems**, [S.l.], v.22, n.5, p.249–290, July 1997.

CATTELL, R.; BARRY, D. K. (Ed.). **The Object Data Standard**: ODMG 3.0. San Francisco: Morgan Kaufmann, 2000. 280p.

CHOU, H.-T.; KIM, W. A Unifying Framework for Version Control in a CAD Environment. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB, 1986, Kyoto, Japan. **Proceedings...** San Francisco: Morgan Kaufmann, 1986. p.336–344.

CLAMEN, S. M. **Type Evolution and Instance Adaptation**. Pittsburgh, PA: Carnegie Mellon University School of Computer Science, 1991. (CMU-CS-92-133).

CLAYPOOL, K. T.; JIM, J.; RUNDENSTEINER, E. A. **OQL_SERF**: an ODMG implementation of the template-based schema evolution framework. [S.l.]: Worcester Polytechnic Institute, 1998. (WPI-CS-TR-98-14).

CLAYPOOL, K. T.; JIN, J.; RUNDENSTEINER, E. A. SERF: schema evalution through an extensible, re-usable and flexible framework. In: INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, CIKM, 1998, Bethesda, Maryland, USA. **Proceedings...** New York: ACM Press, 1998. p.314–321.

CLAYPOOL, K. T.; NATARAJAN, C.; RUNDENSTEINER, E. A. **Optimizing the Performance of Schema Evolution Sequences**. [S.l.]: Worcester Polytechnic Institute, 1999. (WPI-CS-TR-99-06).

CLAYPOOL, K. T.; NATARAJAN, C.; RUNDENSTEINER, E. A. Optimizing Performance of Schema Evolution Sequences. In: INTERNATIONAL SYMPOSIUM ON OBJECT DATABASES, ECOOP, 2000, Sophia Antipolis, France. **Proceedings...** Berlin: Springer-Verlag, 2000. p.114–127. (Lecture Notes in Computer Science, v.1944).

CLAYPOOL, K. T.; RUNDENSTEINER, E. A. Flexible Database Transformations: teh serf approach. **IEEE Data Engineering Bulletin**, [S.l.], v.22, n.1, p.19–24, Mar. 1999.

CLAYPOOL, K. T.; RUNDENSTEINER, E. A.; HEINEMAN, G. T. ROVER: a framework for the evolution of relationships. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER, 19., 2000, Salt Lake City, Utah, USA. **Proceedings...** Berlin: Springer-Verlag, 2000. p.409–422. (Lecture Notes in Computer Science, v.1920).

CLAYPOOL, K. T.; RUNDENSTEINER, E. A.; HEINEMAN, G. T. ROVER: flexible yet consistent evolution of relationships. **Data & Knowledge Engineering**, [S.l.], v.39, n.1, p.27–50, Oct. 2001.

COLAZZO, D.; GHELLI, G.; MANGHI, P.; SARTIANI, C. Types for Correctness of Queries over Semistructured Data. In: INTERNATIONAL WORKSHOP ON THE WEB AND DATABASES, ACM PODS/SIGMOD, 2002, Madison, Wisconsin, USA. **Proceedings...** [S.l.: s.n.], 2002. p.19–24.

COMPUTER ASSOCIATES INTERNATIONAL, Inc. **The Jasmine Documentation**. [S.l.], 1998.

DAVIDSON, S. B.; KOSKY, A. Specifying Database Transformations in WOL. **IEEE Data Engineering Bulletin**, [S.l.], v.22, n.1, p.25–30, Mar. 1999.

DRAPER, D.; FANKHAUSER, P.; FERNÁNDEZ, M.; MALHOTA, A.; ROSE, K.; RYS, M.; SIMÉON, J.; WADLER, P. **XQuery 1.0 and XPath 2.0 Formal Semantics**. In W3C Working Draft. Disponível em: <http://www.w3.org/TR/2003/WD-xquery-semantics-20030502/>. Acesso em: 15 jul. 2003.

EDELWEISS, N. **Sistemas de informação de escritórios** : um modelo para especificãções temporais. 1994. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

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

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 3.ed. [S.l.]: Addison-Wesley Longman, 2000.

FERRANDINA, F.; MEYER, T.; ZICARI, R.; FERRAN, G.; MADEC, J. Schema and Database Evolution in the O2 Object Database System. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB, 1995, Zurich, Switzerland. **Proceedings...** San Francisco: Morgan Kaufmann, 1995. p.170–181.

FRANCONI, E.; GRANDI, F.; MANDREOLI, F. A Semantic Approach for Schema Evolution and Versioning in Object-Oriented Databases. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL LOGIC, CL, 1., 2000, London, UK. **Proceedings...** Berlin: Springer-Verlag, 2000. p.1048–1062. (Lecture Notes in Computer Science, v.1861).

FRANCONI, E.; GRANDI, F.; MANDREOLI, F. Schema Evolution and Versioning: a logical and computational characterisation. In: INTERNATIONAL WORKSHOP ON FOUNDATIONS OF MODELS AND LANGUAGES FOR DATA AND OBJECTS, FOMLADO, 9., 2000, Dagstuhl Castle, Germany. **Proceedings...** Berlin: Springer-Verlag, 2000. p.85–99. (Lecture Notes in Computer Science, v.2065).

GADIA, S. K. **A Seamless Generic Extension of SQL for Quering Temporal Data**. [S.l.]: Iwoa State University Computer Science Department, 1992. (TR-92-02).

GALANTE, R. M. **Um Modelo de Evolução de Esquemas Conceituais para Bancos de Dados Orientados a Objetos com o Emprego de Versões**. 1998. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

GALANTE, R. M. **Evolução de esquemas em bancos de dados orientados a objetos com o emprego de versões**. [S.l.]: Instituto de Informática, UFRGS, 2001. 141p. Exame de Qualificação (Doutorado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

GALANTE, R. M.; EDELWEISS, N.; SANTOS, C. S. dos. Evolução de Esquemas e Propagação de Mudanças usando o Modelo Temporal de Versões. In: CONFERENCIA LATINOAMERICANA DE INFORMÁTICA, CLEI, 28., 2002, Montevideo, Uruguay. **Proceedings...** [S.l.: s.n.], 2002.

GALANTE, R. M.; EDELWEISS, N.; SANTOS, C. S. dos. Change Management for a Temporal Versioned Object-Oriented Database. In: INTERNATIONAL WORKSHOP ON EVOLUTION AND CHANGE IN DATA MANAGEMENT, ECDM, INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELLING, ER, 2002, Tampere, Finland. **Proceedings...** Berlin: Springer-Verlag, 2002. p.1–12. (Lecture Notes in Computer Science, v.2784).

GALANTE, R. M.; EDELWEISS, N.; SANTOS, C. S. dos. TVL_SE: temporal and versioning language for schema evolution in object-oriented databases. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 2003. **Proceedings...** Berlin: Springer-Verlag, 2003. (Lecture Notes in Computer Science, v.2736).

GALANTE, R. M.; EDELWEISS, N.; SANTOS, C. S. dos; MOREIRA, A. F. Data Modification Language for Full Support of Temporal Schema Versioning. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 18., 2003. **Anais...** [S.l.: s.n.], 2003.

GALANTE, R. M.; SANTOS, C. S. dos; EDELWEISS, N. Gerenciamento dinâmico de evolução de esquemas usando o TVM. In: WORKSHOP DE TESES E DISSERTAÇÕES EM BANCO DE DADOS, WTDBD, 1., 2002, Gramado, RS, Brazil. **Anais...** Porto Alegre: Instituto de Informática: UFRGS, 2002. p.72–76. Em conjunto com XVII Simpósio Brasileiro de Banco de Dados, SBBD.

GALANTE, R. M.; SANTOS, C. S. dos; EDELWEISS, N. Temporal and Versioning Approach to Schema Evolution in Object-Oriented Databases. **Data and Knowledge Engineering**, [S.l.], 2003. Em processo de revisão.

GALANTE, R. M.; SANTOS, C. S. dos; RUIZ, D. D. D. A. Um Modelo de Evolução de Esquemas Conceituais para Bancos de Dados Orientados a Objetos com o Emprego de Versões. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 13., 1998, Maringá, Paraná. **Anais...** Maringá: UEM-DIN, 1998. p.303–318.

GALANTE, R. M.; SILVA ROMA, A. B. da; JANTSCH, A.; EDELWEISS, N.; SANTOS, C. S. dos. Dynamic schema evolution management using version in temporal object-oriented databases. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 13., 2002, Aix-en-Provence, France. **Proceedings...** Berlin: Springer-Verlag, 2002. p.524–533. (Lecture Notes in Computer Science, v.2453).

GANÇARSKI, S.; JOMIER, G. A framework for programming multiversion databases. **Data & Knowledge Engineering**, [S.l.], v.36, n.1, p.29–53, Jan. 2001.

GEMSTONE SYSTEMS, INC. **GemStone/S Documentation - Release 6.0.1**. [S.l.], 2003.

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

GORALWALLA, I. A.; SZAFRON, D.; ÖZSU, M. T.; PETERS, R. J. Managing Schema Evolution Using a Temporal Object Model. In: INTERNATIONAL CONFERENCE ON

CONCEPTUAL MODELING, ER, 1997, Los Angeles, California, USA. **Proceedings...** Berlin: Springer-Verlag, 1997. p.71–84. (Lecture Notes in Computer Science, v.1331).

GORALWALLA, I. A.; SZAFRON, D.; ÖZSU, M. T.; PETERS, R. J. A temporal approach to managing schema evolution in object database systems. **Data & Knowledge Engineering**, [S.l.], v.8, n.1, p.73–105, Oct. 1998.

GRANDI, F. A Relational Multi-Schema Data Model and Query Language for Full Support of Schema Versioning. In: NATIONAL CONFERENCE ON ADVANCED DATABASE SYSTEMS, 2002, Isola d'Elba, Italy. **Proceedings...** [S.l.: s.n.], 2002. p.323–336.

GRANDI, F.; MANDREOLI, F. ODMG Language Extensions for Generalised Schema Versioning Support. In: WORKSHOPS ON EVOLUTION AND CHANGE IN DATA MANAGEMENT, REVERSE ENGINEERING IN INFORMATION SYSTEMS, AND THE WORLD WIDE WEB AND CONCEPTUAL MODELING, 1999, Paris, France. **Proceedings...** Berlin: Springer-Verlag, 1999. p.36–47. (Lecture Notes in Computer Science, v.1727).

GRANDI, F.; MANDREOLI, F. A Formal Model for Temporal Schema Versioning in Object-Oriented Databases. **Data & Knowledge Engineering**, [S.l.], v.2, n.46, p.123–167, Aug. 2003.

HÜBLER, P. N.; EDELWEISS, N. Implementing a Temporal Database on Top of a Conventional Database: mapping of the data model and data definition management. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 15., 2000, João Pessoa, Paraíba, Brasil. **Anais...** Porto Alegre: PUCRS, 2000. p.259–272.

ITASCA DISTRIBUTED OBJECT DATABASE MANAGEMENT SYSTEM. **Technical Summary for Release 2.3.5**. [S.l.], 1995.

JANTSCH, A. **TVMSE - Uma Implementação do Versionamento de Esquemas segundo o Modelo TVM**. 2003. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

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

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

KÄFER, W.; SCHÖNING, H. Realizing a Temporal Complex-Object Data Model. **SIGMOD Records**, New York, v.21, n.2, p.266–275, June 1992. Trabalho apresentado na ACM SIGMOD International Conference on Management of Data, 1992.

KIM, W.; BERTINO, E.; GARZA, J. F. Composite Objects Revisted. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1989, Portland, Oregon. **Proceedings...** New York: ACM Press, 1989. p.337–347.

LAGORCE, J.-B.; STOCKUS, A.; WALLER, E. Object-Oriented Database Evolution. In: INTERNATIONAL CONFERENCE ON DATABASE THEORY, ICDT, 6., 1997, Delphi,

Greece. **Proceedings...** Berlin: Springer-Verlag, 1997. p.379–393. (Lecture Notes in Computer Science, v.1186).

LAUNER, L. **Formal definition of Algol 60**. Vienna: IBM Lab, 1968. TR.25.088.

LAUTEMANN, S.-E. An Introduction to Schema Versioning in OODBMS. In: INTERNATIONAL WORKSHOP ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 7., 1996, Zurich, Switzerland. **Proceedings...** [S.l.]: IEEE-CS Press, 1996. p.132–139.

LAUTEMANN, S.-E. A Propagation Mechanism for Populated Schema Versions. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ICDE, 1997, Birmingham U.K. **Proceedings...** Los Alamitos: IEEE Computer Society, 1997. p.67–78.

LAUTEMANN, S.-E. Schema Versions in Object-Oriented Database Systems. In: INTERNATIONAL CONFERENCE ON DATABASE SYSTEMS FOR ADVANCED APPLICATIONS, DASFAA, 1997, Melbourne, Australia. **Proceedings...** [S.l.]: World Scientific, 1997. p.323–332. (Advanced Database Research and Development Series, v.6).

LAUTEMANN, S.-E. Change Management with Roles. In: INTERNATIONAL CONFERENCE ON DATABASE SYSTEMS FOR ADVANCED APPLICATIONS, DASFAA, 6., 1999, Hsinchu, Taiwan. **Proceedings...** Los Alamitos: IEEE Computer Society, 1999. p.291–300.

LAUTEMANN, S.-E.; APOSTOLIDIS, S.; DOLL, A.; HAASE, J. The COAST Object Database System. In: OBJECT-ORIENTED TECHNOLOGY, ECOOP, 1999, Lisbon, Portugal. **Proceedings...** Berlin: Springer-Verlag, 1999. p.378–380. (Lecture Notes in Computer Science, v.1743).

LERNER, B. S. A model for compound type changes encountered in schema evolution. **ACM Transactions on Database Systems, TODS**, [S.l.], v.25, n.1, p.83–127, Mar. 2000.

LUCAS, P. Formal Definition of Programming Languages and Systems. In: IFIP CONGRESS, 1972, Ljubljana, Yugoslavia. **Proceedings...** [S.l.]: North-Holland, 1972. p.291–297. (Information Processing, v.1).

MCCARTHY, J. L. Towards a Mathematical Science of Computation. In: IFIP CONGRESS, 1962, Munich, Germany. **Proceedings...** [S.l.]: North-Holland, 1962. p.21–28. (Information Processing).

MEDEIROS, C. B.; BELLOSTA, M.-J.; JOMIER, G. Multiversion views: constructing views in a multiversion database. **Data & Knowledge Engineering**, [S.l.], v.33, n.3, p.277–306, June 2000.

MILNER, R.; TOFTE, M.; HARPER, R.; MACQUEEN, D. **The Definition of Standard ML (Revised)**. [S.l.]: The MIT Press, 1997.

MONK, S. R.; SOMMERVILLE, I. Schema Evolution in OODBs Using Class Versioning. **SIGMOD Record**, [S.l.], v.3, n.22, p.16–22, Sept. 1993.

MOREIRA, A. F.; GALANTE, R. M.; EDELWEISS, N.; SANTOS, C. S. dos. Semântica Operacional e Sistemas de Tipos para Linguagens de Banco de Dados. In: WORKSHOP DE MÉTODOS FORMAIS, WMF, 6., 2003, Campina Grande, Paraíba, Brasil. **Minicurso...** Campina Grande: UFCG, 2003.

MOREIRA, A. F.; GALANTE, R. M.; EDELWEISS, N.; SANTOS, C. S. dos. **Semantics of TVL/SE - A Temporal and Versioning Language for Schema Evolution in Object-Oriented Databases**. Submetido para publicação.

MOREIRA, V. P. **Consultas a Bancos de Dados Temporais que Suportam Versionamento de Esquemas**. 1999. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

MOREIRA, V. P.; EDELWEISS, N. Schema Versioning: queries to the generalized temporal database system. In: SPATIO-TEMPORAL DATA MODELS AND LANGUAGES, STDML; INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 1999, Florence, Italy. **Proceedings...** Los Alamitos: IEEE Computer Society, 1999. p.458–459.

MOREIRA, V. P.; EDELWEISS, N. Queries to Temporal Databases Supporting Schema Versioning. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 14., 1999, Florianópolis, SC, Brasil. **Anais...** Florianópolis: UFSC, 1999. p.299–313.

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

MORO, M. M.; EDELWEISS, N.; ZAUPA, A. P.; SANTOS, C. S. dos. TVQL - Temporal Versioned Query Language. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 13., 2002, Aix-en-Provence, France. **Proceedings...** Berlin: Springer-Verlag, 2002. p.618–627. (Lecture Notes in Computer Science, v.2453).

MORO, M. M.; GALANTE, R. M.; EDELWEISS, N.; SANTOS, C. S. dos. **Versions and Time in Databases Research Group**. Disponível em: <http://metropole.inf.ufrgs.br/versoestempo>. Acesso em: 12 jul. 2003.

MORO, M. M.; GELATTI, P. C.; GOMES, C. H. P.; ROSSETTI, L. L. F.; ZAUPA, A. P.; EDELWEISS, N.; SANTOS, C. S. dos. **Linguagem de Consulta para o Modelo Temporal de Versões**. Porto Alegre: Universidade Federal do Rio Grande do Sul, 2001. (RP-308).

MORO, M. M.; SAGGIORATO, S. M.; EDELWEISS, N.; SANTOS, C. S. dos. Adding Time to an Object-Oriented Versions Model. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 12., 2001, Munich, Germany. **Proceedings...** Berlin: Springer-Verlag, 2001. p.805–814. (Lecture Notes in Computer Science, v.2113).

MORO, M. M.; SAGGIORATO, S. M.; EDELWEISS, N.; SANTOS, C. S. dos. Dynamic Systems Specification Using Versions and Time. In: INTERNATIONAL DATABASE ENGINEERING & APPLICATIONS SYMPOSIUM, IDEAS, 2001, Grenoble, France. **Proceedings...** Los Alamitos: IEEE Computer Society, 2001. p.99–107.

MORO, M. M.; SAGGIORATO, S. M.; EDELWEISS, N.; SANTOS, C. S. dos. A Temporal Versions Model for Time-Evolving Systems Specification. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, SEKE, 13., 2001, Buenos Aires, Argentina. **Proceedings...** [S.l.: s.n.], 2001. p.252–259.

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

NGUYEN, G. T.; RIEU, D. Schema Evolution in Object-Oriented Database Systems. **Data & Knowledge Engineering**, [S.l.], v.4, n.1, p.43–67, July 1989.

OBJECT DESIGN INC. **ObjectStore Release 6.1**. [S.l.], 2003.

OBJECTIVITY INC. **Objectitivity Technical Overview - Release 6.0**. [S.l.], 2001.

OBJECTIVITY INC. **Schema Evolution in Objectivity/DB**. [S.l.], 2001.

OMG, Object Management Group. **OMG Unified Modeling Language Specification**. Version 1.5: March 2003. Disponível em: <http://www.omg.org/cgi-bin/doc?formal/03-03-01>. Acesso em: 04 set. 2003.

PEIXOTO, C. E. L. **Ambiente de Especificação e Evolução de Esquemas no TVM**. 2003. Trabalho de Diplomação (Bacharelado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

PEIXOTO, C. E. L.; GASPARY, D. F.; MORO, M. M.; EDELWEISS, N. Ferramenta de Apoio à Especificação de Classes do Modelo Temporal de Versões. In: SALÃO DE INICIAÇÃO CIENTÍFICA, 2002. **Livro de Resumos...** Porto Alegre: UFRGS, 2002.

PENNEY, D. J.; STEIN, J. Class Modification in the GemStone Object-Oriented DBMS. **SIGPLAN Notices**, New York, v.22, n.12, p.111–117, Dec. 1987. Trabalho apresentado na OOPSLA, 1987.

PETERS, R. J. **TIGUKAT: a uniform behavioral objectbase management system**. Edmonton, Alberta, Canada: University of Alberta, 1994. (TR-94-06).

PETERS, R. J.; BARKER, K. Change Propagation in an Axiomatic Model of Schema Evolution for Objectbase Management Systems. In: INTERNATIONAL WORKSHOP ON FOUNDATIONS OF MODELS AND LANGUAGES FOR DATA AND OBJECTS FOMLADO, 9., 2000, Dagstuhl Castle, Germany. **Proceedings...** Berlin: Springer-Verlag, 2000. p.142–162. (Lecture Notes in Computer Science, v.2065).

PETERS, R. J.; LIPKA, A.; ÖZSU, M. T.; SZAFRON, D. An Extensible Query Model and Its Languages for a Uniform Behavioral Object Management System. In: INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, CIKM, 2., 1993, Washington, DC, USA. **Proceedings...** New York: ACM Press, 1993. p.403–412.

PETERS, R. J.; ÖZSU, M. T. Axiomatization of Dynamic Schema Evolution in Objectbases. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ICDE, 11., 1995, Taipei, Taiwan. **Proceedings...** Los Alamitos: IEEE Computer Society, 1995. p.156–164.

PETERS, R. J.; ÖZSU, M. T. An Axiomatic Model of Dynamic Schema Evolution in Objectbase Systems. **ACM Transactions on Database Systems**, [S.l.], v.22, n.1, p.75–114, Mar. 1997.

PL/I Definition Group. **Formal definition of PL/I version 1**. [S.l.]: American Nat.Standards Institute, 1986. TR25.071.

PLOTKIN, G. D. Operational Semantics for CSP. In: IFIP TC2 WORKING CONFERENCE ON FORMAL DESCRIPTION OF PROGRAMMING CONCEPTS, 1983, Garmisch-Partenkirchen, Germany. **Proceedings...** [S.l.: s.n.], 1983. p.199–225.

POET SOFTWARE. **POET 5.0 Documentation Set**. [S.l.], 1997.

RA, Y.-G.; RUNDENSTEINER, E. A. A Transparent Schema-Evolution System Based on Object-Oriented View Technology. **IEEE Transactions on Knowledge and Data Engineering**, [S.l.], v.9, n.3, p.600–624, May/June 1997.

RASHID, A. A Database Evolution Approach for Object-Oriented Databases. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, ICSM, 2001, Florence, Italy. **Proceedings...** Los Alamitos: IEEE Computer Society, 2001. p.561–564.

RASHID, A.; SAWYER, P.; PULVERMUELLER, E. A Flexible Approach for Instance Adaptation During Class Versioning. In: INTERNATIONAL SYMPOSIUM OBJECTS AND DATABASES, ECOOP, 2000, Sophia Antipolis, France. **Proceedings...** Berlin: Springer-Verlag, 2000. p.101–113. (Lecture Notes in Computer Science, v.1944).

RIEDEL, H. Outdating Outdated Objects. In: OBJECT-ORIENTED TECHNOLOGY, ECOOP, 1999, Lisbon, Portugal. **Proceedings...** Berlin: Springer-Verlag, 1999. p.214–215. (Lecture Notes in Computer Science, v.1743).

RODDICK, J. A Survey of Schema Versioning Issues for Database Systems. **Information and Software Technology**, [S.l.], v.37, n.7, p.383–393, 1995.

RODDICK, J. F. Dynamically Changing Schemas Within Database Models. **Australian Computer Journal**, [S.l.], v.23, n.3, p.105–109, 1991.

RODDICK, J. F. **A model for temporal inductive inference and schema evolution in relational database systems**. 1994. Tese (Doutorado em Ciência da Computação) — Department of Computer Science and Computer Engineering, La Trobe University, Melbourn.

RODDICK, J. F.; GRANDI, F.; MANDREOLI, F.; SCALAS, M. R. Beyond Schema Versioning: a flexible model for spatio-temporal schema selection. **GeoInformatica**, [S.l.], v.5, n.1, p.33–50, Mar. 2001.

RODRÍGUEZ, L.; OGATA, H.; YANO, Y. TVOO: a temporal versioned object-oriented data model. **Information Sciences**, [S.l.], v.114, n.1-4, p.281–300, Mar. 1999.

RODRÍGUEZ, L.; OGATA, H.; 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.

ROMA, A. B. S.; GALANTE, R. M.; SANTOS, C. S. dos. Operações Primitivas e Complexas na Evolução de Esquemas em BDOO. In: IBEROAMERICAN WORKSHOP REQUIREMENTS ENGINEERING AND SOFTWARE ENVIRONMENTS, IDEAS, 3., 2000, Cancun, Mexico. **Proceedings...** [S.l.: s.n.], 2000.

ROMA, A. B. S.; GALANTE, R. M.; SANTOS, C. S. dos. Operações Complexas para Evolução de Esquemas em BDOO com o emprego de versões. In: CONFERENCIA LATINOAMERICANA DE INFORMÁTICA, CLEI, 28., 2001, Ciudad de Merida, Venezuela. **Proceedings...** [S.l.: s.n.], 2001.

ROMA, A. B. S.; JANTSCH, A.; GALANTE, R. M.; EDELWEISS, N.; SANTOS, C. S. dos. Gerenciamento Temporal de Versões para Evolução de Esquemas em BDOO. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 16., 2001, Rio de Janeiro, RJ, Brasil. **Anais...** Rio de Janeiro: COPPE/UFRJ, 2001. p.110–124.

RUNDENSTEINER, E. A.; CLAYPOOL, K. T.; CHEN, L.; SU, H.; OENOKI, K. SERFing the Web: web site management made easy. **SIGMOD Record**, New York, v.29, n.2, p.585, June 2000. Trabalho apresentado na SIGMOD, 2000.

RUNDENSTEINER, E. A.; CLAYPOOL, K. T.; LI, M.; CHEN, L.; ZHANG, X.; NATARAJAN, C.; JIN, J.; LIMA, S. D.; WEINER, S. SERF: odmg-based generic re-structuring facility. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1999, Philadelphia, Pennsylvania, USA. **Proceedings...** New York: ACM Press, 1999. p.568–570.

SANTOS, C. S. **Implementação de Banco de Dados Temporais com Versionamento de Esquemas**: um estudo comparativo. 2003. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

SIMÉON, J.; WADLER, P. The essence of XML. **ACM SIGPLAN Notices**, New York, v.1, n.38, p.1–13, Jan. 2003. Trabalho apresentado na SIGPLAN-SIGACT, 2003.

SKARRA, A. H.; ZDONIK, S. B. The Management of Changing Types in an Object-Oriented Database. **ACM SIGPLAN Notices**, New York, v.21, n.11, p.483–495, Nov. 1986. Trabalho apresentado na OOPSLA, 1986.

SNODGRASS, R. (Ed.). **The TSQL2 Temporal Query Language**. [S.l.]: Kluwer, 1995.

SNODGRASS, R. T. The Temporal Query Language TQuel. **ACM Transactions on Database Systems, TODS**, [S.l.], v.12, n.2, p.247–298, June 1987.

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

SNODGRASS, R. T.; et al. TSQL2 Language Specification. **SIGMOD Record**, [S.l.], v.1, n.23, p.65–86, Mar. 1994.

SNODGRASS, R. T.; et al. A TSQL2 Tutorial. **SIGMOD Record**, [S.l.], v.23, n.3, p.27–33, Sept. 1994.

STEEL, T. Formal Language Description Languages for Computer Programming. In: IFIP WORKING CONFERENCE ON FORMAL LANGUAGE DESCRIPTION LANGUAGES, 1966, North-Holland. **Proceedings...** [S.l.: s.n.], 1966.

SU, H.; CLAYPOOL, K. T.; RUNDENSTEINER, E. A. Extending the Object Query Language for Transparent Metadata Access. In: INTERNATIONAL WORKSHOP ON FOUNDATIONS OF MODELS AND LANGUAGES FOR DATA AND OBJECTS, FOMLADO, 9., 2000, Dagstuhl Castle, Germany. **Proceedings...** Berlin: Springer-Verlag, 2000. p.182–201. (Lecture Notes in Computer Science, v.2065).

SU, S. Y. W.; CHEN, H.-H. M. Temporal Knowledge Representation Model OSAM*/T and Its Query Language OQL/T. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB, 1991, Barcelona, Catalonia, Spain. **Proceedings...** San Francisco: Morgan Kaufmann, 1991. p.431–442.

TALENS, G.; OUSSALAH, C.; COLINAS, M. F. Versions of Simple and Composite Objec. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB, 1993, Dublin, Ireland. **Proceedings...** San Francisco: Morgan Kaufmann, 1993. p.62–72.

TANSEL, A. U.; CLIFFORD, J.; GADIA, S.; JAJODIA, S.; SEGEV, A.; SNODGRASS, R. (Ed.). **Temporal Databases**: theory, design, and implementation. RedwoodCity: Benjamin/Cummings, 1993.

VERSANT OBJECT TECNOLOGY. **Versant Manuals for Reliase 5.0**. [S.l.], 1997.

WEI, H.-C.; ELMASRI, R. Study and Comparison of Schema Versioning and Database Conversion Techniques for Bi-Temporal Databases. In: INTERNATIONAL WORKSHOP ON TEMPORAL REPRESENTATION AND REASONING, TIME, 1999, Orlando, Florida, USA. **Proceedings...** Los Alamitos: IEEE Computer Society, 1999. p.88–98.

WEI, H.-C.; ELMASRI, R. Schema versioning and database conversion techniques for bi-temporal databases. **Annals of Mathematics and Artificial Intelligence**, [S.l.], v.30, n.1-4, p.23–52, 2000.

WEI, H.-C.; ELMASRI, R. PMTV: a schema versioning approach for bi-temporal databases. In: INTERNATIONAL WORKSHOP ON TEMPORAL REPRESENTATION AND REASONING, TIME, 2000, Cape Breton, Nova Scotia, Canada. **Proceedings...** Los Alamitos: IEEE Computer Society, 2000. p.143–151.

WUU, G. T. J.; DAYAL, U. A Uniform Model for Temporal and Versioned Object-oriented Databases. In: TANSEL, A. U.; CLIFFORD, J.; GADIA, S. K.; SEGEV, A.; SNODGRASS, R. T. (Ed.). **Temporal Databases**: theory, design, and implementation. Redwood City: Benjamin/Cummings, 1993. p.230–247.

YU, D.; KENNEDY, A.; SYMEANDREW, D.; GORDON, D.; SYME, D. Formalization of generics for the .NET common language runtime. In: ACM SIGPLAN-SIGACT SYMPOSIUM ON PRINCIPLES OF PROGRAMMING LANGUAGES, POPL, 31., 2004, Venice, Italy. **Proceedings...** New York: ACM Press, 2004. p.39–51.

ZAUPA, A. P. **Suporte a Consultas no Ambiente Temporal de Versões**. 2002. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

ÖZSU, M. T.; PETERS, R. J.; SZAFRON, D.; IRANI, B.; LIPKA, A.; MUñOZ, A. TIGUKAT: a uniform behavioral objectbase management system. **The VLDB Journal - Special Issues on Persistent Object Systems**, [S.l.], v.4, n.3, p.445–492, July 1995.

ÖZSU, M. T.; VALDURIEZ, P. **Principles of Distributed Database Systems**. 2nd.ed. New Jersey: Prentice-Hall, 1999. Chapter 5, p.102-158.

APÊNDICE A BNF DA LINGUAGEM DE VERSIONAMENTO DE ESQUEMAS

Este anexo apresenta a BNF simplificada para linguagem de criação e versionamento de esquemas. Através da linguagem o usuário pode especificar uma versão de esquema, bem como aplicar as operações de manipulação de versões, que atuam sobre o grafo acíclico dirigido, formado durante o processo de derivação de versões de esquemas. As principais operações aplicáveis sobre as versões de esquema são: `excluir`, `promover`, `ativar`, `bloquear`, `congelar` e `restaurar`.

Cabe salientar que especificação de cada versão de esquema em particular segue a Linguagem de Definição de Classes proposta para o TVM. Por esse motivo, o elemento não-terminal para especificação de classes (`<class>`) não está expandido neste anexo. A BNF completa da Linguagem de Definição de Classes do TVM pode ser encontra em (MORO, 2001), sendo apresentada de forma simplificada na seção 3.2.7.

A notação utilizada é uma BNF ("*Backus Naur Form*" ou "*Backus Normal Form*") simplificada, utilizada em gramáticas livres de contexto. A notação utilizada é a seguinte, sem recursão à esquerda:

- < > para delimitar as metavariáveis;
- | para separar duas alternativas;
- [ ] para itens opcionais;
- { }* para itens repetitivos (zero ou mais vezes);
- { }+ para itens repetitivos (uma ou mais vezes);
- ( ) para conjunto de opções;
- os símbolos são delimitados por um par de aspas duplas;
- símbolos terminais e não terminais são diferenciados em negrito pela apresentação dos terminais.

A precedência dos operadores é a seguinte: opcional, repetição, seqüência e alternativas.

```
<schemaVersionDagOperation> ::= ( <create>
                                | <promote>
                                | <derive>
                                | <activate>
                                | <block>
```

```
                                | <delete>
                                | <freeze>
                                | <restore> )
<create> ::=  CREATE SCHEMA <schema version name>
              <schema specification>
<promote> ::= PROMOTE SCHEMA <schema version selection> [<initial valid time>]
<derive> ::= DERIVE SCHEMA <schema version name>
             FROM <schema version selection>
<activate> ::= ACTIVATE SCHEMA <schema version selection> [<initial valid time>]
<block> ::= BLOCK SCHEMA <schema version selection> [<initial valid time>]
<delete> ::= DELETE SCHEMA <schema version selection>
<freeze> ::= FREEZE SCHEMA [ [cascade] <schema version selection>] [<final valid time>]
<restore> ::= RESTORE SCHEMA [ [top] <schema version selection>] [<initial valid time>]
<schema version name> ::= <identifier>
<schema version selection> ::= SET SCHEMA <selection condition>
<selection condition> ::= <schema version name>
   | [AND VALID <defined interval>]
     [AND TRANSACTION <defined interval>]
<defined interval> ::= "[" <initial instant> ".." <final instant> "]"
   | "["  ".." <final instant> "]"
<initial instant> ::= <instant value>
<final instant> ::= <instant value>
<initial valid time> ::= <instant value>
<final valid time> ::= <instant value>
<schema specification> ::= <class> {; <class>;}*
<class> ::= [public] [ abstract | final]
   CLASS <class name> [hasVersions] [ inherit
      [byExtension] <class name> [correspondence (1:1 | 1:n | n:1 | n:m)]]
      [[temporal] aggregate_of[n] <class name> (by value | by reference)
      {,[temporal]aggregate_of[n] <class name> (by value | by reference)}]*
   ( [ Properties:
        { [ public | private | protected] [static]
        [temporal] <attribute name> ":" <attribute domain> ; }+ ]
     [ Relationships:
        { [temporal] <relationship name> (0:1 | 0:n | 1:1 | 1:n | n:m)
        [inverse <inverse relationship name> ] <related class>; }+ ]
     [ Operations:
        { [ public | private | protected] [static]
        [abstract | final] <operation definitions> }+ ] )
<class name> ::= <identifier>
<attribute name> ::= <identifier>
<attribute domain> ::= <domain>
<relationship name> ::= <identifier>
<inverse relationship name> ::= <identifier>
<related class> ::= <identifier>
<operation definitions>::= [ <visibility> ] [static] [ abstract | final]
   <operation name> "(" [ <parameter list> ] ")" [ ":" <returned value domain> ]
   { ; [ <visibility> ] [static] [ abstract | final]
      <operation name> ( [ <parameter list> ] ) [ ":" <returned value domain> ] }*
<visibility> ::= public | private | protected
<operation name> ::= <identifier>
<parameter list> ::= <parameter list> ":" <parameter domain> [default <value> ]
   {; <parameter list> ":" <parameter domain> [default <value> ] }*
<returned value domain> ::= <domain>
<parameter domain> ::= <domain>
<domain> ::= integer
   | real
   | string
   | char
   | boolean
   | date
   | time
   | instant
   | OIDt
   | <set>
   | <class name>
<value> ::= <integer number>
   | <real number>
   | <string value>
   | <char value>
   | <boolean value>
   | <instant value>
   | <date value>
```

```
   | <hour value>
   | <OID value>
   | null
<set> ::= set ( <domain> )
<integer number> ::= <digit> { <digit> }*
<real number> ::= <digit> { <digit> }* "," <digit> { <digit> }*
<string value> ::= "'" { any character including blank  } "'"
<char value> ::=  "'" <letter> "'" | "'" <digit> "'"
<boolean value> ::= true | false
<instant value> ::= <date value> "," <hour value>
<date value> ::= <number> "/" <number> "/" <number> <number>
<hour value> ::= <number> ":" <number>
<OID value> ::= <pointer>
<pointer> ::= { any character including blank  }
<instant value> ::= <date value> "," [<hour value>]
<number> ::= <digit><digit>
<identifier> ::= <letter> { <letter> | <digit> |"_"}*
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter> :: =  A | B | C | D | E| F | G | H | I |J | K | L | M | N | O | P | Q | R
            | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j
            | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
```

APÊNDICE B BNF DA LINGUAGEM DE MODIFICAÇÃO DE ESQUEMAS

Este anexo apresenta a BNF simplificada para Linguagem de Atualização de Objetos. A linguagem permite expressar operações de modificação nas versões de esquemas, estando classificadas em: inclusão de elmentos `modify`, exclusão de elementos `delete` e alteração de elementos `modify`.

```
<primitiveUpdate> ::= (<add> | <delete> | <modify> ) [<schema version name>]
 [newSV <schema version name>]

<add> ::= ADD ( CLASS <class>
 | ATTRIBUTE [TEMPORAL] <attribute name> : <attribute domain> [DEFAULT value] IN <class name>
 | RELATIONSHIP [TEMPORAL] <relationship name>  (0:1|0:n|1:1|1:n|n:m)
  [INVERSE <relationship name> ] <class name> IN <class name>
 | OPERATION <operation name> "(" [<parameter list> ] ")" [ : <returned value domain> ] IN <class name> )

<delete> ::= DELETE (CLASS <class name>
 | ATTRIBUTE <attribute name> FROM <class name>
 | RELATIONSHIP <relationship name> FROM <class name> <class name>
 | OPERATION <operation name> )

<modify> ::= MODIFY (CLASS NAME <class name> INTO <class name>
 | CLASS TYPE <class name> INTO (hasVersions|normal)
 | ATTRIBUTE NAME <attribute name> INTO <attribute name> IN <class name>
 | ATTRIBUTE DOMAIN <attribute name> INTO <attribute domain> IN <class name>
 | ATTRIBUTE TYPE <attribute name> INTO (temporal|normal) IN <class name>
 | OPERATION NAME <operation name> INTO <operation name> ) IN <class name>
 | RELATIONSHIP NAME [INVERSE] <relationship name> INTO <relationship name> ) IN <class name>
 | RELATIONSHIP CARDINALITY [INVERSE] <relationship name> (1:1|1:n|n:1|n:n) IN <class name>
 | RELATIONSHIP TYPE <relationship name> INTO (temporal|normal) IN <class name> )

<schema version name>::= <identifier>
<class name> ::= <identifier>
<attribute name> ::= <identifier>
<relationship name> ::= <identifier>
<operation name> ::= <identifier>
<attribute domain> ::= <domain>
<returned value domain> ::= <domain>
<parameter list> ::= <parameter list> : <parameter domain> [DEFAULT <value>]
   {; <parameter list> : <parameter domain> [DEFAULT <value>]}*
<parameter domain> ::= <domain>
<domain> ::= integer
   | real
   | string
   | char
   | boolean
   | date
   | time
   | instant
   | OIDt
   | <set>
   | <class name>
<value> ::= <integer number>
```

```
   | <real number>
   | <string value>
   | <char value>
   | <boolean value>
   | <instant value>
   | <date value>
   | <hour value>
   | <OID value>
   | null
<set> ::= set ( <domain> )
<integer number> ::= <digit> { <digit> }*
<real number> ::= <digit> { <digit> }* "," <digit> { <digit> }*
<string value> ::= "'" { any character including blank  } "'"
<char value> ::=  "'" <letter> "'" | "'" <digit> "'"
<boolean value> ::= true | false
<instant value> ::= <date value> "," <hour value>
<date value> ::= <number> "/" <number> "/" <number> <number>
<hour value> ::= <number> ":" <number>
<OID value> ::= <pointer>
<pointer> ::= { any character including blank  }
<instant value> ::= <date value> "," [<hour value>]
<number> ::= <digit><digit>
<identifier> ::= <letter> { <letter> | <digit> |"_"}*
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter> :: =  A | B | C | D | E| F | G | H | I |J | K | L | M | N | O | P | Q | R
            | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j
            | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
```

APÊNDICE C BNF DA LINGUAGEM DE ATUALIZAÇÃO DE OBJETOS

Este anexo apresenta a BNF simplificada para Linguagem de Modificação de Esquemas. A linguagem permite expressar operações de modificação de dados, por exemplo `insert`, `update` e `delete`, considerando a vigência de versões no banco de dados intencional e extensional. É possível também expressar modificações que envolvem diferentes versões do esquema conceitual simultaneamente, denominadas atualizações multi-esquemas.

Cabe salientar que a recuperação de dados convencionais e de dados temporais versionados, considerando cada uma das versão do esquema, é realizada através da linguagem de consulta TVQL (cláusula `<condition>`), apresentada na seção 3.3. Por esse motivo, o elemento não-terminal para especificação de consultas TVQL não está expandido neste anexo. A BNF completa da linguagem de consulta TVQL pode ser encontra em (MORO et al., 2001), sendo apresentada de forma simplificada na seção 3.3.

```
<query>  ::= ( <insert> | <update> | <delete> ) <schema version selection>
<insert> ::= INSERT INTO <class name>
               [( <attribute name> = <expression> [VALIDITY INTERVAL <defined interval> ]
               [, <attribute name> = <expression> [VALIDITY INTERVAL <defined interval> ]]*)]
               VALUES ( <constant> [, <constant> ]*)
               [VALIDITY INTERVAL defined interval]
           [VALIDITY INTERVAL <defined interval> ]
<delete> ::= DELETE FROM <class name> [WHERE <condition> ][fValidity = <instant value> ]
<update> ::= UPDATE <class name>
               SET <attribute name> = <expression> [VALIDITY INTERVAL <defined interval> ]
               [, <attribute name> = <expression> [VALIDITY INTERVAL <defined interval> ] ]*
               WHERE <condition>
<schema version selection> ::= SET SCHEMA <selection condition>
<selection condition> ::= [<schema version name>]
   | [AND VALID <defined interval>]
     [AND TRANSACTION <defined interval>]
<defined interval> ::= "[" <initial instant> ".." <final instant> "]"
   | "["  ".." <final instant> "]"
<initial instant> ::= <instant value>
<final instant> ::= <instant value>
<schema version name> ::= <identifier>
<class name> ::= <identifier>
<attribute name> ::= <identifier>
<condition> ::= <query tvql>
<query tvql> ::= SELECT [ EVER ] [ DISTINCT ] <target clause>
        FROM <identific clause>
        [ WHERE [ EVER ] <search clause> ]
        [ GROUP BY <group clause> { , <group clause> }
          [ HAVING <logical expression> ] ]
        [ ORDER BY <order clause>  , <order clause>  ]
        [ <setOp> <query> ] ";"
<instant value> ::= <date value> "," [<hour value>]
<date value> ::= <number> "/" <number> "/" <number> <number>
```

```
<hour value> ::= <number> ":" <number>
<number> ::= <digit><digit>
<identifier> ::= <letter> { <letter> | <digit> |"_"}*
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter> :: =  A | B | C | D | E| F | G | H | I |J | K | L | M | N | O | P | Q | R
             | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j
             | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
```