Evolução de Esquemas e Propagação de Mudanças usando o Modelo Temporal de Versões

Renata de Matos Galante [1,2]
Nina Edelweiss [1]
Clesio Saraiva dos Santos [1]

[1]Universidade Federal do Rio Grande do Sul - UFRGS
Instituto de Informática
Porto Alegre - Rio Grande do Sul -Brasil

[2]Universidade de Caxias do Sul - UFRGS
Centro de Ciências Exatas e Tecnologia
Departamento de Informática
Caxias do Sul - Rio Grande do Sul - Brasil

e-mail: {galante, nina, clesio}@inf.ufrgs.br

Abstract

In this paper, we propose a schema versioning mechanism to manage the dynamic schema evolution in temporal object-oriented database. The schema evolution management uses an object-oriented data model that supports temporal features and versions definition - the Temporal Versions Model - TVM. One interesting feature of our proposal is that TVM is used to control not only the schema versioning, but also the storage of extensional database and propagation of the changes performed on the objects. The extensional data level supports integration with the existing database, allowing the maintenance of conventional and temporal versioned objects. The instance propagation approach is proposed through the specification of propagation and conversion functions. These functions assure the correct instance propagation and allow the user to handle all instances consistently in both backward and forward schema versions. Finally, the initial requirements concerning data management in the temporal versioning environment, during schema evolution, are presented.

Keywords: Temporal object-oriented database, schema evolution, schema versioning, transaction processing.

Resumo

Este artigo propõe um mecanismo de versionamento de esquemas para gerenciar a evolução dinâmica de esquemas em bancos de dados temporais orientados a objetos. O gerenciamento da evolução de esquemas tem como base um modelo dados orientados a objetos que suporta características temporais e definição de versões - *Temporal Version Model* -TVM. Uma característica interessante é o uso do TVM no controle não somente do versionamento de esquemas, mas também no armazenamento das instâncias e na propagação das mudanças nos objetos. Além disso, a extensão de dados pode armazenar tanto objetos convencionais quanto temporais versionados. O método de propagação das mudanças de esquemas nas instâncias é proposto através da especificação de funções de propagação e de funções de conversão. Essas funções garantem que as instâncias 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. Por fim, são identificados os requisitos iniciais para o processamento de transações durante a evolução de esquemas.

Palavras-chave: Bancos de dados temporais orientados a objetos, evolução de esquemas, versionamento de esquemas, processamento de transações.

1 Introdução

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 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 e versionamento de esquemas são duas técnicas utilizadas para realizar modificações na estrutura do banco de dados [8], 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.

O suporte ao versionamento de esquemas com características temporais tem sido estudado extensivamente no âmbito de bancos de dados relacionais [2, 17, 1, 11]. 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.

A maioria das propostas presentes na literatura tratam o versionamento de esquemas através de características temporais [2, 10, 17] ou pelo controle de versões [9, 16]. A principal contribuição deste trabalho é definir um mecanismo de evolução de esquemas que integre essas duas abordagens.

O objetivo deste trabalho é apresentar um mecanismo de evolução de esquemas e estratégias de propagação nas instâncias, tendo como base um modelo de dados orientados a objetos que suporta características temporais e definição de versões (TVM - *Temporal Version Model* [12]). Além disso, são identificados os requisitos iniciais para o processamento de transações durante a evolução de esquemas.

As demais seções estão organizadas da seguinte forma. A seção 2 apresenta o Modelo TVM, estabelecendo as premissas para a realização desse trabalho. A seção 3 apresenta uma arquitetura para o gerenciamento dinâmico da evolução de esquemas usando o TVM. Na seção 4 são especificados os mecanismos para versionamento temporal de esquemas. A seção 5 descreve o mecanismo de gerenciamento temporal de versões perante a modificação de esquemas. O processo de propagação das mudanças nas instâncias é exposto na seção 6. Os requisitos iniciais para o processamento de transações durante a evolução de esquemas são identificados na seção 7. A seção 8 apresenta um estudo comparativo dos trabalhos relacionados ao tema abordado. As conclusões e as perspectivas futuras são mostradas na seção 9.

2 Base Conceitual - Modelo Temporal de Versões

Como base conceitual, essa seção apresenta um resumo das principais características do *Modelo Temporal de Versões* (TVM). O TVM [12] baseia-se nos conceitos de versão e tempo para armazenar as versões de um objeto, os seus tempos de vida e o histórico das alterações feitas nos valores dos atributos e relacionamentos dinâmicos. Este modelo é uma extensão ao Modelo de Versões proposto por Golendziner [7], que estende a um modelo de dados orientados a objetos, conceitos e mecanismos que suportam a definição e manipulação de objetos, versões e configurações.

O tempo é associado ao objeto, à versão, aos atributos e aos relacionamentos. Um mesmo objeto apresenta uma linha de tempo para cada uma das versões. Desse modo, duas diferentes ordens temporais são utilizadas: (i) tempo ramificado para um objeto, devido às diferentes linhas de tempo de cada uma de suas versões, e (ii) tempo linear para as versões. O tempo varia de forma discreta, e a temporalidade é representada no modelo através do rótulo de tempo intervalar, bitemporal (tempos de validade e transação) e implícito.

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.

A figura 1 ilustra a hierarquia de classes do Modelo Temporal de Versões, possibilitando a definição de classes de aplicação não temporais e não versionáveis, bem como versionáveis e temporais. A classe `Objeto Temporal` 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.

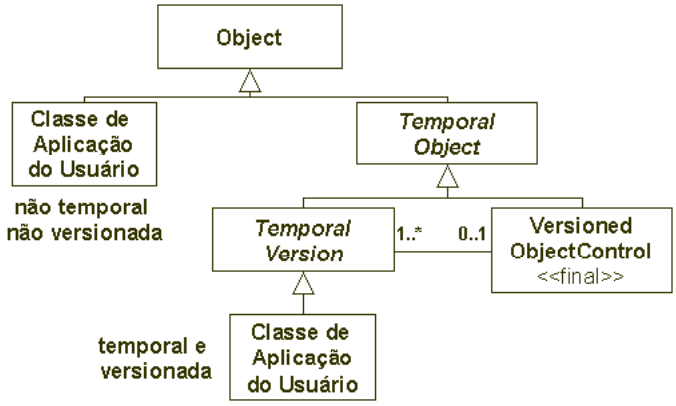
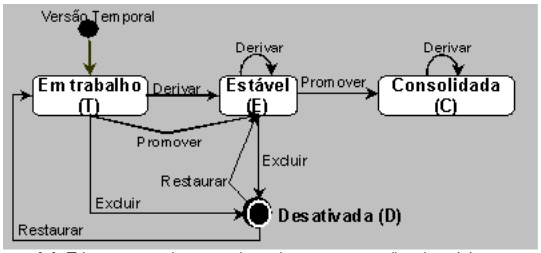


Figure 1: Hierarquia de classes do TVM e classes de aplicação

As versões podem passar por determinados *status* a fim de refletir seu estágio de desenvolvimento e/ou consistência e para que operações possam ser definidas sobre elas. O diagrama de estados (figura 2-a) ilustra as transições entre um *status* e outro, bem como os eventos que ocasionam tais transições. A figura 2-b apresenta as operações aplicáveis às versões de esquemas em cada um dos estágios de desenvolvimento.



(a) Diagrama de estados de uma versão de objeto

Status Operações	T	E	C	D
Derivar	S	S	S	-
Promover	S	S	-	-
Modificar	S	N	N	N
Excluir	S	S	N	N
Consultar	S	S	S	S

S permitido **N** não permitido
- não definido

(b) *Status* e operações

Figure 2: Diagrama de estados de uma versão de objeto

Uma versão que está no *status* em trabalho pode servir de base para derivação, ser alterada, consultada ou ainda excluída logicamente. No *status* estável, pode servir como base de uma derivação, ser consultada, excluída logicamente desde que não possua nenhuma versão sucessora, e não pode ser alterada. No *status* consolidada pode ser consultada, não pode ser alterada nem excluída logicamente. No *status* desativada pode apenas ser consultada.

3 Gerenciamento Dinâmico da Evolução de Esquemas usando TVM

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. Esse seção apresenta as premissas para o mecanismo para o gerenciamento temporal de evolução de esquemas, tendo como base o TVM.

A figura 3 ilustra a arquitetura proposta para o trabalho, estando dividida em quatro módulos:

- versionamento temporal - representa o núcleo do processo de versionamento de esquemas, implementando as primitivas do gerenciamento temporal de versões;
- gerenciador de esquemas - responsável pela evolução e gerenciamento dos esquemas, repercutindo as modificações e garantindo a validade estrutural e comportamental dos esquemas;

- gerenciador de propagação - interage com o gerenciador de esquemas, propagando os efeitos da evolução através da adaptação da base de dados, garantindo a consistência dos dados com relação aos seus esquemas associados;
- gerenciador de transações - semântica de atualização de objetos (`insert`, `update` e `delete`) de forma a garantir o sincronismo entre a disponibilidade de acesso aos dados e as propagações de mudança nos níveis de esquemas e dados.

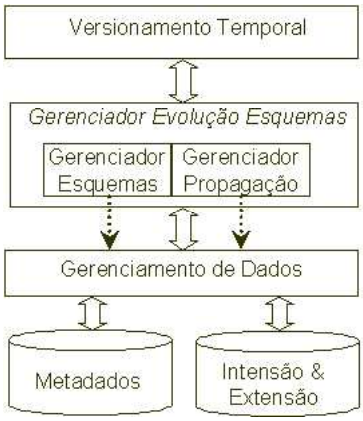


Figure 3: Arquitetura para o mecanismo de evolução dinâmica de esquemas

O núcleo fundamental do trabalho é a incorporação do TVM tanto no nível dos esquemas quanto no nível das instâncias. O TVM foi escolhido por modelar de forma completa e concisa todos os requisitos exigidos pela evolução de esquemas.

Os detalhes da arquitetura proposta são apresentados nas seções seguintes.

4 Versionamento Temporal de Esquemas

Durante o desenvolvimento de um projeto de esquema, a evolução pode ser 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 é esperado.

As versões de um esquema versionado estão relacionadas através de um relacionamento de derivação, formando um Grafo Acíclico Dirigido [7], no qual uma versão pode ser derivada de uma ou de várias existentes.

A figura 4 apresenta graficamente um esquema sem versões (`esquema simples`) e um `esquema versionado` com suas respectivas versões (`SV1, SV2, SV3, SV4`).

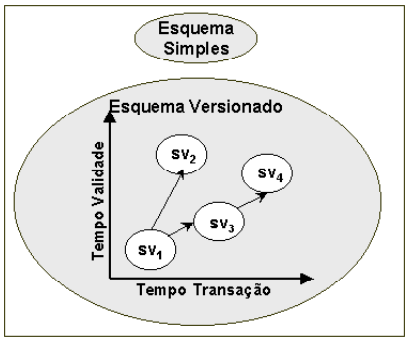


Figure 4: Esquemas versionados e não versionados

O tempo é associado tanto ao esquema versionado quanto as suas versões. Recursivamente, os elementos de cada versão do esquema (classes, atributos, métodos e relacionamentos) mantém informações temporais

a fim de representar sua evolução. Como diversos esquemas podem evoluir em paralelo, são definidas duas ordens de tempo: ramificado para o esquema versionado e linear para cada uma das versões. A temporalidade é representada através de elementos bitemporais (conjunto de intervalos temporais, tempo de transação e tempo de validade).

4.1 *Status* das Versões de Esquema

De forma similar às versões de objetos, as versões de esquemas passam por *status* que refletem seu estágio de desenvolvimento e/ou consistência. As transições entre os *status* e os eventos que causam tais transições são ilustrados na figura 5.

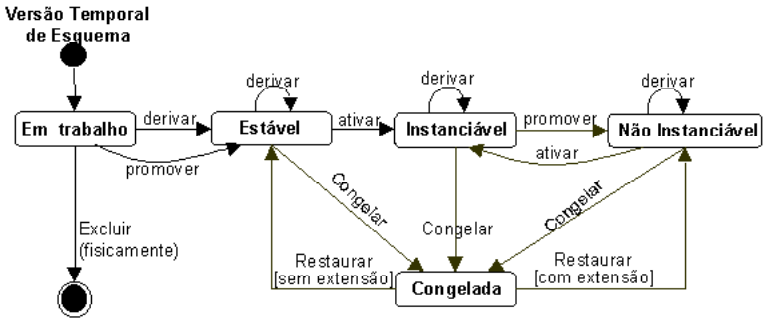


Figure 5: Diagrama de estados de uma versão de esquema

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

- em trabalho - representa um esquema que está em fase de criação. Toda modificação causa uma correção, não sendo mantido o histórico da evolução. As versões podem servir como base de derivação. Nesse caso, a versão base é promovida para o *status* estável, e a nova versão é criada no *status* em trabalho. As versões podem ainda ser promovidas para estável, consultadas e modificadas. A exclusão de um esquema em trabalho é sempre física, pois as versões não possuem uma extensão de dados associada a elas;

- 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. Neste caso, as versões podem servir como base de derivação, ser consultadas, congeladas (exclusão lógica, somente para as versões folha), ou ainda ativadas (tornadas instanciáveis);

- instanciável - é o único *status* em que o esquema pode ser instanciado (instâncias em operação). As versões podem ser derivadas, promovidas para não instanciável, consultadas ou ainda congeladas;

- não instanciável - as versões possuem uma extensão associada a elas, porém novas instanciações não são permitidas (extensão estável). As versões podem servir como base de derivação, ser consultadas, ativadas ou ainda congeladas;

- congelada - corresponde a uma exclusão lógica, na qual a versão pode ser somente consultada ou restaurada. Esta restauração se dará para o *status* não instanciável (caso exista uma extensão associada a esta versão de esquema) ou estável (quando ainda não existe extensão).

A tabela 1 apresenta a definição das operações que podem ser aplicadas sobre as versões de esquema em cada um dos *status* em que se encontram.

Table 1: Operações aplicáveis sobre as versões de esquema

Operações	*Status*	Descrição
`Derivar`	T, E, I, NI	deriva a versão a partir de outra existente.
`Promover`	T, I	promove a versão em trabalho para estável, e instanciável para não instanciável
`Ativar`	E, NI	define a versão como corrente (instanciável), associando a ela uma extensão e permitindo novas instanciações
`Congelar`	E, I, NI	exclui logicamente a versão, através do encerramento do seu tempo de validade (somente para versões folha)
`Restaurar`	C	restaura a versão para o *status* estável (quando não possui extensão) ou para não instanciável (quando possui extensão)
`Excluir`	T	exclui fisicamente a versão
`Modificar`	T	modifica fisicamente a versão
	E, I	qualquer modificação provoca a derivação de uma nova versão
`Consultar dados`	I, NI, C	consulta a versão e seus dados associados

T - Em trabalho, **E** - Estável, , **I** - Instanciável, **NI** - Não Instanciável, **C** -Congelada

4.2 Estrutura de Metadados

A fim de manter o histórico da evolução e versionamento de esquemas, uma estrutura de metadados é modelada através de classes que mantém informações sobre a evolução dos esquemas e de suas respectivas classes, métodos, atributos e relacionamentos. A figura 6 ilustra a especificação da estruturada dos metadados através de um diagrama de classes UML (*Unified Modeling Language*).

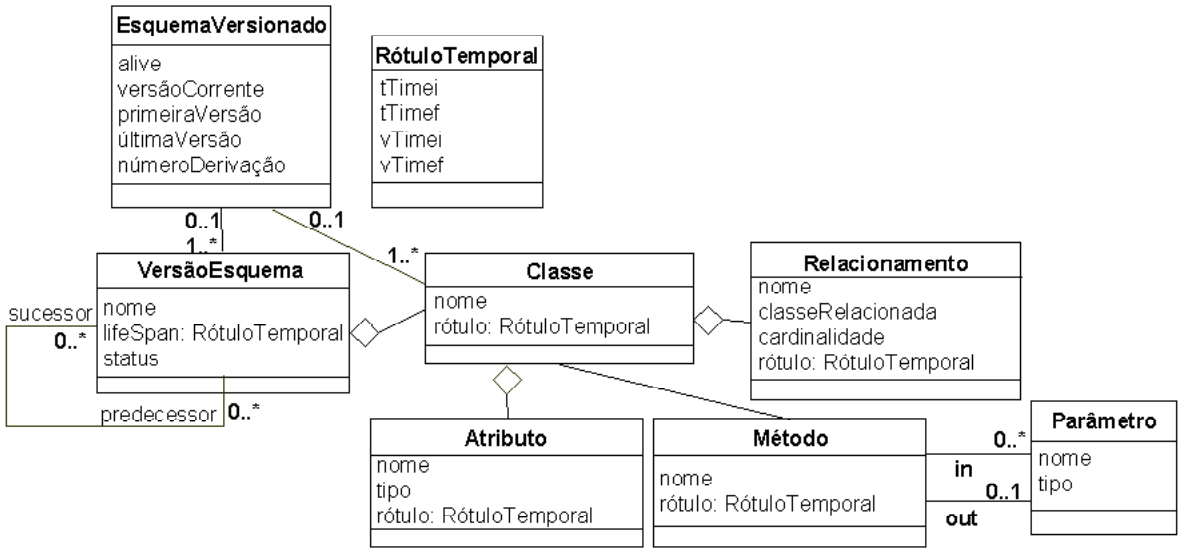


Figure 6: Estrutura dos Metadados

A estrutura dos metadados é modelada através das seguintes classes: `EsquemaVersionado`, `VersãoEsquema`, `Classe`, `Atributo`, `Método` and `Relacionamento`, servindo de base para o processo de evolução e versionamento de esquemas [1].

O tempo de vida inicial e final de cada esquema versionado é mantido através do atributo `alive`, pré-definido para a classe `EsquemaVersionado`. Para as versões de esquemas e seus atributos associados, são mantidos os seguintes atributos temporais: `vTimei`, `vTimef`, `tTimei`, `tTimef`, que representam, respectivamente, os tempos de validade inicial e final e os tempos de transação inicial e final.

[1] A especificação completa da estrutura dos metadados está publicada em [4, 3]

5 Gerenciamento Temporal de Versões para Modificação de Esquemas

A evolução do esquema conceitual e físico envolve operações sobre aspectos estruturais e comportamentais dos elementos do esquema que podem implicar em alterações nas instâncias envolvidas e em modificações nos programas aplicativos que utilizam essas instâncias. Essa relação entre instância e esquema impõe a necessidade de técnicas de evolução de esquemas e estratégias de propagação das instâncias vigentes na base de dados. O gerenciamento da modificação e versionamento de esquemas envolve quatro etapas:

- restrições de integridade - critérios para garantir que cada operação de modificação preserve a integridade das versões de esquema e de seus elementos;
- taxonomia das operações - conjunto de operações atômicas permitido para as versões de esquema e de seus elementos;
- atualização do novo esquema - mecanismo proposto para refletir as mudanças nas versões de esquema através do gerenciamento temporal de versões;
- propagação de mudanças - estratégias utilizadas para atualizar a extensão de dados de acordo com as modificações de esquema (vide seção 6).

5.1 Definição das Restrições de Integridade

A amplitude do gerenciamento da evolução de um esquema pode ser medida pelas possibilidades de transformação que oferece. O mecanismo de suporte deve assegurar que essas modificações preservem a integridade do esquema. Para tanto, é definido um conjunto de requisitos, denominados invariantes, para garantir a validade do esquema e de suas instâncias. Regras são estabelecidas para reger essa evolução, permitindo a realização de atualizações somente quando não conduzem o esquema a um estado inconsistente. São utilizados três tipos de invariantes: estruturais, comportamentais e de instanciação.

Os invariantes estruturais visam assegurar a integridade do conjunto de classes, suas descrições internas e relacionamento de generalização/especialização e agregação. Os invariantes comportamentais asseguram a integridade do conjunto de métodos declarado no esquema. Os invariantes instanciação asseguram a integridade das instâncias com relação ao esquema definido.

5.2 Definição das Operações de Modificação de Esquemas

A derivação de uma nova versão de esquemas é desencadeada pela aplicação de uma operação de modificação de esquemas. Quando uma nova operação é aplicada no esquema, os invariantes são testados e, se a alteração pretendida preserva os filtros de integridade, uma nova versão de esquema é derivada com as alterações realizadas. Para tanto, um conjunto de operações é definido. A tabela 2 apresenta a taxononima das operações de modificação de esquema definida. As operações são classificadas de acordo com o tipo de alteração que realiza no esquema: (*i*) modificações na estrutura do esquema, (*ii*) modificações na estrutura das classes, e (*iii*) modificações no comportamento dos esquema.

Table 2: Taxonomia para evolução de esquemas

Estrutura do Esquema	
Incluir/Excluir classe	Renomear classe
Mudar ordem de superclasse	Incluir Excluir superclasse
Generalizar classes	Dividir classe
Unir classes	
Estrutura das Classes	**Comportamento das Classes**
Incluir/Excluir atributo	Incluir/Excluir método
Renomear atributo	Renomear método
Mudar domínio atributo	Mudar código método
Mudar valor inicial atributo	Mudar domínio método
Incluir/Excluir agregação	Mudar contra domínio métodos

Sempre que uma operação de modificação de esquema é realizada, uma nova versão de esquema é derivada. O versionamento total é empregado pois qualquer modificação gera uma nova versão de esquema.

Para as versões de esquema são associados os tempos de transação e validade (versionamento bitemporal), permitindo mudanças retroativas e proativas. Esse tipo de versionamento permite o acesso às versões de esquema, tanto pelo histórico das transações quanto pelas suas validades.

5.3 Armazenamento da Intensão e da Extensão

Para armazenar todo histórico das modificações, uma alternativa para a implementação do gerenciamento temporal de esquemas é a utilização de versões de esquemas armazenadas em múltiplos repositórios. Nesse caso, qualquer operação de modificação acarreta a derivação de uma nova versão de esquema e, conseqüentemente, um novo repositório de dados é criado. Quando uma nova versão de esquema é derivada, as instâncias são adaptadas ao novo esquema e copiadas para o novo repositório.

Por exemplo, a figura 7-(a) ilustra a operação de incluir a classe `Disciplina`, na primeira versão do esquema (`Esquema,1`). Por conseguinte, uma nova versão de esquema é derivada (`Esquema,2`). Para cada versão de esquema, um novo repositório de dados é criado (`repositório 1` e `repositório 2`), armazenando as modificações realizadas.

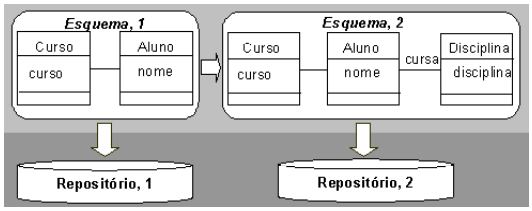


Figure 7: Armazenamento em múltiplos repositórios

Implicitamente, o mecanismo de gerenciamento da evolução de esquemas realiza mapeamento que atualizam a estrutura dos metadados (por exemplo, figura 7-(b)) com as informações sobre a modificação realizada. Assim, cada versão de esquema possui uma instância na classe `VersãoEsquema` definida na figura 6.

5.3.1 Alternativa de Implementação: Repositório Único

Como o simples processo de derivação de esquemas e a criação de repositórios de dados podem implicar na excessiva proliferação de versões, uma outra alternativa para implementação é definida, visando melhorar o desempenho do sistema, evitando interrupções freqüentes e garantindo a integridade das informações armazenadas. Para tanto, propõe-se o armazenamento em único repositório somente para as operações que alteram a estrutura das classes. Nesse caso, todo o histórico da evolução é igualmente mantido.

A figura 8-(a) ilustra a operação de inclusão do atributo `endereço` na classe `Aluno`. Conseqüentemente, uma nova versão de esquema é derivada, porém os dados associados as duas versões de esquemas são armazenados no mesmo repositório.

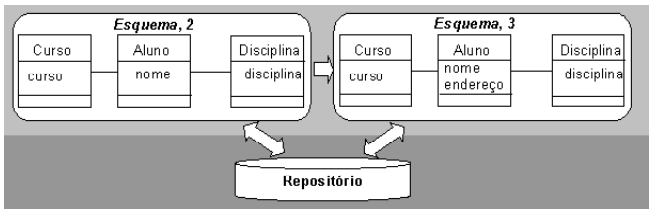


Figure 8: Armazenamento em único repositório

A estrutura dos metadados (vide figura 6) deve ser atualizada para refletir as modificações realizadas. Cada versão de esquema deve armazenar informações temporais a respeito de seus componentes, atualizando as classes `Classe`, `Atributo`, `Método` e `Relacionamento`.

No nível de implementação física, os vários repositórios podem compartilhar estruturas, sendo evitada a replicação de dados não afetados pela evolução de esquemas.

6 Propagação da 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. Esse problema pode ser resolvido através dos seguintes pontos: (*i*) identificar as instâncias que precisam ser modificadas, e (*ii*) transformar as instâncias de forma que representem as modificações de esquema.

Para a propagação das mudanças de esquema nas instâncias é proposto um mecanismo híbrido que contempla as técnicas de conversão imediata e conversão adiada. Conforme ilustrado na figura 9, cada versão de esquema tem associado um conjunto de instâncias, definindo seu escopo de acesso. Uma modificação de esquema implica na atualização imediata dos objetos que estão sendo utilizados pelas aplicações (em [5] são definidas regras que regem a escolha das instâncias importantes às 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.

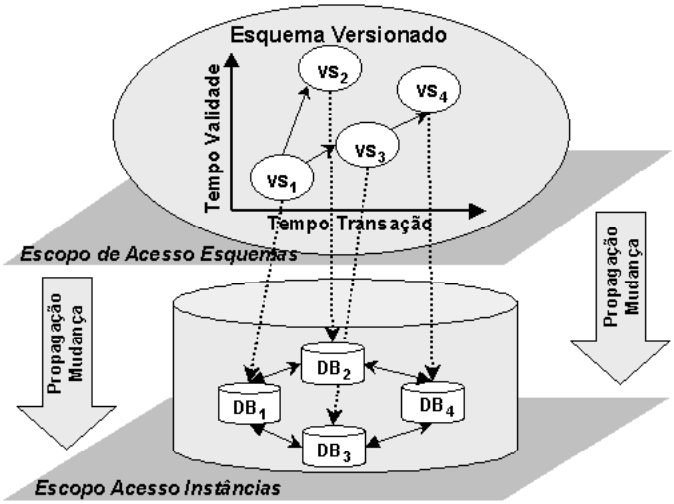


Figure 9: Escopo de acesso às instâncias

Em um nível de implementação, funções devem ser especificadas para 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.

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 classe modificada 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 de esquema para outra.

6.1 Funções de Propagação

As funções de propagação envolvem a implementação de duas funções: *forward* e *backward*. A primeira descreve o comportamento das instâncias, quando solicitadas pela nova versão do esquema. Inversamente, a segunda descreve o comportamento das instâncias, quando solicitadas pela versão de esquema antecessora. Nesse caso, as funções são reversíveis, pois as instâncias podem ser livremente convertidas entre as diversas versões de esquema existentes.

A figura 10 mostra o algoritmo que especifica as funções de propagação para o exemplo apresentado anteriormente na figura 7, no qual a classe `Disciplina` é incluída na nova versão do esquema. A função *forward* define as seguintes operações envolvidas na mudança: (*i*) criar o objeto com seus valores, e (*ii*) especificar o relacionamento com a classe `Aluno`. Inversamente, a função *backward* exclui o objeto disciplina e o seu relacionamento com a classe `Aluno`.

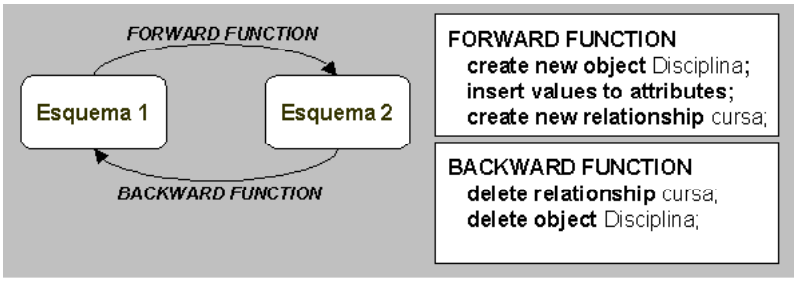


Figure 10: Funções de propagação

6.2 Funções de Conversão

As funções de conversão implicam na implementação de duas funções: *forward* e *backward*. As funções *forward* são associadas à classe em que a atualização foi realizada e descrevem as características dos objetos quando solicitados pela próxima versão, isto é, implementam operações que adaptam os objetos segundo a evolução de esquema. As funções *backward* são associadas à nova versão e descrevem o comportamento dos objetos na versão anterior, ou seja, antes de realizadas as modificações.

A figura 11 mostra a especificação das funções de conversão aplicadas para o exemplo apresentado na figura 8, no qual o atributo `endereço` é incluído na classe `Aluno`. Na função *forward* o atributo `endereço` é incluído, sendo atribuído o valor *null*. Inversamente, na função *backward* o atributo `endereço` deve ser excluído.

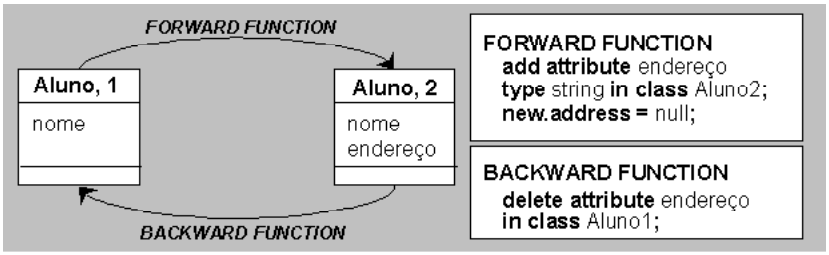


Figure 11: Funções de Conversão

7 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 objetos. 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. Um modelo de transações deve ser especificado para permitir a execução concorrente de operações de modificação de esquemas e manipulação de instâncias. Como as operações podem estar intercaladas em uma transação, o mecanismo deve prover estratégias para garantir a integridade da pertinência temporal entre esquemas e dados.

Como nosso trabalho nesta área é relativamente recente, resultados mais concretos serão apresentados em artigos futuros.

8 Trabalhos Relacionados

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 resolver adequadamente o

problema. Considerando a temporalidade e o versionamento de esquemas, alguns trabalhos têm explorado esses conceitos no âmbito de sistemas de bancos de dados relacionais [2, 17, 1, 11], e outros têm incorporado esses conceitos em bancos de dados orientados a objetos [6, 9, 10, 14, 15, 16].

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. Gançarski et. al [6] propõem um modelo formal com versionamento temporal de esquemas e objetos. Tigukat [14, 15] é um modelo de objetos com características temporais que inclui um modelo de consulta e políticas para evolução dinâmica de esquemas e controle de versões. Mandreoli [10] apresenta uma descrição formal para o processo de modificação e evolução de esquemas em bancos de dados orientados a objetos, incluindo um conjunto completo de primitivas de mudança de esquemas. A proposta descrita em [9] apresenta um método de versionamento de esquema que suporta mudanças dinâmicas de esquemas em bancos de dados orientados a objetos durante a vigência das aplicações, com ênfase na definição dos critérios de atualização de instâncias e integração de versões de esquemas. O Sades Evolution (*Semi-Autonomous Database Evolution System*) [16] propõe um método flexível de adaptação das instâncias para sistemas que empregam o versionamento de classes para gerenciar a evolução de esquemas.

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 [2] são propostas duas soluções para o armazenamento dos dados da extensão: (i) repositório único, no qual os dados correspondentes a todas as versões de esquema são mantidos segundo um esquema global, e (*ii*) múltiplos repositórios, no qual distintos repositórios são mantidos para cada versão de esquema. O PMTV (*Partial Multiple Table Versioning*) [17] propõe um método híbrido para o armazenamento da atualização de esquemas em bancos de dados bitemporais relacionais. São definidos algoritmos para atualização do esquema e conversão dos dados associados, considerando as operações de inclusão e exclusão de atributos, tanto para o armazenamento em repositório único como em múltiplos repositórios.

Com a análise da literatura, verifica-se a necessidade de um modelo completo, que trate o versionamento temporal de esquemas em bancos de dados temporais orientados a objetos, abrangendo os diversos aspectos relacionados: operações de modificação de esquema, regras de integridade para garantir a consistência entre versões de esquema e suas instâncias associadas, tratamento dos diversos *status* pelos quais uma versão de esquema pode passar durante o seu ciclo de vida, operações para gerenciar a derivação de versões, emprego de versões alternativas e a utilização de tempo de transação e de tempo de validade.

9 Conclusões e Trabalhos Futuros

Este artigo apresenta um mecanismo de evolução de esquemas e estratégias de propagação nas instâncias, tendo como base o TVM. A união dos conceitos de versão e tempo permite o armazenamento de diversas versões de esquema, como também, para cada versão, o histórico da sua modificação. Adicionalmente, a utilização de bancos de dados bitemporais provê flexibilidade ao mecanismo de evolução de esquemas, permitindo o acesso a informações passadas e futuras.

O método de propagação das mudanças de esquemas nas instâncias é proposto através da especificação de funções de propagação e de funções de conversão. Essas funções garantem que as instâncias 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 de esquema para outra. Além disso, a extensão de dados pode armazenar tanto objetos convencionais quanto temporais versionados.

O TVM foi selecionado por apresentar características peculiares que auxiliam no processo de evolução de esquemas, permitindo uma modelagem mais natural e concisa da realidade. Sua utlização mostrou-se adequada à resolução desse problema, conduzindo a uma maior completude o mecanismo de evolução de esquemas proposto.

No âmbito do TVM, o diferencial deste trabalho fundamenta-se na definição clara e precisa dos *status* pelos quais as versões de esquema podem passar, assim como sua aplicação através da proposição de operações sobre as versões de esquema.

Atualmente, está sendo desenvolvida uma extensão para o mecanismo de evolução de esquemas apresentado. Esta extensão pode ser separada em duas partes. Primeira, um mecanismo de gerenciamento de transações está sendo definido para o ambiente dinâmico de evolução de esquemas. Esse gerenciamento é importante para garantir a execução concorrente das operações de versionamento e modificação de esquemas, bem como a atualização dos dados associados, garantindo um estado consistente ao banco de dados. Segunda, uma linguagem de definição e manipulação de objetos está sendo especificada para prover a correção sintática e semântica do processo da evolução de esquemas e manipulação de objetos. Essa linguagem é uma extensão da linguagem TVQL (*Temporal Versioned Query Language*) [13], definida para o TVM. A

TVQL é baseada em SQL (*Structured Query Language*), adicionando novas características para recuperação de informações temporais e versões.

Uma vez definidos os requisitos necessários para a evolução de esquemas com a utilização do TVM, está em desenvolvimento seu mapeamento para um banco de dados convencional. O objetivo é implementar um protótipo para simulação da viabilidade do mecanismo proposto.

References

[1] S. F. Angonese and N. Edelweiss. Gerenciador Temporal de Versões de Esquema. In *Conferência Latino Americana de Informática*, Merida, Venezuela, 2001.

[2] C. de Castro, F. Grandi, and M. R. Scalas. Schema Versioning for Multitemporal Relational Databases. *Information Systems*, 22(5):249–290, 1997.

[3] R. M. Galante, N. Edelweiss, and C. S. dos Santos. Change Management for a Temporal Versioned Object-Oriented Database. In *Second International Workshop on Evolution and Change in Data Management*, Tampere, Finland, october 2002. Springer-Verlag. (Held in conjunction with the 21st International Conference on Conceptual Modelling).

[4] R. M. Galante, A. B. S. Roma, A. Jantsch, N. Edelweiss, and C. S. dos Santos. Dynamic Schema Evolution Management using Version in Temporal Object-Oriented Database. In *International Conference on Database and Expert Systems Applications*, Aix-en-Provence, France, september 2002. Springer-Verlag.

[5] R. M. Galante, C. S. Santos, and D. D. A. Ruiz. 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*, pages 303–318, Maringá, PR, Brasil, Outubro 1998. Maringá: UEM-DIN.

[6] S. Gançarski and G. Jomier. A framework for programming multiversion databases. *Data & Knowledge Engineering*, 36(1):29–53, january 2001.

[7] L. G. Golendziner and C. S. Santos. Version and configurations in object-oriented database systems: a uniform treatment. In *International Conference on Management Data*, pages 18–37, Pune, India, december 1995. New Delhi : Tata Mcgraw-Hill.

[8] C . S. Jensen, C. Dyreson, M. Böhlen, J. Clifford, R. Elmasri, S. K. Gadia, and F. Grandi. *Temporal Databases: Research and Practice*, chapter A Consensus Glossary of Temporal Database Concepts - February 1998 Version, pages 367–405. Springer-Verlag, Heidelberg, 1998.

[9] S. Lautemann. Schema Versions in Object-Oriented Database Systems. In *International Conference on Database Systems for Advanced Applications*, pages 323–332, Melbourne, Australia, 1997.

[10] F. Mandreoli. *Schema Versioning in Object-Oriented Databases.* PhD thesis, Universitá Degli Studi di Bologna, january 2001.

[11] V. P. Moreira and N. Edelweiss. Schema versioning: queries to a Generalized Temporal Databases System. In *Workshop on Spatio-Temporal Data Models and Languages*, 1999. in conjuntion with DEXA.

[12] M. M. Moro, S. M. Saggiorato, N. Edelweiss, and C. S. Santos. Adding Time to an Object-Oriented Versions Model. In *International Conference on Database and Expert Systems Applications*, pages 805–814, Munich, Germany, September 2001.

[13] M. M. Moro, A. P. Zaupa, N. Edelweiss, and C. S. dos Santos. TVQL - Temporal Versioned Query Language. In *International Conference on Database and Expert Systems Applications*, Aix-en-Provence, France, september 2002. Proceedings.

[14] M. T. Ozsu, R. J. Peters, D. Szafron, B. Irani, A. Lipka, and A. Muñoz. TIGUKAT: A Uniform Behavioral Objectbase Management System. *The VLDB Journal - Special Issues on Persistent Object Systems*, 4(3):445–492, july 1995.

[15] R. J. Peters and M. T Oszu. Axiomatization of Dynamic Schama Evolution in Objectbases. In *International Conference on Data Engineering*, Taiwan, march 1995.

[16] A. Rashid, P. Sawyer, and E. Pulvermüller. A Flexible Approach for Instance Adaptation during Class Versioning. In *International Symposium Objects and Databases*, volume 1944 of *Lecture Notes in Computer Science*, pages 101–113, Sophia Antipolis, France, 2001. Springer.

[17] H. Wei and H. Elmasri. PMTV: A Schema Versioning Approach for Bi-temporal Databases. In *International Workshop on Temporal Representation and Reasoning*, 2000.