Evolução de Esquemas Utilizando Versões em Bancos de Dados Orientados a Objetos*

Renata de Matos Galante
Universidade de Caxias do Sul – UCS
Centro de Ciências Exatas e Tecnologia – CCET
Departamento de Informática – DEIN
Caxias do Sul - RS – Brasil
E-mail: galante@inf.ufrgs.br

Clesio Saraiva dos Santos
Universidade Federal do Rio Grande do Sul - UFRGS
Instituto de Informática
Porto Alegre - RS – Brasil
E-mail: clesio@inf.ufrgs.br

Duncan Dubugras Alcoba Ruiz
Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS
Faculdade de Informática
Porto Alegre – RS – Brasil
E-mail: duncan@inf.pucrs.br

Abstract

In this paper, we propose a flexible model to support schema evolution in object-oriented databases, as well as the strategies to propagate the corresponding changes to the database instances. Versions of the schema, as well as version of the schema elements represent the history of modifications. In this model, previous states are preserved allowing the user to make queries about consistency and modifications in both backward and forward versions. The paper also presents the main characteristics of the version model adopted [1] and discusses a case study about a real system in execution that allows the validation of the model proposed and identification of requirements to be fulfilled by the schema evolution mechanisms.

***Keywords**: object-oriented database, version, and schema evolution.*

1. Introdução

Um dos mais importantes problemas na construção e manutenção de aplicações envolvendo bancos de dados são as inevitáveis mudanças impostas ao longo do tempo. As alterações podem ocorrer em qualquer etapa do ciclo de vida de um sistema, tanto durante a fase de projeto, quando o esquema raramente permanece estável ou completamente definido, quanto durante a fase de operação do sistema, em resposta à evolução da realidade considerada.

A necessidade de alteração do esquema conceitual tem sido identificada, em muitos domínios de aplicações, principalmente, emergindo em sistemas de aplicações não convencionais, como, por exemplo, sistemas de automação de escritório, projetos de engenharia e inteligência artificial, dentre outros.

Apesar de reconhecida a importância dos mecanismos de evolução de esquemas em sistemas de bancos de dados e, mais recentemente, várias pesquisas [2,3,4,5,6,7] terem expandido e explorado tais funcionalidades, ainda não existe um consenso entre as propostas apresentadas, nem um modelo completamente definido ou totalmente implementado. É reconhecida a importância da definição de mecanismos elaborados, com amplo suporte às necessidades de evolução impostas pelas atuais aplicações de banco de dados.

O tratamento de evolução de esquemas permite o desenvolvimento de esquemas conceituais onde refinamentos são introduzidos gradualmente, sem causar danos às aplicações em andamento. Para tanto, mecanismos de versões são utilizados para representar o histórico das modificações em esquemas, classes, métodos e objetos, garantindo a manipulação do esquema em qualquer perspectiva de versão.

Inúmeros trabalhos têm adotado o conceito de versão no tratamento da evolução de esquemas em bancos de dados orientados a objetos, sem que haja, no entanto, um consenso entre eles. Versões de esquemas e instâncias são utilizadas em [2] e [3] a fim de representar o estado evolucionário do esquema. Em [4] o conceito de versão é adotado para esquemas, classes e instâncias. Em [5] as unidades alteráveis são contextos, compostos por uma seleção de classes, previamente definida no esquema, e apenas a essas classes é permitida a característica de possuir versões. Em [6] o versionamento é utilizado para esquemas e classes: versões de

* Trabalho realizado com suporte financeiro da CAPES.

esquemas são criadas quando modificações são realizadas na estrutura das classes e versões de classes, por mudanças efetuadas em suas propriedades e/ou relacionamentos de superclasse. Em [7] instâncias, classes e métodos são tratados de maneira uniforme, sendo todos considerados objetos versionáveis.[1]

O escopo deste trabalho refere-se a proposta de um Modelo de Evolução de Esquemas [8,9] onde versões são utilizadas para armazenar os diferentes estados do esquema, de suas classes e métodos e, ainda, para posterior adaptação das instâncias vigentes no banco de dados, permitindo, com isso, a recuperação dos objetos nas diversas perspectivas sob as quais são solicitados.

O Modelo de Versões adotado [1] 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. Assim, uma versão descreve um objeto em um determinado período de tempo ou sob um certo ponto de vista, cujo registro é relevante para a aplicação considerada.

As demais seções estão organizadas da seguinte maneira: um estudo de caso é mostrado na seção 2, considerando um sistema real em execução. A seção 3 apresenta o modelo de dados e os conceitos básicos relacionados a versões. Na seção 4, o Modelo de Evolução de Esquemas proposto é apresentado e a seção 5 apresenta as conclusões, apontado tendências futuras em mecanismos de evolução de esquemas.

2. Análise de uma Situação Real

Com o intuito de identificar requisitos e funcionalidades necessárias a um modelo de evolução de esquemas foi desenvolvido um estudo de caso, considerando a evolução do Sistema de Controle Acadêmico da Universidade Federal do Rio Grande do Sul, UFRGS, denominado Sistema Discente. Dentro desse contexto, buscou-se identificar padrões de alteração de esquemas introduzidos no sistema, bem como modificações que não foram incorporadas, embora julgadas importantes no âmbito de um modelo de evolução de esquemas.

O Sistema Discente [11,12,13] foi desenvolvido pelo Centro de Processamento de Dados da UFRGS. Sua implementação teve início no ano de 1972 e, desde então, mantém o registro eletrônico de todos os alunos que tiveram vínculo com a universidade, cursos e respectivos currículos e, a partir de 1994, as turmas oferecidas semestralmente. Inicialmente, implementado em Cobol e usando Sistema Gerenciador de Banco de Dados Hierárquico, DMS-II Unisys, divide-se em três principais subsistemas: programação de currículos, programação de turmas e controle de alunos.

O Sistema de Controle Acadêmico pode ser divido em quatro fases distintas, caracterizando sua evolução ao longo do tempo:

- FASE 1 (1973): o sistema teve início com um esquema de banco de dados hierárquico, cuja implementação foi desenvolvida na linguagem Cobol, em um banco de dados DMS II, sob o sistema Burroughs, com modelo de máquina B-6700;
- FASE 2 (1987): apenas uma parte do Sistema Discente, referente a matrícula, foi desenvolvida. A implementação foi realizada na linguagem Linc II, em um banco de dados DMS II, sob o sistema Unisys, com modelo de máquina A10;
- FASE 3 (1995): a implementação foi desenvolvida na linguagem Linc II, em um banco de dados DMS II, sob o sistema Unisys, com modelo de máquina A10/A14, procurando atender as demandas de um modelo de banco de dados relacional;
- FASE 4 (1996/1997): a implementação foi desenvolvida na linguagem PowerBuilder, em um banco de dados Sybase, sob o sistema operacional Unix, caracterizando-se por um banco de dados expressamente relacional.

Cada fase de evolução caracterizou-se pela definição de um novo esquema ao banco de dados, bem como pela troca de um sistema implementado por outro. Observa-se uma evolução gradual na passagem de um sistema de banco de dados hierárquico para uma modelagem relacional.

[1] Um levantamento bibliográfico mais detalhado está documentado em [10].

A análise do Sistema Discente permitiu a identificação de padrões de mudanças, alguns deles não previstos na bibliografia consultada [2,3,4,5,6,7]:

- alterações em entidades: criação e exclusão de entidades;
- alterações em relacionamentos: inclusão e exclusão, mudança entre entidades existentes e inclusão de auto-relacionamentos;
- alterações em atributos: modificação em domínio e junção de dois atributos em um.

Além dos padrões citados anteriormente, facilidades que podem ser incorporadas a um modelo de evolução de esquemas foram também identificadas, tais como:

- necessidade de mecanismos que auxiliem o projetista no conhecimento da realidade da aplicação, bem como em todo seu estado evolucionário, a fim de garantir maior segurança e confiabilidade na definição das etapas de evolução;
- mecanismos de aproveitamento e validação das informações presentes no banco de dados; assim, informações existentes que perduram após a evolução do esquema são atualizadas seguindo a nova definição, implicando, por exemplo, na reformatação do banco de dados;
- especificações de mecanismos de evolução de esquema capazes de guiar as alterações, assegurando a validade da semântica especificada para o sistema, bem como a consistência da base de dados e sua coerente correspondência com o esquema definido.

Esse estudo de caso permitiu a identificação de padrões de mudanças e facilidades de evolução de esquemas a serem suportadas por bancos de dados orientados a objetos, proporcionadas, principalmente, pela possibilidade do estudo de problemas significativos baseado em dimensões reais, contribuindo assim, para garantir a aplicabilidade e adequação da proposta definida.

3. O Modelo de Dados e o Modelo de Versões

Nesta seção são apresentados os conceitos relativos ao modelo de dados e de versões, com vistas às necessidades impostas pelo Modelo de Evolução de Esquemas proposto.

3.1. O Modelo de Dados

O Modelo de Dados está baseado nos princípios fundamentais do paradigma de orientação a objetos em bancos de dados, definidos para a maioria das propostas e sistemas presentes na literatura, divulgados no Manifesto Atkinson [14]. O Modelo proposto permite representar os conceitos de orientação a objetos, como: identidade de objetos, objetos simples e complexos, tipos e classes, relacionamentos de herança simples e múltipla e encapsulamento, entre outros.

3.2. O Modelo de Versões

O Modelo de Versões adotado [1] estende a um modelo de dados orientado a objetos conceitos e mecanismos que suportam a definição e manipulação de objetos, versões e configurações.

Uma das características relevantes é a presença do mecanismo de herança por extensão, onde as versões são admitidas nos vários níveis da hierarquia de herança. O mecanismo de herança por extensão está associado à idéia de protótipo, pois sendo, por exemplo, C2 uma subclasse de C1, dois objetos são criados: um como instância de C1 (protótipo) associado a outro como instância de C2 (extensão do protótipo). O objeto criado como instância de C1 é denominado ascendente e a instância de C2, descendente.

A existência de versões em vários níveis de uma hierarquia de herança introduz a idéia de mapeamentos entre versões, aos quais podem ser associadas restrições de cardinalidade que disciplinam as correspondências entre versões situadas em diferentes níveis.

As versões constituem objetos de primeira classe, possuindo um identificador próprio (OID). Objetos que possuem versões são denominados objetos versionados. As versões de um objeto versionado são organizadas formando um grafo acíclico dirigido, refletindo seus relacionamentos, onde apenas a primeira versão não possui uma antecessora. Uma referência a um objeto versionado é denominada referência

dinâmica, uma vez que pode indicar qualquer uma das versões daquele objeto. O mecanismo de versão corrente permite determinar automaticamente a versão de um objeto versionado a ser utilizada. Apresenta também referências estáticas, onde uma versão específica do objeto versionado é indicada. As versões possuem ainda um estado (*status*), que reflete seu estágio de desenvolvimento, podendo ser em trabalho, estável ou consolidada.

O modelo propõe ainda o conceito de configuração, na forma de versões configuradas. Considerando um objeto composto, em que os componentes podem ser versionados, uma configuração associa exatamente uma versão para cada um dos componentes. Como o modelo apresenta herança por extensão, este 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.

4. Proposta de um Modelo de Evolução de Esquemas

Nesta seção são definidas as principais características do Modelo de Evolução de Esquemas como forma de conduzir alterações em bancos de dados orientados a objetos. As idéias apresentadas são adaptáveis a vários sistemas de banco de dados orientados a objetos, pois os mecanismos são propostos de forma genérica e não definidos para um sistema específico.

4.1. Definição de Restrições

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[2], para garantir a validade do esquema bem como das instâncias armazenadas na base de dados. Regras são estabelecidas para reger essa evolução, permitindo a realização de atualizações somente quando não conduzem o esquema a um estado inválido.

Diversas propostas, presentes na literatura, tais como [5,7,15,16], adotam invariantes como forma de garantir a integridade frente as modificações. Esse Modelo utiliza invariantes estruturais, comportamentais e de instanciação.

Os Invariantes Estruturais visam assegurar a integridade do conjunto de classes, suas descrições internas e relacionamentos de generalização e agregação, estabelecendo que:

- os relacionamentos devem resultar num grafo acíclico e conexo, com as classes ligando-se, no mínimo, à superclasse global;
- classes devem possuir nomes únicos, existindo quando referidas como domínio de atributos;
- atributos devem possuir nomes únicos em cada classe, com conflitos de herança resolvidos, permanecendo nos níveis onde foram definidos, exceto se houver redefinição na subclasse; neste caso, devem possuir domínio igual ou mais especializado ao de sua definição inicial.

Os Invariantes Comportamentais asseguram a integridade do conjunto de métodos declarado no esquema, impondo que os métodos de uma classe devem possuir nomes únicos, com conflitos de herança solucionados, permanecendo nos níveis onde foram definidos, exceto se houver redefinição nas subclasses; nesse caso, devem possuir comportamento idêntico ou mais específico ao de sua definição.

Os Invariantes de Instanciação asseguram a integridade dos objetos com relação ao esquema definido, onde:

- todos os objetos referidos devem estar presentes no banco de dados;

[2] Os invariantes estruturais e comportamentais estão definidos em [15] e os de instanciação, definidos em [17]. Algumas adaptações são realizadas, pois o modelo adota o mecanismo de herança por extensão, diferindo das atuais propostas presentes na literatura.

- os valores definidos para os atributos de uma versão de instância devem pertencer ao domínio definido para versão de classe ao qual está associado ou é igual a NULL; e
- para cada versão da classe deve haver, no mínimo, uma versão de cada instância existente no instante de tempo em que a versão da classe era válida, de acordo com a descrição dessa versão da classe.

Esses critérios foram introduzidos para garantir que cada operação de atualização preserve a integridade do esquema e a validade do banco de dados. 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

4.2. Operações de Atualização

As operações de atualização realizáveis pelo Modelo estão dividida em três blocos distintos: as modificações que alteram a estrutura do esquema (seção 4.2.1); as modificações que alteram a estrutura das classes (seção 4.2.2) e as modificações que alteram o comportamento das classes (seção 4.2.3).

A Figura 1 apresenta as operações de modificação realizáveis pelo Modelo de Evolução de Esquemas.[3]

Taxonomia de Evolução de Esquemas

1. Alterações na estrutura do Esquema
    - 1.1. Incluir uma classe
    - 1.2. Excluir uma classe
    - 1.3. Mudar o nome de uma classe
    - 1.4. Mudar a ordem de superclasse de uma classe
    - 1.5. Incluir uma superclasse
    - 1.6. Excluir uma superclasse
    - 1.7. Criar uma nova classe como superclasse generalizando outras existentes
    - 1.8. Decompor uma classe em novas classes
    - 1.9. Unir várias classes em uma nova classe
2. Alterações na estrutura das classes
    - 2.1. Incluir um atributo
    - 2.2. Excluir um atributo
    - 2.3. Mudar o nome de um atributo
    - 2.4. Mudar o domínio de um atributo
    - 2.5. Mudar o valor inicial de um atributo
    - 2.6. Incluir uma ligação de composição
    - 2.7. Excluir uma ligação de composição
3. Alterações no comportamento das classes
    - 3.1. Incluir um método
    - 3.2. Excluir um método
    - 3.3. Mudar o nome de um método
    - 3.4. Mudar o código de um método
    - 3.5. Mudar o domínio e/ou contradomínio de um método

FIGURA 1 - Modelo de Evolução de Esquema: Panorama das Modificações

A execução das operações de atualização acarretam a derivação de versões de esquemas, classes ou métodos, como a seguir:

[3] Mais detalhes podem ser encontrados em [9].

- **versões de esquema**: são derivadas em decorrência de modificações realizadas na estrutura do esquema;
- **versões de classes**: são geradas em decorrência de modificações executadas na estrutura das classes, neste caso, tanto a classe quanto os atributos devem estar presentes na mesma versão de esquema; e
- **versões de métodos**: são geradas após realizadas modificações na implementação dos métodos definidos para uma versão de classe ou alteração no domínio e/ou contradomínio de um método.

A utilização de versões para esquemas, classes, métodos permite controlar a excessiva proliferação de versões, evitando a redundância de dados. Por exemplo, a inclusão de um atributo acarreta a derivação de uma versão para a classe, e não todo para todo esquema definido.

4.2.1. Alterações na Estrutura do Esquema

A execução dessas operações provoca a **derivação de novas versões de esquema** devido às modificações realizadas na estrutura do esquema do banco de dados.

Por exemplo, a Figura 2 ilustra a generalização das classes *Professor* e *ProfessorAssistente*, na primeira versão do esquema (Esquema, 1), e a derivação de uma nova versão (Esquema, 2) contendo a generalização criada, classe *Docente*.

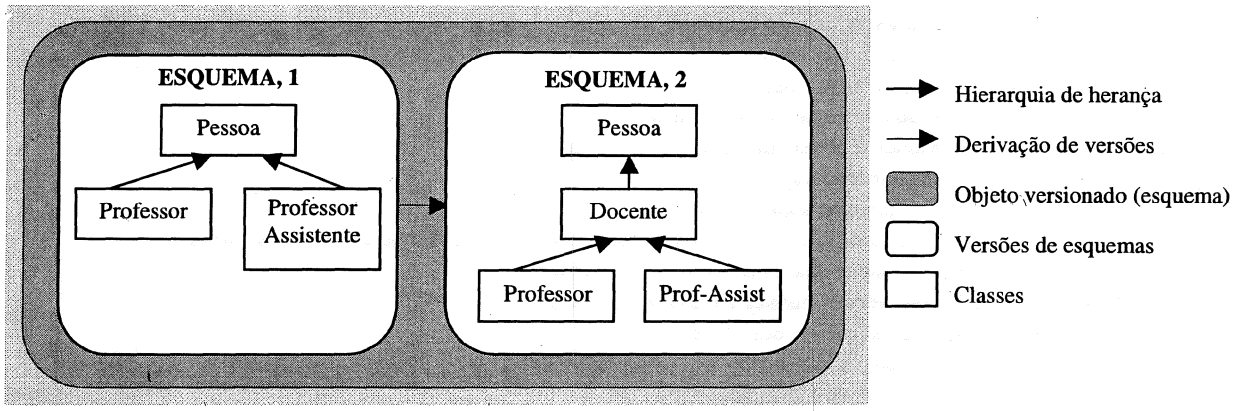

FIGURA 2 - Generalização de Classes

O procedimento padrão permite ao usuário indicar um conjunto de classes e o sistema, automaticamente, transfere as propriedades com nomes e definições comuns para a nova classe gerada. Esta nova classe deve garantir a unicidade de nomes. Senão a operação deve ser rejeitada e o usuário, notificado.

As classes selecionadas para generalização devem apresentar superclasses comuns e, nesse caso, passam a ser superclasses diretas da nova classe criada.

As propriedades que migram para a nova classe devem possuir nomes iguais, porém podem apresentar domínios diferentes. Por *default*, a propriedade pertencente à primeira classe indicada para generalização é selecionada. Entretanto, o usuário pode escolher outra propriedade, bastando indicar a classe a que pertence.

4.2.2. Alterações na Estrutura das Classes

A execução de alterações na estrutura das classes acarreta a **derivação de versões de classe** devido às modificações realizadas em sua estrutura. Nesse caso, tanto a classe quanto os atributos alterados devem estar presentes em uma das versões do esquema.

Por exemplo, a Figura 3 ilustra a inclusão de um relacionamento de agregação entre as classes *Aluno* e *Projeto*. Na primeira versão da classe *Aluno* (Aluno, 1), o domínio do atributo *projeto* é constituído por um valor atômico (alfanumérico). Realizada a modificação, o atributo *projeto* adota como domínio um objeto da classe *Projeto*. Em conseqüência, uma nova versão para a classe *Aluno* é gerada (Aluno, 2).

Esta operação permite, portanto, estabelecer um relacionamento de agregação entre duas classes presentes em uma das versões do esquema. O domínio de um atributo, inicialmente um valor atômico, passa a ser constituído por uma outra classe, sendo estabelecida uma ligação de composição.

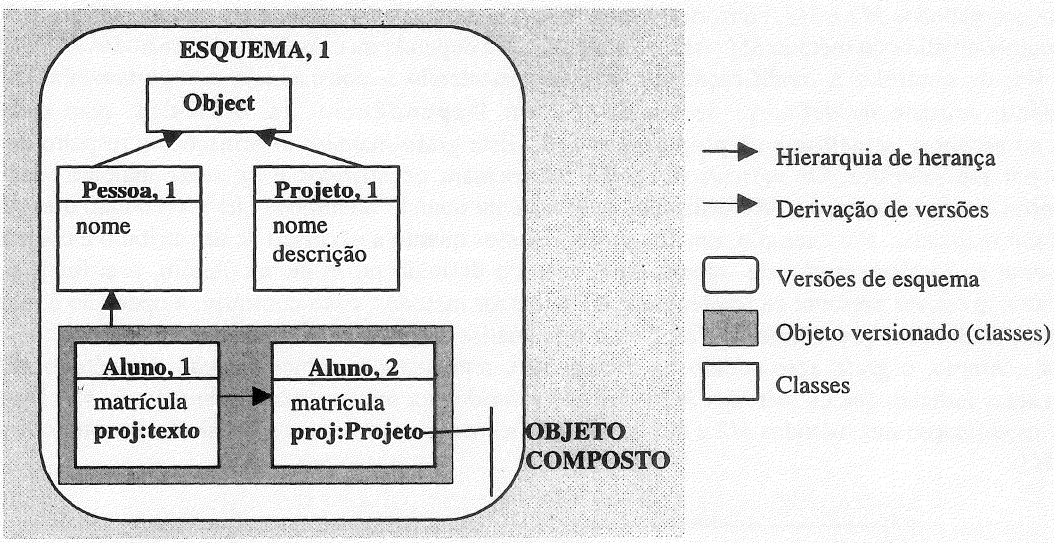

FIGURA 3 - Inclusão de uma Ligação de Composição

4.2.3. Alterações no Comportamento das Classes

A execução de alterações nos métodos provoca a derivação de versões tanto de classe, considerando: inclusão, exclusão e renomeação de métodos, quanto de métodos, como: alteração de código e mudança de domínio e/ou contradomínio. Tanto a classe quanto os métodos modificados devem estar presentes em uma das versões do esquema.

As modificações relacionadas aos métodos precisam garantir a consistência comportamental, respeitando corretamente a assinatura definida no instante da criação.

Tal verificação é essencial porque garante a compatibilidade dos tipos definidos. Porém, não é suficiente para assegurar a consistência comportamental. Por exemplo, a exclusão de um atributo *A*, utilizado por um método *M*, produz um erro na execução desse método, quando requerido após a evolução do esquema. Para tanto, o Modelo proposto estende dois procedimentos adicionais a fim de preservar esta consistência de comportamento.

O primeiro é manter associada a cada versão de esquema uma lista com informações a respeito dos métodos, Lista de Utilização de Métodos, sinalizando os atributos por eles utilizados. Para cada versão de esquema é gerada uma tabela contendo as versões de classes e suas propriedades. Por exemplo, na Figura 4 os atributos da classe *Pessoa* são referidos pelos métodos *M1* e *M2*. Tanto a exclusão do atributo *nome*, quanto do atributo *endereço* é permitida, somente se os métodos forem alterados e a lista atualizada.

CLASSES	ATRIBUTOS	Método: M1	Método: M2
Classe *Pessoa*	nome	x	x
(versão:1)	endereço		x

FIGURA 4 - Lista de Utilização de Métodos

Um método pode ainda fazer referência a outro em sua implementação. Por exemplo, o método *M1* realiza chamadas aos métodos *M2* e *M3*, conforme ilustrado na Figura 5(a). A exclusão do método *M2* ou *M3* levaria a uma execução inválida do método *M1*. Assim, o método *M1* depende da existência dos métodos *M2* e *M3*.

A fim de controlar a modificação em métodos mantendo a consistência comportamental, o segundo procedimento consiste na definição de um `Grafo de Dependência de Métodos`, para cada método definido no esquema, semelhante ao proposto em [16]. Este grafo mantém informações a respeito de todos os métodos por ele referidos. Os *vértices* do grafo representam os *métodos* e as *setas* indicam seus *métodos dependentes*. Essas informações são utilizadas pelo sistema quando alterações que prejudicam a execução dos métodos são realizadas. Por exemplo, um dos procedimentos quanto a exclusão de um atributo é controlar se ele está presente na implementação de algum outro método definido no esquema. Assim, o atributo é excluído somente se não estiver presente na implementação de outros métodos, caso contrário, a operação é rejeitada e o usuário notificado. Esse procedimento é realizado pela análise do grafo de dependência de método.

Por exemplo, o grafo apresentado na Figura 5(b), representa a implementação do método *M1* (Figura 5(a)). As setas indicam que os métodos *M2* e *M3* são referidos na implementação do método *M1*. Desse modo, qualquer modificação nos métodos *M2* e *M3* é realizada somente se preservar a consistência comportamental do método *M1*.

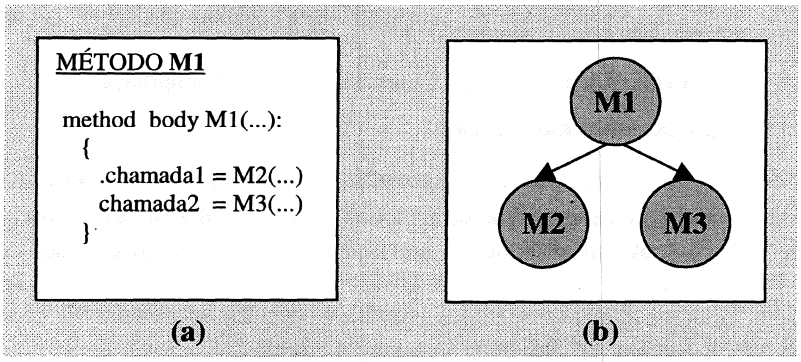

FIGURA 5 - Grafo de Dependência de Métodos

Assim, antes de realizar uma modificação em um método, o Modelo proposto impõe a verificação do Grafo de Dependência de Métodos e a alteração é realizada somente quando não provoca erros nos métodos existentes. Na descoberta de erros, a operação é rejeitada e o usuário é notificado a fim de realizar modificações necessárias.

4.3. Funções de Adaptação

As `funções de adaptação` são definidas a fim de adequar as informações presentes na base de dados com a requerida versão de classe e/ou esquema, acionadas no instante em que o objeto é recuperado. Tais funções podem ser de dois tipos: de `propagação` ou de `conversão`.

4.3.1. Funções de Propagação

As `Funções de propagação`, implementadas pelo usuário, são definidas entre duas versões de esquema, estabelecendo relacionamentos de equivalência entre suas classes. Por exemplo, a Figura 6 ilustra uma alteração no nome de uma classe, classe *Aluno* para *Estudante*, e, consequentemente, a derivação de uma nova versão para o esquema. Nesse caso, qualquer instância, criada pela classe *Aluno* ou *Estudante*, pode ser recuperada em ambas as versões de esquema.

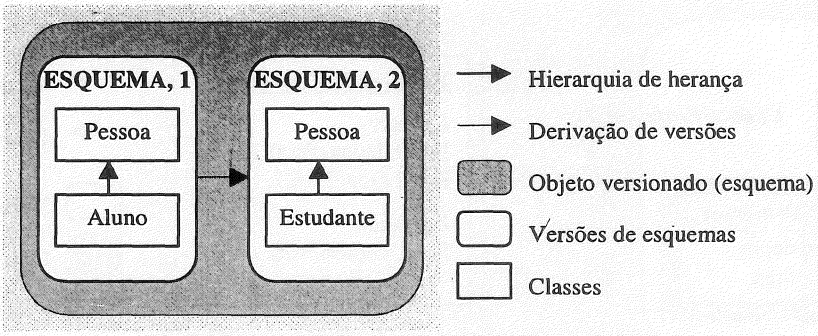

FIGURA 6 - Alteração no Nome de uma Classe

As funções de propagação apresentam duas finalidades quanto aos relacionamentos entre as classes em versões distintas de esquema: `Igualdade` - define como igual duas classes (em versões distintas de esquema) que possuem nomes diferentes mas apresentam propriedades e comportamento idênticos; e `Diferença` – define como diferente duas classes que possuem nomes iguais, porém propriedades e comportamento distintos.

Esse relacionamento semântico é implementado por duas funções de propagação: `Função Update` - conectada a versão inicial do esquema, indicando a equivalência com a próxima versão; e `Função Backdate` - conectada a versão nova do esquema, indicando a equivalência com as classes da versão anterior.

4.3.2. Funções de Conversão

As `Funções de conversão` devem ser especificadas a fim de mapear o conteúdo dos objetos de uma versão de classe para outra, dentro de um mesmo esquema.

Uma função de conversão implica na implementação de duas funções: `Função Update` - associada a classe onde a atualização foi realizada, descrevendo as características dos objetos segundo a próxima versão da classe; e `Função Backdate` - associada a nova versão da classe, descrevendo o comportamento dos objetos na versão anterior as alterações.

As funções de conversão são geradas automaticamente pelo sistema em decorrência da execução das operações de atualização (seção 4.2) na estrutura das classes. Entretanto, elas podem ser construídas e/ou modificadas pelo usuário. Assim, quando uma versão de classe é derivada, caso o usuário não implemente a função de conversão, ela deve ser definida pelo sistema, seguindo definições padrão.

Por exemplo, a Figura 7 mostra a inclusão do atributo *área* na primeira versão da classe *Professor* (*Professor,1*) e, por conseguinte, uma nova versão para classe é derivada (*Professor,2*). Duas funções são geradas: uma Função Update, conectada à primeira versão da classe, descrevendo o comportamento dos objetos após as modificações; e uma Função Backdate, conectada à segunda versão da classe, descrevendo o comportamento dos objetos antes da realização das modificações.

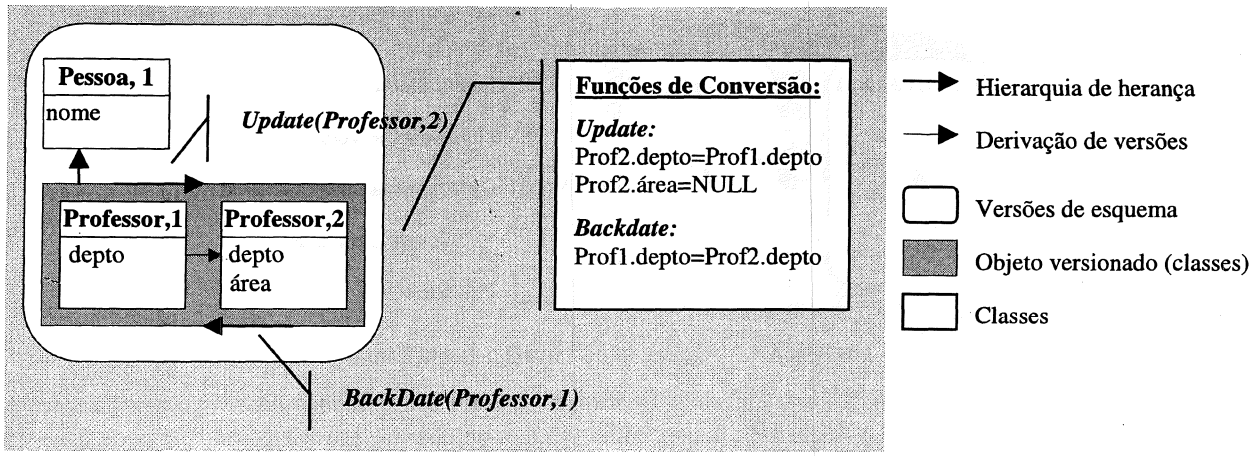

FIGURA 7 - Exemplo de uma Função de Conversão

4.4. Estratégias 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 de classes) definidas no esquema. O mecanismo de versões é a técnica adotada para adaptação das instâncias vigentes no banco de dados.

Quanto um objeto é requerido, por um programa ou consulta, sua estrutura é comparada à definição corrente do esquema e classe a que pertence, verificando se existe uma versão para essa instância do objeto. Caso não haja, uma nova versão é derivada pelo acionamento das funções de conversão, utilizando as regras para armazenamento definidas na seqüência.

Como projetos de bancos de dados, em geral, são extensos e complexos, a realização imediata da adaptação das instâncias pode provocar bloqueios nas aplicações em execução, restringindo as funcionalidades do Modelo proposto. A técnica de conversão adiada[4] é adotada, realizando a adaptação dos objetos somente quando o acesso a eles é requerido.

Como o simples processo de derivação pode implicar em uma excessiva proliferação de versões, alguns requisitos adicionais são impostos visando melhorar o desempenho do sistema, evitando interrupções e ainda garantindo a integridade das instâncias armazenadas. Assim, o processo de derivação compreende as seguintes dimensões para versões:

- versões físicas: versões dos objetos são gravadas fisicamente no banco de dados;
- versões lógicas: versões dos objetos são calculadas pelo sistema e permanecem ativas durante a vigência da aplicação, não sendo propagadas para o banco de dados.

Como a proliferação excessiva de versões pode aumentar a complexidade de armazenamento, manipulação e desempenho, as versões são mantidas fisicamente quando sua utilização é considerada freqüente e mantidas logicamente em situação oposta, sendo averiguadas por mecanismos de métricas definidos por [4]:

- DEFINIÇÃO 1 - Peso de uma Versão de Classe: identificado pela razão entre o número de programas associados a versão da classe e o total de programas definidos para o esquema. Tratando-se da versão corrente, ao peso é atribuído valor máximo (1), pois a disponibilidade das instâncias é considerada altamente importante;
- DEFINIÇÃO 2 - Nível de Pertinência (PT): definido pelo administrador do banco de dados, podendo variar de 0 a 1. Uma versão de classe é considerada pertinente, para os programas aplicativos, quando o peso, calculado pela definição 1, for maior ou igual ao valor de PT e obsoleta, caso contrário.

[4] Na literatura, a técnica de conversão adiada está definida em [16] e também adotada por [4], dentre outros.

Assim, as instâncias são armazenadas fisicamente quando pertinentes e calculadas logicamente, quando obsoletas. Enfim, cada versão de objeto pertence a, pelo menos, um esquema real (físico) podendo adaptar-se a outras perspectivas de esquema/classe.

5. Considerações finais

Presentemente, uma vasta gama de trabalhos tem buscado propor soluções para a questão da evolução de esquemas em bancos de dados orientados a objetos, não somente com o uso de versões, como, por exemplo [2,3,4,5,6,7], mas também através de técnicas de conversão [16], visões [18], dentre outras.

O estudo do Sistema Discente da UFRGS 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 da proposta. Constatou-se que uma das grandes dificuldades encontradas foi a substituição de um banco de dados (esquemas e programas) por outro, inviabilizando a possibilidade de retornar a estados anteriores a transição.

Neste trabalho, um Modelo de Evolução de Esquemas genérico foi apresentado como forma de guiar as modificações em um modelo de banco de dados orientado a objetos, em termos de esquemas, classes, objetos e métodos, e com o objetivo de manter o histórico das alterações realizadas. Esse modelo tem como base o Modelo de Versões definido em [1]. Existem inúmeros trabalhos que associam evolução de esquemas a versões [2,3,4,5,6,7]. Entretanto, cada um enfatiza pontos peculiares e individuais. Dentro desse contexto, buscou-se identificar e tratar todos os aspectos relacionados a evolução de esquemas com versões ao invés de enfatizar características individuais.

Assim, podemos enfatizar as seguintes características abordadas no Modelo de Evolução de Esquemas proposto:

- utilização de versões para esquemas, classes, objetos e métodos, armazenando o histórico das modificações e permitindo a manipulação dos objetos sob qualquer perspectiva de versão;
- definição de restrições quanto às alterações a fim de garantir a consistência dos esquemas resultantes e de suas instâncias frente às modificações realizadas;
- identificação de operações de modificação de esquemas, em particular, modificações quanto à estrutura de objetos complexos, como inclusão e exclusão de ligações de composição, que não foram encontradas em trabalhos prévios;
- tratamento para a excessiva proliferação de versões de objetos, com a utilização de estratégias de propagação de instâncias, armazenando no banco de dados apenas aquelas importantes à aplicação.

Portanto, o diferencial do Modelo proposto é prover uma generalidade frente ao processo de evolução de esquemas conceituais, ao invés de dar ênfase a conceitos esparsos, investigando e explorando todas as nuances envolvidas no processo de evolução de esquemas em bancos de dados orientados a objetos.

O Modelo de Versões de Golendziner [1] foi selecionado por apresentar características peculiares que auxiliam o processo de evolução de esquemas. O mecanismo de herança por extensão, em que há a especificação de versões nos vários níveis da hierarquia de herança, permite uma modelagem mais natural e concisa da realidade.

Por se tratar de um modelo de versões completo e consolidado, fez com que muitos aspectos fossem identificados e abordados no tratamento de evolução de esquemas. Sua utilização mostrou-se adequada à resolução desse problema, conduzindo a uma maior completude do Modelo de evolução proposto. No entanto, as potencialidades do modelo de versões não foram inteiramente exploradas, como, por exemplo, o mecanismo de configuração e o mapeamento de versões entre os diferentes níveis de uma hierarquia.

Enfim, o uso de versões no processo de evolução de esquemas apresenta uma vantagem significativa por permitir o armazenamento do histórico das modificações, possibilitando a manipulação e consulta às versões que habitam simultaneamente o esquema conceitual e a base de dados.

Agradecimentos

Em especial, à professora Dr.a. Lia Goldstein Golendziner (*in memorian*) pela relevante contribuição em motivações, interesse e idéias e pela permanente atenção dedicada aos autores.

Referências

[1] GOLENDZINER, Lia Goldstein. **Um modelo de versões para banco de dados orientados a objetos.** Porto Alegre: CPGCC da UFRGS, 1995. 147p. (Tese de doutorado)

[2] KIM, Won; CHOU, Hong-Tai. Versions of schema for object-oriented databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 14., August 29 - September 1, 1988, Los Angeles, Califórnia. **Proceedings**... [S.l.: s.n.,1988]. p. 148-159.

[3] LAUTEMANN, S. Schema Versions in Object-Oriented Database Systems. INTERNATIONAL CONFERENCE ON DATABASE SYSTEMS FOR ADVANCED APPLICATION, 5., April 1997, Melbourne, Austrália. **Proceedings...** [S.l.:s.n.,1997]. p. 323-332.

[4] BENATALLAH, Boualem; FAUVET, Marie-Christine. Evolution de schéma & Adaptation des instances. Congrés INFORSID'95, Grenoble - France, May 1995. **Proceedings...** [S.l.:s.n.,1995].

[5] ANDANY, J.; LÉONARD M.; PALISSER C. Management of schema evolution in databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES CONFERENCE, 17., Sept. 3-6 1991, Barcelona, Catalonia, Spain. **Proceedings...** [S.l.: s.n., 1995]. p.161-170.

[6] ODBERG, Erik. Schema Modification Management for Object-Oriented Databases. In: CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, 6., June 6-7, 1994, Utrecht, The Netherlands. **Proceedings...** [S.l.:s.n.,1994].

[7] FORNARI, Miguel Rodrigues; GOLENDZINER, Lia Goldstein. Evolução de esquemas utilizando versões em banco de dados orientados a objetos. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 8., 12-14 maio 1993, Campina Grande. **Anais...** Campina Grande, SBC/UFPb, 1993. p.113-127.

[8] GALANTE, Renata de Matos; SANTOS, Clesio Saraiva dos; RUIZ, Duncan Dubugras Alcoba. 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, 13., 1998, Maringá-PR. **Anais...** Maringá: UEM-DIN, 1998. p.303-318.

[9] GALANTE, Renata de Matos. **Um Modelo de Evolução de Esquemas Conceituais para Bancos de Dados Orientados a Objetos com o Emprego de Versões.** Porto Alegre: CPGCC da UFRGS, 1998. (Dissertação de Mestrado)

[10] GALANTE, Renata de Matos. **Suporte à evolução de esquemas com o uso de versões**: trabalho individual. Porto Alegre: CPGCC da UFRGS, 1996. 60p. (Trabalho Individual n°575).

[11] UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL - CENTRO DE PROCESSAMENTO DE DADOS. **Descrição de Campos do Sistema de Controle Acadêmico - UDB.** Porto Alegre: CPD-UFRGS, 1980.

[12] UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL - CENTRO DE PROCESSAMENTO DE DADOS. **Modelo Conceitual de Dados da Área de Ensino da UFRGS.** Porto Alegre: CPD-UFRGS, 1997.

[13] UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL - CENTRO DE PROCESSAMENTO DE DADOS. **Modelo de Dados de Implementação do Sistema Discente sobre o SGBD DMSII-Unisys.** Porto Alegre: CPD-UFRGS, 1997.

[14] ATKINSON, M. et al. **The object-oriented database system - manifesto.** [S.l.:s.n., 1989]. (Rapport Technique Altaïr 30-89)

[15] BANERJEE, Jay; KIM, Won. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, May 27-29 1987, San Francisco, CA. **Proceedings...** New York: ACM Press, 1987. pp. 311-322.

[16] ZICARI, Roberto. A framework for schema updates in an object-oriented database system. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 7., April 8-12, 1991, Kobe, Japan. **Proceedings...** Los Alamitos: IEEE, 1991. p.2-13.

[17] FORNARI, Miguel Rodrigues. **Um estudo sobre evolução de esquemas em banco de dados.** Porto Alegre: CPGCC da UFRGS, 1992. 78p. (Trabalho Individual)

[18] MOTSCHINING-PITRIK, R. Requirement and comparison of view mechanisms for object-oriented databases. **Information Systems**, Oxford, v.21, n.3, p.229-252, May 1996.