UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Um Modelo de Evolução de Esquemas Conceituais para Bancos de Dados Orientados a Objetos com o Emprego de Versões

por

RENATA DE MATOS GALANTE

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

Prof. Dr. Clesio Saraiva dos Santos
Orientador

Porto Alegre, dezembro de 1998.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

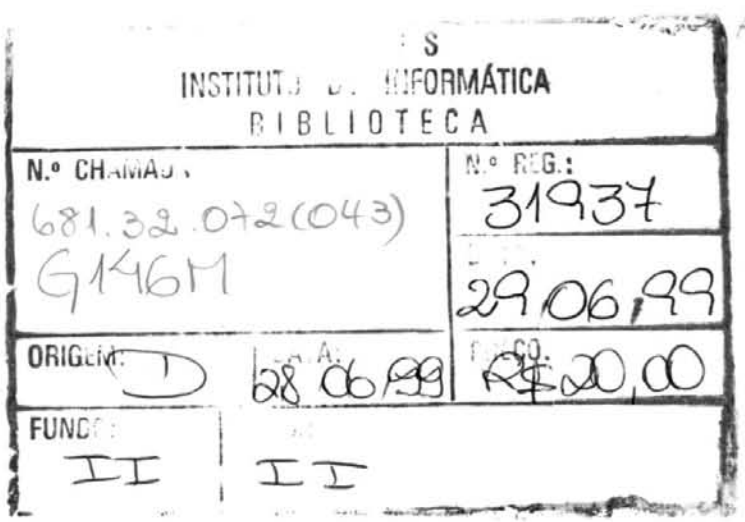
Galante, Renata de Matos

Um Modelo de Evolução de Esquemas Conceituais para Bancos de Dados Orientados a Objetos com e Emprego de Versões / por Renata de Matos Galante. - Porto Alegre: CPGCC da UFRGS, 1998.

92f.: il.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 1998. Orientador: Santos, Clesio Saraiva dos.

1. Orientação a objetos. 2. Evolução de esquemas. 3. Versões. I. Santos, Clesio Saraiva dos. II. Título.



Armazenamento da Informação - SBD
Banco: Dados
Versões: Banco: Dados
Orientação. Objetos
CNPq 1.03.03.00-6

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Prof.a. Wrana Panizzi
Pró-Reitor de Pós-Graduação: Prof. José Carlos Ferraz Hennemann
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenadora do CPGCC: Prof.a. Carla Maria Dal Sasso Freitas
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

"*O real não está na saída e nem na chegada, está na travessia.*"
(João Guimarães Rosa)

Dedico esse trabalho a minha família, por não ter poupado esforços e incentivos para que fosse realizado e, principalmente, por toda confiança depositada.

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:

- ao meu orientador Prof. Dr. Clesio Saraiva dos Santos, pelo crédito de confiança, condução serena do trabalho, orientação segura, correção criteriosa e, principalmente, por todo apoio, incentivo e ajuda dispendidos;
- à Prof.a Dra. Lia Golendziner, pela relevante contribuição em motivações, interesse e idéias e pela permanente atenção dedicada;
- ao Prof. Dr. Duncan Dubugras Alcoba Ruiz pelo auxílio, comentários e sugestões valiosas;
- ao Instituto de Informática e Curso de Pós-Graduação em Ciência da Computação da UFRGS pela oportunidade oferecida para realização desse trabalho e à CAPES (Coordenação do Aperfeiçoamento de Pessoal de Nível Superior) pelo suporte financeiro;
- Às bibliotecárias Beatriz Haro e Ida Rossi;
- aos amigos que compartilharam esses anos de estudos e expectativas: Carolina Sturm Trindade, Daniela Theodoro, Gilda Assis, Gustavo Arnold, Ítalo Modesto Dutra, Ivete Martins Pinto, Juliana Lombardi, Juliano Tonezer, Michelle Fontana, Oni de Almeida, Patrícia Pretes, Patrícia Tagliamento, Ricardo Dorneles e Simone Dias Cruz;
- aos amigos da Concept: Arnaldo, Cleide, Denise, Márcio, Marta e Paulo, em especial, à Márcia Tanaka pelo exemplo de profissionalismo e perseverança.
- à Maria Goreti e Renato Mallmann, Carol e Renatinha, pela excepcional acolhida;
- à Karen Basso, pela presença amiga, exemplo de empenho e dedicação e um agradecimento especial ao amiguinho André pela constante admiração;
- à Fabiana Zamora, pela amizade sincera e exemplo de segurança e determinação;
- à Andréia Dal Pizzol, pela amizade e companhia nas longas viagens;
- à grande amiga de todas as horas, Marilene Noronha (Marie), pela sinceridade e paciência e, em especial, por compartilhar as inúmeras "versões" de incertezas e alegrias que surgiram ao longo dessa caminhada. Ainda, à Áurea e Torquatro Noronha pelos almoços e agradável companhia nas tardes de domingo;
- à Ana Paula e Marcos pela constante preocupação;
- a Luciano Porto Barreto, por tudo;
- a minha família, pelo amor, compreensão, estímulo e apoio dispendidos em todos os momentos de minha vida: aos meus pais, Ailton e Izabel, pela permanente preocupação com a formação intelectual e as minhas irmãs, Fernanda e Claudia, pela presença constante, embora distantes.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Aplicações ditas não convencionais, como, por exemplo, CAD, CASE, Automação de Escritórios, entre outras, 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.

Neste trabalho o Modelo de Versões proposto por Golendziner é empregado no contexto da evolução de esquemas. 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, mantendo um histórico da evolução do esquema do banco de dados.

É proposto um modelo flexível de suporte a evolução de esquemas em bancos de dados orientados a objetos, bem como estratégias de propagação das instâncias vigentes na base de dados. O histórico das modificações é representado pela derivação de versões do esquema e de seus elementos. Os estados anteriores às transformações são preservados, permitindo aos usuários a navegação retroativa e proativa entre versões, para realização de operações consistentes de modificação e consulta.

PALAVRAS-CHAVE: Bancos de Dados Orientação a Objetos, Evolução de Esquemas e Versões.

TITLE:" A Schema Evolution Model for Object-Oriented Databases with Versions"

Abstract

Non-conventional applications such as CAD, CASE, office automation often require the maintenance of various database states, to keep track of the history of the performed updates. The concept of version is employed to support such requirement.

In this work, the version model proposed by Golendziner is used in the schema evolution context. Versions are used to store the different states of the schema, classes and methods, as well as for the mapping of database instances among the various schema versions, thus keeping the history of the database schema evolution.

A flexible model is proposed to support schema evolution in object-oriented databases, as well as the strategies to propagate the corresponding changes to the database instances. Versions of schema, as well as versions of the schema elements represent their evolution history. In the proposed model, previous states are preserved allowing the user to make queries about consistency and modifications in both backward and forward version.

KEYWORDS: Object-Oriented Databases, Schema Evolution and Versions.

1 Introdução

As aplicações ditas convencionais empregam geralmente o conceito de transação para atualização das informações, convertendo a base de dados de um estado consistente para outro igualmente correto. Entretanto, aplicações não convencionais, como, por exemplo, CAD, CASE, Automação de Escritórios, entre outras, exigem freqüentemente, a manutenção de diversos estados da base de dados, retendo o histórico das modificações realizadas.

Como resposta a tal requisito, alguns trabalhos mais recentes têm enfocado a questão do suporte a versões. Um deles [GOL93a, GOL93b, GOL95a, GOL95b, GOL95c, GOL95d] 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. Dentro desse contexto, 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.

A utilização de versões tem crescido significativamente nos últimos anos, tendo aplicações em diversas áreas. Em [DIT88] são encontrados diversos objetivos e aplicações de versões, tais como:

- registro de diferentes alternativas de um objeto: permite ao usuário explorar simultaneamente diferentes configurações de um mesmo objeto;
- controle de concorrência: utilizadas com a finalidade de isolar determinados usuários de mudanças realizadas por outros usuários;
- recuperação de falhas: uma versão pode representar um estado consistente do banco de dados na qual o sistema pode retornar na presença de falhas;
- aumento do desempenho em sistemas distribuídos: versões de um mesmo dado são mantidas em diferentes "*sites*", sendo permitida a presença de dados obsoletos em um curto espaço de tempo a fim de evitar freqüentes transmissões de dados;
- sistemas hipermídia: versões são empregadas na manutenção do histórico de documentos, permitindo suporte ao trabalho corporativo e mantendo diferentes representações para uma mesma informação;
- evolução de esquemas: versões são utilizadas para manutenção do histórico das modificações realizadas em bancos de dados, permitindo futuras atualizações e consultas.

Sistemas de bancos de dados orientados a objetos têm sido desenvolvidos, principalmente, para modelos e aplicações altamente dinâmicas que manuseiam objetos estruturados: grandes e complexos, que apresentam, freqüentemente, modificações tanto no seu valor quanto em sua estrutura. Por exemplo, em projetos de engenharia são necessárias alterações também no esquema para tratar adequadamente projetos em evolução.

Um banco de dados sobrevive, tipicamente, a décadas servindo programas de aplicação, entretanto, com o passar do tempo, alterações são requeridas a fim de refletir de forma completa e concisa a parte relevante do mundo real que está sendo modelada.

As modificações podem ocorrer tanto durante a fase de projeto, como na fase de desenvolvimento do sistema. No primeiro caso, representa um refinamento da aplicação e/ou experimentação de novas percepções, uma vez que o esquema não está completamente definido. No segundo, retrata uma evolução da realidade considerada, como, por exemplo, reformulação das necessidades dos usuários, correção de um erro de projeto, adaptação de novos componentes ou, ainda, melhoria no desempenho da aplicação.

Num contexto de orientação a objetos, é difícil encontrar um sistema gerenciador de banco de dados que ofereça uma solução genérica ao problema de evolução de esquemas, devido à complexidade em manter a coerência quando ocorre a propagação dos efeitos das mudanças. Surgem inúmeras conseqüências dessa evolução em cada um dos níveis da aplicação. No nível de esquema, modificações em tipos, classes, relacionamentos de agregação e generalização não podem afetar outras partes do esquema definido. No nível dos programas aplicativos, as alterações não devem invalidar os programas existentes. No nível das instâncias, a evolução deve respeitar as relações de correspondência estrutural e semântica entre os dados e o esquema definido.

Na ocorrência de modificações, o sistema deve fornecer ferramentas de manipulação de esquemas para que o banco de dados possa ser corretamente atualizado. Vários trabalhos têm adotado o conceito de versão no tratamento de 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 [KIM88] e [LAU96a, LAU96b, LAU97a, LAU97b], a fim de representar o estado evolucionário do esquema. Em [BEN94, BEN95a, BEN95b] o conceito de versão é adotado para esquemas, classes e instâncias. Em [AND91] as unidades alteráveis são contextos, compostos por uma seleção de classes, previamente definida no esquema, e apenas a essas visões é permitida a característica de possuir versões. Em [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] o versionamento é utilizado para esquemas e classes. Versões de 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 [FOR93a] instâncias, classes e métodos são tratados de maneira uniforme, sendo todos considerados objetos versionáveis.[1]

A utilização de orientação a objetos juntamente com o mecanismo de controle de versões permite aliar duas poderosas ferramentas de modelagem, pois, enquanto o uso de versões consegue modelar adequadamente o caráter evolutivo de um ambiente de projeto, a abordagem orientada a objetos permite uma modelagem mais próxima da realidade da aplicação, representando a natureza complexa das atividades envolvidas. Versionamento e evolução de esquemas representam uma adequada solução para o manuseio de esquemas de dados, pois admite definir diferentes organizações conceituais de um mesmo domínio da aplicação, mantendo a funcionalidade das instâncias armazenadas frente a alterações, evitando a perda de informações e garantindo a compatibilidade dos programas aplicativos.

Esse trabalho provê um modelo de evolução de esquemas, onde versões, particularmente o modelo definido por Golendziner em [GOL95a], são utilizadas para

---

[1] Um levantamento bibliográfico mais detalhado a respeito de propostas que utilizam mecanismos de versões no tratamento de evolução de esquemas está documentado em [GAL96].

representar 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 trabalho está organizado da seguinte maneira. O capítulo 2 apresenta os conceitos básicos e as características relevantes presentes nos modelos de evolução de esquemas propostos na literatura, considerando as várias técnicas existentes para seu tratamento e estabelecendo uma possível estrutura padrão para um mecanismo de evolução de esquemas. O capítulo 3 apresenta uma abordagem comparativa entre as diversas propostas presentes na literatura, considerando os conceitos básicos relacionados à evolução de esquemas e versões. Um estudo de caso é mostrado no capítulo 4, considerando a evolução do Sistema de Controle Acadêmico vigente na Universidade Federal do Rio Grande do Sul, com o intuito de identificar requisitos e funcionalidades necessárias a um mecanismo de evolução de esquemas. O capítulo 5 apresenta os conceitos básicos do modelo de dados e de versões, identificando as características requeridas pela proposta. O modelo de evolução de esquemas é detalhado no capítulo 6, apresentando as restrições quanto à validade das modificações de esquemas, operações permitidas, bem como o método de propagação para as instâncias armazenadas no banco de dados. No capítulo 7 são apresentadas as conclusões e identificados aspectos que podem originar maiores aprofundamentos em estudos futuros.

2 Evolução de Esquemas – Características e Técnicas

Durante o desenvolvimento desse trabalho, diversas propostas de evolução de esquemas foram consideradas para levantamento bibliográfico, conforme analisado no capítulo 3. Nesse capítulo, com base nas características presentes nessas propostas, é realizado um levantamento a respeito dos procedimentos de evolução de esquemas resultando em um panorama amplo de todo processo. São identificadas as principais técnicas envolvidas em evolução de esquemas conceituais bem como as estratégias de propagação das alterações nas instâncias armazenadas. Por fim, uma estrutura é estabelecida, definindo as características essenciais para um modelo de evolução de esquemas.

2.1 Evolução de Esquemas e suas Aplicabilidades

Bancos de dados são utilizados para armazenar informações do mundo real, onde seu esquema conceitual representa os requisitos definidos para a aplicação. Entretanto, com o passar do tempo, esses requisitos evoluem, impondo a necessidade de atualização desse esquema. O processo de evolução de esquemas consiste em manter o sistema funcionando face às mudanças realizadas nas definições do banco de dados, garantindo a coerência das informações armazenadas. Sendo assim, o mecanismo de evolução de esquemas pode ser considerado como um requisito relevante para os novos sistemas gerenciadores de bancos de dados.

Bancos de dados orientados a objetos permitem ainda uma modelagem mais dinâmica e abrangente da realidade, sendo geralmente adotados para modelagem de estruturas complexas presentes em aplicações de bancos de dados não-convencionais. A necessidade de alteração do esquema conceitual tem sido identificada, em muitos domínios de aplicações, principalmente, emergindo em sistemas de aplicações intensivas, como, por exemplo, sistemas de automação de escritório, projetos de engenharia, inteligência artificial, dentre outras. Dentro desse contexto, tem-se tornado necessário desenvolver ferramentas que ultrapassem as características de sistemas individuais aumentando as possibilidades de reestruturação em bancos de dados.

2.1.1 Evolução de Esquemas e suas Exigências

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.

Existe um considerável número de razões que impõe necessidades de evolução em um sistema de banco de dados orientado a objeto, tais como:

- correção do esquema de dados diante de problemas encontrados durante a fase de operação e experimentação com a base de dados;

- adaptação do esquema de dados aos novos componentes ou circunstâncias que envolvem o sistema de banco de dados e seu ambiente de utilização;
- atualização de componentes do esquema de modo a acompanharem as constantes modificações de requisitos dos usuários e a evolução do empreendimento;
- refinamento de componentes do esquema que foram pouco detalhados durante a fase de projeto e portanto estavam semanticamente pobres;
- experimentação de novas concepções do esquema na busca de alternativas para a modelagem da base de dados.

2.1.2 Evolução do Esquema Conceitual e Físico

A evolução do esquema conceitual e físico envolve operações sobre aspectos estruturais e comportamentais dos elementos do esquema que podem implicar em alterações das 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.

2.1.3 Técnicas de Evolução de Esquemas

As técnicas de evolução de esquemas não possuem denominações usuais, pois não foram ainda classificadas. As operações básicas, nas quais se fundamentam, podem ser utilizadas como critério de classificação.

2.1.3.1 Conversão

Em uma abordagem baseada na correção, o estado anterior à atualização não é conservado, isto é, a revisão no esquema é repercutida automaticamente sobre o ambiente de maneira irreversível, sem levar em consideração a perda de informações. A nova definição é trocada pela antiga. Por exemplo, o modelo de evolução de esquemas proposto pelo sistema O2 [FER95] situa-se nesta categoria.

Algumas funcionalidades adicionais são requeridas a fim de preservar a coerência do esquema frente à base de dados e aplicativos existentes:

- definir restrições para que as modificações preservem a correspondência entre as instâncias da base de dados e as entidades do esquema;
- definir mecanismos de propagação de modificação sobre os objetos persistentes da base de dados, para que respeitem as novas definições impostas pela estrutura do esquema;
- definir um formalismo para guiar as modificações de esquema.

Essa técnica é conveniente e geralmente empregada por aplicações monousuários ou quando a perda de informação é consideravelmente aceitável.

2.1.3.2 Adição

Essa técnica tem como característica a inclusão de novos elementos ao esquema implantado, ou seja, a idéia é acrescentar uma quantidade significativa de novos componentes de representação ao esquema de dados. A adição de uma grande quantidade de componentes caracteriza sua extensão para novos contextos de representação do empreendimento.

Essa técnica pode oferecer meios para adição de subesquemas, como, por exemplo, a adição de esquemas suplementares [CAM96] ou mesmo a incorporação gradativa de esquemas em relação às instâncias vigentes na base de dados.

2.1.3.3 Versões

Em uma abordagem baseada em versões o estado anterior às modificações é retido mantendo o histórico da evolução.

São identificados dois tipos de versões: históricas e alternativas. Nas versões históricas, cada versão é mantida aos dados que lhe são associados, ou seja, as versões são armazenadas em espaços físicos separados, sendo independentes entre si. Toda modificação é efetuada sobre a estrutura corrente e as versões antigas são recuperáveis somente para consulta. As versões alternativas permanecem armazenadas num espaço comum, evoluem em paralelo e operam sobre a mesma coleção de dados. São acessíveis para consultas e revisões e todas as operações lhes são aplicáveis. São ainda consideradas versões de esquemas, onde uma mudança no esquema de dados desencadeia uma versão completa da base; e versões de classes, nas quais toda evolução de uma classe cria uma nova versão da classe, de suas subclasses e das classes ligadas por relacionamento de agregação. Nesse caso, uma visão global do esquema deve ser criada a fim de definir as ligações existentes entre as diferentes classes, representando o esquema num determinado instante de tempo.

A utilização da técnica de versionamento apresenta alguns inconvenientes, como, por exemplo, o tempo adicional despendido na geração, manutenção e controle da proliferação das versões. A complexidade imposta pela navegação no grafo de herança tradicional é também adicionada pela necessidade de navegação em uma hierarquia de versões.

Entretanto, inúmeras vantagens são apresentadas por essa técnica, tais como:

- capacidade de registrar todo o histórico das modificações realizadas no esquema, bem como a posterior utilização para revisões e consultas;
- permitir o trabalho em paralelo, onde diferentes caminhos para modelagem de aplicações complexas podem ser testados. Por exemplo, em aplicações de projeto (CAD), um único esquema é utilizado simultaneamente por vários usuários, no qual cada um, individualmente, pode modificar as entidades do esquema; e
- possibilidade de reconstituição em caso de falha, permitindo retornar a estados anteriores as modificações.

2.1.4 Estratégias de Propagação nas Instâncias

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. Na literatura, pode-se encontrar muitas variações dessas estratégias para adaptá-las a problemas específicos. No entanto, essas estratégias podem enquadrar-se nas categorias descritas a seguir.

2.1.4.1 Conversão

Nesta abordagem, as transformações realizadas na estrutura do esquema são refletidas fisicamente na base de dados, ou seja, as instâncias armazenadas são convertidas conforme a nova definição, sendo perdidas as informações anteriores. Por exemplo, a exclusão de um atributo de uma classe implica na exclusão desse atributo em todos os objetos criados para a classe.

Duas estratégias são consideradas por esta técnica:

- Conversão Imediata – todos os objetos da base de dados são atualizados após a modificação do esquema;
- Conversão Adiada – os objetos são atualizados somente quando são requeridos pela aplicação.

A vantagem da utilização de conversão adiada ou imediata está relacionada à quantidade de objetos envolvidos no processo. Por exemplo, torna-se inviável o uso da técnica de conversão imediata quando a aplicação manipula um grande número de objetos volumosos e complexos, pois o sistema permanece paralisado durante o período de conversão. Por outro lado, quando a quantidade de objetos é limitada, a conversão imediata torna-se conveniente, pois melhora o desempenho do sistema, não havendo parada para conversão.

A técnica de conversão é indicada para aplicações que não solicitam informações sobre antigos estados de dados, pois estas são sempre perdidas.

2.1.4.2 Emulação

A técnica de emulação ou mapeamento consiste em prorrogar indefinidamente a atualização das instâncias mesmo após o surgimento de um novo esquema. Nenhuma alteração física é realizada nas informações instanciadas, isto é, o sistema realiza uma adaptação lógica da informação segundo o novo esquema, que age como um filtro para criar a ilusão de instâncias modificadas, porém essas atualizações não são repercutidas fisicamente no banco de dados.

Essa técnica é geralmente utilizada em conjunto com a técnica de versões de esquemas e/ou classes para fins de adaptação das instâncias presentes no banco de dados. A principal vantagem é a inexistência de proliferação de versões, facilitando o processo de controle e manipulação. Entretanto, o sistema sofre uma degradação, pois, toda vez que um objeto é solicitado, é preciso adaptá-lo com a estrutura atual definida para a classe.

2.1.4.3 Versões

A técnica de versões, para adaptação das instâncias presentes no banco de dados, consiste em gerar versões de objetos para cada nova versão de classe ou esquema derivada na estrutura.

As versões de instâncias podem ser criadas livremente pelo usuário, desde que monitoradas pelo sistema de banco de dados, ou podem também ser geradas automaticamente por cópia ou conversão, de uma versão de instância para outra, desde que seja utilizado o esquema de dados correspondente.

Um dos inconvenientes da utilização da técnica de versionamento de objetos é a excessiva proliferação de versões. Como conseqüência, há a dificuldade na navegação pela estrutura de versões, complexidade no controle de compatibilidade entre objetos e classes e acréscimo no custo de armazenamento do histórico das versões.

Essas desvantagens podem ser compensadas pelos diversos benefícios que apresenta, por exemplo:

- permite o armazenamento do histórico das modificações realizadas na estrutura do esquema e repercutidas na base de dados;
- a evolução do esquema não provoca o encerramento do sistema nem requer a reorganização do banco de dados;
- permite que as informações presentes no banco de dados comportem-se diferentemente, adaptando-se às diversas definições de uma mesma versão de classe ou esquema.

2.2 Padrão para um Mecanismo de Evolução de Esquemas

Embora cada proposta analisada apresente relativa variação nas funcionalidades que proporcionam em relação ao processo de evolução de esquemas, é possível estabelecer categorias relevantes presentes na maioria das propostas e, a partir dessa análise, uma estrutura genérica para um modelo de evolução de esquemas com versões pode ser definida.

Cada modelo de dados define elementos existentes em seu esquema, uma taxonomia de operações de evolução de esquemas, como [Kim90], e regras semânticas 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.

A Figura 2.1 apresenta uma possível estrutura genérica para um mecanismo de evolução de esquemas.

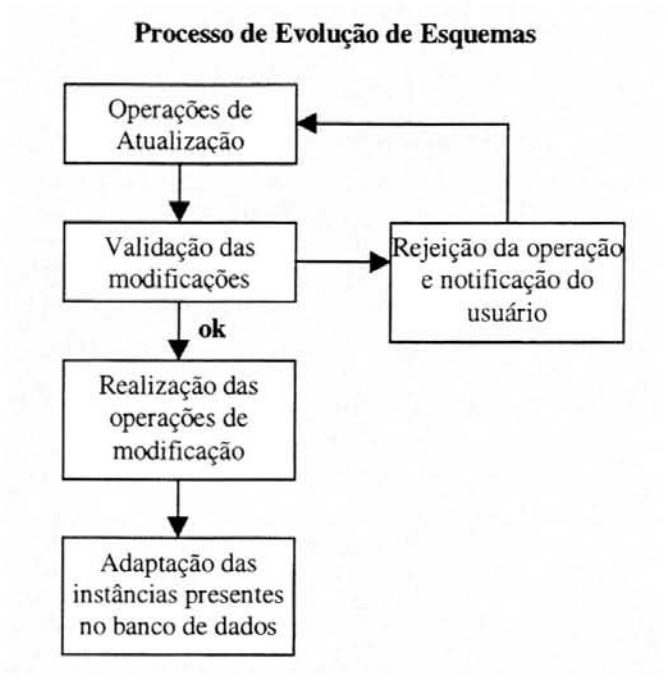


FIGURA 2.1 - Estrutura básica para mecanismo de evolução de esquemas

No estudo realizado por Benatallah [BEN94, BEN95a, BEN95b] alguns fatores de qualidade são estabelecidos para que o mecanismo forneça uma certa satisfação aos usuários, tais como:

- o modelo de evolução de esquemas deve fornecer diferentes níveis de abstração durante a realização das operações, tanto do ponto de vista do usuário quanto do conhecimento do domínio da aplicação;
- coerência: toda atualização realizada deve garantir a consistência e integridade do esquema, sendo capaz de detectar possíveis anomalias de uma operação e reagir solucionando o problema;
- completude: a evolução do esquema deve sempre ser realizada independente da forma como a situação se apresente;
- perenidade das informações: a evolução do esquema deve evitar a perda de informações, bem como permitir acesso reversível dos dados presentes no banco de dados.

Um ponto relevante a ser considerado em um mecanismo de evolução de esquemas é a transparência nas alterações, ou seja, as aplicações devem continuar operacionais frente a qualquer tipo de modificação realizado, permanecendo imunes às alterações.

A adição de todas essas funcionalidades tem grande impacto sobre os componentes do sistema, porém o acesso às transformações e propagações das modificações precisa apresentar um desempenho aceitável, de modo que o sistema deve ser dotado de estruturas de armazenamento apropriadas para o bom acesso às informações pela aplicação.

2.3 Considerações

Apesar de reconhecida a importância dos mecanismos de evolução de esquemas em sistemas de bancos de dados e, mais recentemente, de várias linhas de pesquisas, como, por exemplo [AND91, BAN87, BEN95b, CHE94, FER95, FOR93a, LAU96a, LAU96b, LAU97a, LAU97b], 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 um mecanismo elaborado com amplo suporte às necessidades de evolução impostas pelas atuais aplicações de banco de dados.

Cabe salientar que tanto as técnicas de modificação quanto as de adaptação da base de dados não são consideradas isoladamente, podendo haver uma combinação de várias técnicas em um único modelo, desde que atenda às necessidades da aplicação, proporcionando, assim, maiores possibilidades para confecção de um mecanismo adequado e eficiente.

Evolução de esquemas com a utilização de versões 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.

3 Evolução de Esquemas Conceituais: Sistemas e Características Encontradas

Neste capítulo é realizada 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 de versões. Os itens selecionados como parâmetros de comparação foram identificados perante o tratamento adotado pela maioria das propostas analisadas, pois não existe um modelo padrão nem um mecanismo completamente definido ou implementado. Uma análise mais minuciosa destas abordagens está documentada em [GAL96].

3.1 Revisão Bibliográfica

Foram analisadas seis propostas que utilizam o mecanismo de versões no tratamento de evolução de esquemas. Nesta seção, uma visão geral é apresentada para cada proposta, seguindo a ordem cronológica de publicação.

3.1.1 Proposta segundo Won Kim

Em [KIM88] é proposto um modelo de versões de esquema, caracterizando uma extensão ao modelo de versões de objetos simples, ambos desenvolvidos pelo grupo do autor. Um protótipo de um banco de dados orientado a objetos, denominado ORION, foi desenvolvido, estando implementado o paradigma de orientação a objetos, persistência e compartilhamento de objetos através do suporte de transações. Várias funções para aplicações de CAD/CAM são fornecidas, como versões, objetos compostos, evolução dinâmica de esquema e gerenciamento para dados multimídia.

3.1.2 Proposta Farandole 2

Farandole 2 [AND91] é um laboratório experimental escrito em ADA. Inclui um sistema gerenciador de banco de dados, denominado ECRINS [JUN86], e um ambiente de processo. Os autores estão vinculados à Universidade de Genève, na Suiça.

Essa proposta apresenta um mecanismo baseado em versões para gerenciar 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 [ALJ95, JAD95], porém o mecanismo de versões é um processo cuja implementação está em andamento.

3.1.3 Proposta segundo Miguel Rodrigues Fornari

A proposta descrita em [FOR92, FOR93a, FOR93b] tem os autores vinculados à Universidade Federal do Rio Grande do Sul, Brasil, e apresenta um mecanismo genérico de evolução de esquemas para bancos de dados orientados a objetos baseado no modelo

de dados chamado GNOMO (*Generic Object-Oriented Data Model*). A correção do esquema e a consistência da base de dados é mantida pela definição de invariantes e geração de versões. Uma série de possíveis operações é descrita para evolução de esquemas, procurando manter a transparência das alterações.

O ambiente STAR é um *framework* para desenvolvimento de ambientes de projeto de sistemas digitais assistido por computador [WAG91], sendo composto por alguns subsistemas, chamados gerentes. Dentro desse contexto, um gerente de evolução de esquemas foi proposto para esse ambiente [FOR94a, FOR94b, FOR94c].

A implementação do protótipo da ferramenta de evolução de esquemas foi realizada utilizando o sistema KRISYS e a linguagem LISP, sendo que está prevista a definição de uma linguagem textual, no estilo SQL, para permitir a modelagem de esquemas de objetos e especificar uma interface gráfica a ser integrada ao *browser*, baseada em ambientes de janelas padrão *OpenWindows*.

3.1.4 Proposta segundo Erik Odberg

A proposta descrita em [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] apresenta um método de versionamento para gerenciar modificações no esquema, permitindo que diversos aplicativos retratem diferentes percepções do mundo real. Uma nova versão de esquema representa a diferença existente com relação ao esquema base definido. Um dos principais objetivos é a habilidade em estabelecer relacionamentos semânticos entre classes definidos em qualquer versão do esquema. Esses relacionamentos garantem o comportamento consistente dos objetos em qualquer contexto em que são requeridos, pois mantêm uma associação entre os objetos e as classes dentro de diferentes contextos de versão de esquema.

Um limitado protótipo para bancos de dados orientados a objetos, incluindo modelo de regras de objetos e aspectos essenciais ao método de gerenciamento de modificação de esquema, está sob implementação. O objetivo final é demonstrar um projeto viável, com implementação dos conceitos de bancos de dados orientados a objetos.

3.1.5 Proposta segundo Boualem Benatallah

A proposta descrita em [BEN94, BEN95a, BEN95b] apresenta um método híbrido para gerenciamento de evolução de esquemas, combinando técnicas de modificação e versionamento. Provê operações para derivação de novas versões e técnicas de adaptação das instâncias presentes no banco de dados, baseando-se na disponibilidade dos objetos e limitando o número de versões àquelas que apresentam freqüente acesso.

Boualem Benatallah é membro do grupo de banco de dados do Instituto IMAG (Pólo Ciência da Computação e Matemática Aplicada, Grenoble – França). Suas pesquisas estão direcionadas à evolução de esquemas em sistemas de bancos de dados orientados a objetos. Esse modelo está sendo implementado sob o SGBD O2.

3.1.6 Proposta segundo Sven-Eric Lautemann

A proposta descrita em [LAU96a, LAU96b, FER96, LAU97a, LAU97b] 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.

Esta proposta faz parte de um projeto, denominado *COAST (Complex Object And Schema Transformation)*, desenvolvido no *Center Competence* da Universidade de Frankfurt/Main, na Alemanha. Seu objetivo principal é implementar o protótipo de um SGBDOO que suporte múltiplas versões de esquema que possam ser utilizadas por aplicações em paralelo.

3.2 Conceitos Associados aos Modelos de Dados

3.2.1 Tipos de Herança

Os tipos de herança, simples e múltipla, presentes no modelo de dados de cada proposta, são identificados. Quanto à presença de herança múltipla, as técnicas adotadas para resolução de conflitos de nomes são também analisadas.

FARANDOLE 2 [AND91] apresenta somente herança simples, permitindo que uma classe contenha uma subclasse única e direta. As demais propostas, além de herança simples, admitem a presença de herança múltipla.

FORNARI [FOR93a] e KIM [KIM88] estabelecem uma ordem de prioridade quanto à resolução de conflitos de herança, podendo ser indicada pelo projetista ou resolvida pelo sistema. Nesse caso, a propriedade presente na primeira superclasse definida para a classe é selecionada. ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] apresenta restrições apenas no instante em que as modificações são realizadas no esquema, não permitindo que propriedades resultantes de herança sejam alteradas ou removidas.

3.2.2 Mecanismos de Herança

Os mecanismos de herança adotados pelas propostas de evolução de esquemas são analisados e comparados, distinguindo-se:

- herança por refinamento: todas as propriedades das superclassses migram para os níveis inferiores da hierarquia de especialização;
- herança por extensão: as propriedades permanecem nos níveis (classes) onde foram definidas.

Todas as propostas analisadas apresentam herança por refinamento, ao contrário do adotado pelo modelo de versões [GOL95a], em que as versões existem em mais de um nível da hierarquia de herança. A utilização do mecanismo de herança por extensão provê uma modelagem mais dinâmica das aplicações, pois, durante a fase de projeto, o esquema raramente apresenta-se estável ou completamente definido, sendo possível realizar modificações em cada nível separadamente, garantindo a coerência das instâncias existentes.

3.3 Conceitos Associados aos Modelos de Versões

3.3.1 Objeto Versionado

O conceito de objeto versionado é abordado como uma estrutura que representa e agrupa todas as versões de um mesmo objeto do mundo real. Assim, as propostas que adotam esse conceito são identificadas e comparadas.

ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d], FORNARI [FOR93a] e LAUTEMANN [LAU97a, LAU97b] não definem explicitamente o conceito de objeto versionado, apenas apresentam a estrutura dos relacionamentos entre as versões (vide 3.3.2).

BENATALLAH [BEN94, BEN95a, BEN95b] e FARANDOLE 2 [AND91] apresentam o conceito de objeto versionado, cuja função é armazenar informações sobre suas versões. Possuem um identificador único, calculado pelo sistema, sem conter atributos próprios. Entretanto, essas propostas apresentam algumas características distintas. Segundo BENATALLAH, esquemas, classes e objetos são representados individualmente pelo conceito de entidade, visto que tudo é versionável. Em FARANDOLE 2, apenas esquemas são versionáveis, apresentando uma identificação única para o objeto genérico, sendo o nome definido pelo usuário.

3.3.2 Estrutura dos Relacionamentos

As versões estão organizadas logicamente a fim de refletir seus relacionamentos. A estrutura de derivação de versões é identificada e analisada, distinguindo-se:

- lista: cada versão possui apenas uma versão antecessora e outra sucessora;
- árvore: uma versão pode possuir inúmeras versões sucessoras mas somente uma antecessora; e
- grafo acíclico dirigido: o número de versões antecessoras e sucessoras para uma determinada versão é ilimitado, desde que não haja ciclos.

Com relação à estrutura dos relacionamentos, as propostas apresentam-se bem balanceadas. KIM [KIM88] apresenta uma estrutura em forma de árvore. BENATALLAH [BEN94, BEN95a, BEN95b] permite a representação através de uma cadeia linear, isto é, um grafo em forma de lista. Segundo ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] a estrutura dos relacionamentos é chamada de cadeia linear, porém permite múltiplas folhas na hierarquia de derivação, caracterizando-se por uma estrutura em forma de árvore. Por fim, FARANDOLE 2 [AND91], FORNARI [FOR93a] e LAUTEMANN [LAU97a, LAU97b] optaram pelo grafo acíclico dirigido.

Todas as propostas podem ser comparadas com o modelo de versões adotado [GOL95a], sendo que FARANDOLE 2 e FORNARI possibilitam a representação da evolução de esquema de forma mais flexível e real, pois admitem combinações de versões no momento da derivação, ou seja, uma versão pode possuir mais de uma antecessora, como também versões distintas podem ter origem comum.

3.3.3 Tipos e Estados das Versões

As versões são geralmente classificadas a fim de refletir seu estágio de desenvolvimento e/ou consistência e para que operações possam ser definidas sobre elas. Conforme [GOL95a], as versões apresentam estados (*status*) que refletem esse estágio de desenvolvimento. Uma versão em trabalho é uma versão temporária, podendo sofrer tanto alterações como ser removida. Uma versão estável não pode sofrer alterações, apenas ser compartilhada ou removida. E uma versão consolidada é aquela que atingiu seu estágio final de desenvolvimento, não podendo ser alterada nem removida, somente compartilhada.

FARANDOLE 2 [AND91] e FORNARI [FOR93a] classificam as versões em estados, considerando-os em trabalho e estáveis. Alterações podem ser realizadas livremente nas versões em trabalho, sendo que as estáveis não podem ser alteradas nem removidas. Segundo KIM [KIM88], as versões apresentam dois estados: transitória, podendo sofrer alterações e ser removida e em trabalho, podendo ser excluída mas não atualizada.

3.3.4 Definição de Versão Corrente

É identificada a existência de versão corrente e as técnicas adotadas para sua escolha, por exemplo, a presença de um critério padrão, podendo ou não ser modificado pelo usuário.

Em ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] não há uma noção rigorosa com relação à versão mais recente de esquema, pois uma versão pode apresentar múltiplas folhas, e para toda manipulação deve ser especificado o identificador da versão. Em BENATALLAH [BEN94, BEN95a, BEN95b] a versão corrente é sempre a última criada e as demais representam o histórico das modificações. Em FARANDOLE 2 [AND91] a versão corrente é sempre escolhida pelo usuário e em KIM [KIM88] é aquela em uso pela aplicação.

Em FORNARI [FOR93a] a versão corrente é considerada a mais recentemente criada, podendo ser alterada pelo usuário. Entretanto, a mudança da versão corrente de uma classe implica na mudança da versão corrente nos demais níveis da hierarquia devido à existência do conceito de contexto de esquema que abrange todas as versões correntes do grafo de derivação.

3.3.5 Ligação de Versão a um Objeto Composto

Os métodos de resolução quanto às referências ao objeto versionado são analisados, caracterizando-se por: ligações estáticas, que representam uma referência a uma versão específica do objeto; e ligações dinâmicas que representam uma referência ao objeto versionado, podendo indicar qualquer uma de suas versões.

FARANDOLE 2 [AND91] e KIM [KIM88] são as mais abrangentes, permitindo tanto ligações estáticas quanto dinâmicas.

ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] e FORNARI [FOR93a] apresentam apenas ligações estáticas, porém com algumas diferenças. Em

FORNARI todas as operações são realizadas na versão corrente e em ODBERG é preciso indicar a versão a ser manipulada através do identificador de esquema.

O método adotado por BENATALLAH [BEN94, BEN95a, BEN95b] é o mais limitado, pois admite apenas ligações dinâmicas. Somente o objeto versionado é manipulado e as versões são controladas no nível do sistema. A versão de esquema utilizada é sempre a última. Para classes, funções especiais determinam quais versões estão sendo utilizadas pelos programas no momento da compilação. O processo de gerenciamento de versões é transparente aos usuários.

3.3.6 Criação de Versões

Examina as formas de criação das versões: explícita, onde o controle é mantido pelo usuário e implícita, quando gerenciado pelo sistema.

BENATALLAH [BEN94, BEN95a, BEN95b] permite a criação de versões implícita e explícita. Por *default*, uma nova versão é criada quando a evolução do esquema provoca a perda de informações e/ou incompatibilidade dos programas aplicativos, podendo essa regra ser modificada pelo usuário.

Em FARANDOLE 2 [AND91] a derivação de uma nova versão é sempre resultado de uma decisão humana. Dessa forma, nem toda alteração do esquema provoca a derivação de novas versões.

KIM [KIM88], ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] e FORNARI [FOR93a] permitem apenas a criação implícita, pois novas versões são geradas pelo sistema após a realização das alterações na versão antecessora.

3.3.7 Exclusão de Versões

São apresentadas e comparadas as restrições presentes nos modelos quanto à exclusão de versões e conseqüente atualização do grafo de derivação.

BENATALLAH [BEN94, BEN95a, BEN95b], ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] e FORNARI [FOR93a] não permitem a exclusão de versões, pois estas são geradas pela ocorrência de modificações do esquema para representar o histórico da evolução.

Em FARANDOLE 2 [AND91] é possível excluir apenas versões que encontram-se no estado em trabalho. Seus componentes são recursivamente excluídos e as ligações existentes com as versões das quais é derivada são eliminadas. KIM [KIM88] permite a exclusão de versões classificadas como transitórias, restringindo a exclusão a versões folha na hierarquia de derivação.

3.4 Conceitos Associados ao Mecanismo de Evolução de Esquemas

3.4.1 Granularidade do Versionamento

Considerando as propostas de evolução de esquemas, o requisito de granularidade determina as unidades sujeitas ao versionamento, como – esquemas, classes, objetos e/ou métodos. O conceito de versão é também analisado.

Segundo KIM [KIM88] e LAUTEMANN [LAU97a, LAU97b] esquemas e objetos são versionáveis, representando o estado evolucionário do esquema, diferindo, entretanto, no tratamento das versões de objetos analisadas na seção 3.4.5.

Em BENATALLAH [BEN94, BEN95a, BEN95b], esquemas, classes e objetos são representados de maneira diferente por seus identificadores, mas a todos é permitida a característica de possuir versões. Novas versões, tanto de esquema quanto de classe, são derivadas por mudanças ocorridas na última versão. As versões dos objetos são geradas pela propagação de evolução de classes.

Em FARANDOLE 2 [AND91] as unidades alteráveis são contextos. Um contexto é formado por uma seleção de classes previamente definidas no esquema. Apenas a estas visões de esquema é permitido o versionamento.

ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] permite o versionamento de esquemas e classes. A derivação de uma nova versão do esquema é provocada por modificações realizadas na estrutura das classes e as versões de classes são geradas por mudanças efetuadas em suas propriedades e/ou nos relacionamentos de superclasse.

Segundo FORNARI [FOR93a] objetos, classes e métodos são tratados de maneira uniforme, sendo todos considerados como objetos versionáveis.

3.4.2 Alterações Permitidas na Definição das Classes

As possíveis modificações, como inclusão, exclusão e troca de nome de propriedades, realizáveis na definição das classes, são citadas e comentadas.

Todas as propostas permitem a inclusão, exclusão e alteração no domínio dos atributos. Em ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d], uma troca de domínio é resultado de uma exclusão e posterior inclusão.

BENATALLAH [BEN94, BEN95a, BEN95b], FORNARI [FOR93a] e LAUTEMANN [LAU97a, LAU97b] permitem a realização de modificações na definição e no código dos métodos. Em FARANDOLE 2 [AND91] nada consta a respeito de métodos e em ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] a implementação é executada separadamente da definição das classes.

Uma característica peculiar presente em ODBERG é a possibilidade de compartilhamento de classes em diferentes versões de esquema. As propriedades e superclasses da primeira versão são herdadas e redefinições são permitidas.

3.4.3 Alterações Permitidas na Estrutura das Classes

São apresentadas e comparadas as modificações permitidas nos grafos de agregação e generalização, como inclusão/exclusão de relacionamentos de herança e composição, e inclusão, exclusão e troca de nomes de classes, isoladamente.

Todas as propostas permitem a inclusão e exclusão de classes. Com exceção de ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] que, após a exclusão de uma classe, suas subclasses precisam ser manualmente reconectadas nas classes ascendentes. Nas demais propostas, as subclasses são eliminadas recursivamente. Segundo FORNARI [FOR93a] fica a cargo do projetista escolher entre a remoção das subclasses ou torná-las subclasses diretas da superclasse da classe eliminada.

As propostas apresentam ainda a possibilidade de inclusão e exclusão de superclasse, sendo que em ODBERG não é possível modificar a classe raiz do contexto. E LAUTEMANN [LAU97a, LAU97b] permite também a modificação dos relacionamentos de herança entre as classes.

A proposta [FOR93a] é que apresenta um modelo mais abrangente, permitindo, além das modificações apresentadas, mover atributos e métodos de uma superclasse para as subclasses e vice-versa. Incorpora ainda mecanismos adicionais que auxiliam a transparência das alterações, tais como lista com propriedades que tiveram nomes alterados e definição de um valor padrão no acréscimo de parâmetros de um método.

Apenas o modelo FARANDOLE 2 [AND91] não permite a alteração de nomes. Essa restrição visa garantir a independência dos programas quando aplicados em várias versões de um mesmo contexto.

A proposta definida por ODBERG é a única que apresenta uma linguagem para definição de mudanças, denominada CSL – *Change Specification Language*.

3.4.4 Restrições quanto a Alterações no Esquema

É verificada a existência de restrições com relação às modificações no esquema, visando a assegurar a consistência das instâncias existentes na base de dados.

Somente FARANDOLE 2 [AND91] e FORNARI [FOR93a] definem restrições quanto à validade da base de dados frente às modificações do esquema. Em FARANDOLE 2, como não ocorre o versionamento das instâncias, as restrições preocupam-se com o compartilhamento dos objetos e validade da conexão entre as classes. Em FORNARI, são apresentados três tipos de invariantes: estruturais, comportamentais e de instanciação, que asseguram, respectivamente, a integridade das classes e relacionamentos de agregação/generalização, validade do conjunto de métodos e consistência das instâncias armazenadas.

3.4.5 Métodos Utilizados para Adaptação das Instâncias

As mudanças realizadas em esquemas conceituais precisam manter a consistência e a compatibilidade dos objetos presentes na base de dados. Assim, são apresentadas e comparadas as técnicas utilizadas no gerenciamento de evolução das instâncias vigentes no banco de dados frente às modificações realizadas no esquema, tais como:

- conversão: as instâncias antigas são convertidas para a nova definição da classe;
- emulação: uma versão para o objeto é criada para uma classe particular e calculada quando solicitada por outras versões de classes. Nesse caso, não é armazenada fisicamente, mas permanece ativa durante a vigência da classe;
- versionamento: toda derivação de versões de classes e/ou esquemas acarreta a criação de versões para as instâncias presentes na base de dados.

KIM [KIM88] utiliza o modelo de versões de objetos, definido para o ORION, para adaptação das instâncias presentes no banco de dados. A proposta de evolução de esquemas define o escopo de acesso a uma versão de esquema onde os objetos são herdados, podendo sofrer alterações em todas as versões de esquema.

BENATALLAH [BEN94, BEN95a, BEN95b], FORNARI [FOR93a] e LAUTEMANN [LAU97a, LAU97b] também adotam o método de versionamento de objetos, porém de formas distintas. Em FORNARI, como apenas classes são versionáveis, a criação de uma versão de classe, por ter sido alterada, leva à derivação de versões de todas as instâncias existentes naquele dado instante. Em BENATALLAH, além da técnica de versionamento, é utilizada a emulação. Através de alguns conceitos de métrica, a importância das versões é verificada. As consideradas importantes à aplicação são fisicamente armazenadas e as consideradas obsoletas são calculadas no instante em que a versão do objeto é solicitada. E em LAUTEMANN o mecanismo de propagação é baseado em funções de conversão que mapeiam o conteúdo dos objetos entre as diferentes versões de esquema.

Em FARANDOLE 2 [AND91] e ODBERG [ODB93, ODB94a, ODB94b, ODB94c, ODB94d] não existe um método para adaptação das instâncias. Em FARANDOLE 2 os objetos existentes na base de dados são compartilhados entre as diferentes versões de classe, isto é, cada versão de contexto fica associada com seus objetos relacionados. A criação de um objeto implica integrá-lo à versão quando existente na base de dados ou gerá-lo, caso contrário. ODBERG adota um método de representação compartilhada, mantendo uma classe comum com informações sobre as diversas versões da classe, mantida por Funções de Propagação de Mutação (MPF), cujo funcionamento é semelhante ao de gatilhos de bancos de dados.

3.5 Comparação entre as Propostas

As tabelas 3.1(a), 3.1(b) e 3.1(c), colocadas no final desse capítulo, apresentam um resumo comparativo, abordando as características envolvidas no processo de evolução de esquemas.

3.6 Considerações

Nesse capítulo foi realizada uma análise de diversas propostas que buscam soluções para evolução de esquemas conceituais em bancos de dados orientados a objetos através da utilização de versões. As características relevantes adotadas por essas propostas foram analisadas e comparadas, inclusive os requisitos do modelo de versões e necessidades de evolução de esquemas.

Os modelos examinados apresentam o mecanismo de versão como forma de tratar evolução de esquema. Sendo assim, esses conceitos diferem de uma proposta para outra, visto que cada uma procura suprir suas carências próprias.

Frente às comparações apresentadas, constata-se que o modelo de gerenciamento de versões pode ser considerado como uma evolução no modelo de banco de dados, pois certamente melhora a flexibilidade na utilização e alteração, abrindo novas perspectivas para o processo de projeto. Evolução de esquemas com versões tem sido objeto de vários trabalhos de pesquisa, entretanto nenhum mecanismo foi totalmente implementado nos sistemas para os quais foram propostos.

Dentro desse contexto, fica clara a necessidade de incorporar mecanismos que auxiliem a evolução de esquemas em bancos de dados orientados a objetos, a fim de facilitar o processo de evolução e garantir a integridade da base de dados.

Finalmente, cabe salientar que tanto as pesquisas quanto as propostas de implementação relativas a evolução de esquemas com o uso de versões continuam em desenvolvimento, comprovando assim, a necessidade de um mecanismo que auxilie o manuseio de modificações em bancos de dados.

TABELA 3.1(a) Comparação entre as propostas de evolução de esquemas

PROPOSTA DE EVOLUÇÃO DE ESQUEMAS COM VERSÕES						
Itens para comparação	***Won Kim***	***Farandole 2***	***M. R. Fornari***	***Erik Odberg***	***Boulean Benatallah***	***Stven-Eric Lautemann***
Tipos de herança	simples e múltipla	simples	simples e múltipla	simples e múltipla	simples e múltipla	simples e múltipla
Mecanismos de herança	por refinamento	por refinamento	por refinamento	por refinamento	por refinamento	por refinamento
Objeto Versionado (*)	sem atributos próprios	sem atributos próprios	nada consta	nada consta	sem atributos próprios	nada consta
Estrutura dos relacionamentos (*)	árvore	grafo acíclico	grafo acíclico	árvore	lista	grafo acíclico
Tipos e estados das versões (*)	transitória, em trabalho	em trabalho, estáveis	em trabalho, estáveis	nada consta	nada consta	nada consta
Definição de versão corrente	versão corrente é aquela em uso pela aplicação	versão corrente selecionada pelo usuário	última criada, podendo ser alterada pelo usuário	nada consta	última versão criada	nada consta
Ligação de versões a um objeto composto (*)	estática e dinâmica	estática e dinâmica	estática	estática	dinânica	nada consta

(*) Extraído de [GOL95a]

TABELA 3.1(b) Comparação entre as propostas de evolução de esquemas

PROPOSTA DE EVOLUÇÃO DE ESQUEMAS COM VERSÕES						
Itens para comparação	***Won Kim***	***Farandole 2***	***M. R. Fornari***	***Erik Odberg***	***Boulean Benatallah***	***Stven-Eric Lautemann***
Criação de versões (*)	implícita	Explícita	implícita	implícita	implícita e explícita	explícita
Exclusão de versões (*)	apenas versões transitórias e/ou folhas na hierarquia de derivação	apenas para versões em trabalho	não permite	não permite	não permite	nada consta
Granularidade do versionamento	esquemas e objetos	Esquemas	classes, métodos e instâncias	esquemas e classes	esquemas, classes e objetos	esquemas e objetos
Alterações na definição das classes	- inclusão e exclusão de propriedades	inclusão e exclusão de relacionamentos de composição	- inclusão e exclusão de propriedades - modificação na definição das propriedades	- inclusão e exclusão de atributos	- inclusão e exclusão de propriedades - modificação na definição das propriedades - modificação no código dos métodos	

(*) Extraído de [GOL95a]

TABELA 3.1(c) - Comparação entre as propostas de evolução de esquemas

PROPOSTA DE EVOLUÇÃO DE ESQUEMAS COM VERSÕES						
Itens para comparação	***Won Kim***	***Farandole 2***	***M. R. Fornari***	***Erik Odberg***	***Boulean Benatallah***	***Stven-Eric Lautemann***
Alteração na estrutura das classes	- inclusão e exclusão de classes - modificação no relacionamento de herança entre as classes	- inclusão e exclusão de classes - redefinição de superclasse e subclasse	- inclusão e exclusão de classes - inclusão e exclusão de superclasse e subclasse -mover propriedades da superclasse para subclasse e vice-versa	- inclusão e exclusão de classes - alterar superclasse das classes existentes	- inclusão e exclusão de classes - inclusão e exclusão de relacionamentos de herança	- inclusão, exclusão e renomeação de classes - inclusão e exclusão de superclasses
Restrições quanto a alterações no esquema	não são verificadas	define regras que impõem restrições as alterações a fim de garantir a consistência da base de dados	define invariantes de esquema que asseguram a consistência da base de dados	não são verificadas	não são verificadas	não são verificadas
Métodos utilizados para adaptação das instâncias	versionamento	compartilhamento das instâncias com todas as versões	versionamento	compartilhamento das instâncias com todas as versões	versionamento e emulação	versionamento

(*) Extraído de [GOL95a]

4 Um Estudo de Caso: O Sistema de Controle Acadêmico da Universidade Federal do Rio Grande do Sul

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. Inicialmente, a idéia básica do sistema e as fases de evolução pelas quais passou é exposta. A seguir, é apresentado um histórico analítico em que padrões e categorias de modificações são identificados. E, por fim, são relacionadas as facilidades que poderiam ser incorporadas a um modelo de evolução de esquemas e as considerações finais.[2]

4.1 O Sistema de Controle Acadêmico da UFRGS

Um dos principais objetivos desse estudo de caso consiste na identificação de requisitos e funcionalidades necessárias a um modelo de evolução de esquemas conceituais para bancos de dados. O estudo de um sistema em execução tem o intuito de analisar problemas significativos baseado em dimensões reais, visando a garantir a aplicabilidade e adequação da proposta a ser definida.

Dentro desse contexto, busca-se identificar padrões de alterações de esquemas realizados no sistema e modificações que não foram incorporadas, embora julgadas importantes no âmbito de um modelo de evolução de esquemas.

O estudo de caso considerado é o Sistema de Controle Acadêmico da Universidade Federal do Rio Grande do Sul [UNI80, UNI97a, UNI97b], UFRGS, denominado "Sistema Discente". Esse sistema foi desenvolvido pelo Centro de Processamento de Dados da UFRGS, pela qual é mantido até os dias atuais, sendo de uso exclusivo dos responsáveis. Sua implementação teve início no ano de 1973 e, desde então, mantém o registro eletrônico dos alunos matriculados na instituição, cursos e respectivos currículos, e as turmas oferecidas semestralmente. O sistema pode ser dividido em três principais subsistemas:

- Programação de Currículos. As disciplinas caracterizam-se por estruturas independentes, sendo oferecidas por departamentos. Podem ou não apresentar pré-requisitos. São classificadas em alternativas – disciplina rótulo que possui um auto-relacionamento com alternativas, e equivalentes – regra de transição responsável pelo encadeamento de disciplinas antigas que correspondem às disciplinas atuais. Os cursos são administrados por uma comissão de graduação, cuja tarefa é selecionar as disciplinas e compor os cursos. Para cada curso há apenas um currículo vigente, com validade de ano para ano. Após encerrado, o currículo não pode mais ser aberto para modificações. A grade curricular é um referencial para o aluno.
- Programação de Turmas. Apresenta as vagas oferecidas para cada comissão de graduação. Essa parte do sistema é encarregada pela composição dos horários das turmas nas salas. Uma turma é composta pela junção entre uma disciplina e uma sala de aula, formando as turmas de alunos. Existe ainda uma relação

[2] [HEU98, KOR93, NAV78, NAV92] Bibliografia suporte para realização desse estudo de caso.

com comissão de graduação, formando a comissão de graduação da turma, e com cursos, gerando os cursos da turma. Os usuários podem carregar currículos vigentes e futuros.

- Controle de Alunos. Esse subsistema consiste na matrícula do aluno em uma disciplina, denominada disciplina corrente. Cada disciplina apresenta um histórico. Os alunos são classificados dentro do semestre em função da disciplina mais atrasada, segundo a organização da grade curricular. O vestibular está relacionado a um curso, porém não é a única forma de ingresso na instituição. Por exemplo, o processo de convênio permite que alunos estrangeiros ingressem e freqüentem a universidade normalmente, mas, nesse caso, devem seguir a programação de currículos sem atraso. O controle de alunos apresenta ainda dois estados para o banco de dados, caracterizando os alunos em estado ativo e afastado.

4.2 As Fases de Evolução do Sistema de Controle Acadêmico da UFRGS

O Sistema Discente pode ser dividido em quatro fases distintas, caracterizando sua evolução ao longo do tempo. Essa evolução é mostrada graficamente na Figura 4.1. (As datas colocadas entre parênteses indicam o início de cada fase, tendo como término o começo da fase seguinte)

- FASE 1 (1972/1973): 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 operacional Burroughs, com modelo de máquina B-6700;
- FASE 2 (1987): apenas uma parte do Sistema Discente, referente à 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 às demandas de um modelo de banco de dados relacional;
- FASE 4 (1996/1997): implementada na linguagem PowerBuilder, em um banco de dados Sybase, sob o sistema operacional Unix, caracterizando-se por um banco de dados expressamente relacional.

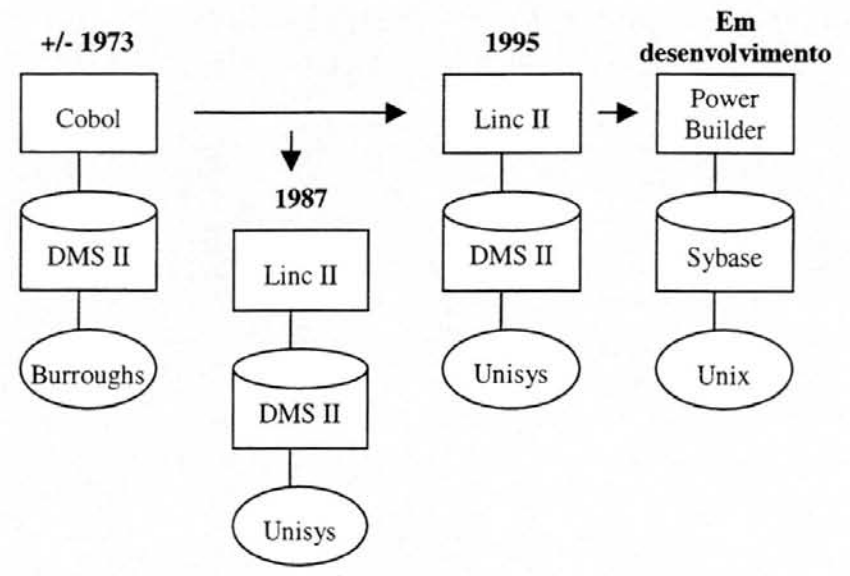


FIGURA 4.1 - Evolução do Sistema Discente da UFRGS

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

Na terceira etapa, 1995, o sistema apresentava-se dividido em dois bancos de dados, denominados Discente e Grafast. O Grafast caracterizava-se por ser o espelho de alunos do sistema Discente, representando os alunos afastados. Nessa fase, o retorno de um aluno afastado implicaria a cópia dos dados do aluno do Grafast para o Discente.

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. E, 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 do banco de dados.

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.

4.3 Modificações nos Módulos do Sistema

Como citado na seção 4.1, é possível distinguir três subsistemas para o Sistema Discente, porém se optou em dividi-lo em diversos módulos, com o intuito de realizar uma análise mais detalhada das modificações. A comparação é realizada, principalmente, entre as fases 1 e 3, pela quantidade de documentação disponível. Cabe salientar, que a análise da última etapa não foi documentada pelo fato de estar em constante modificação e por se tratar de um trabalho em andamento.

Para cada módulo, são identificadas as principais modificações relacionadas à semântica do Sistema Discente da UFRGS.

Em Alunos, conforme ilustrado no diagrama ER na Figura 4.2, a mudança mais evidente foi a criação de uma nova entidade, denominada *Disciplinas do Aluno*, generalizando tanto entidades existentes, dentre elas *Disciplina Corrente*, como novas entidades, por exemplo, *Disciplina Futura*. Houve também a inclusão de relacionamentos com novas entidades, como *Convênio* e *Microfilmagem*. A entidade principal desse módulo, chamada *Alunos*, foi composta pela união, *join*, entre três entidades existentes: *Common1*, *Common2* e *Alunos*. Os motivos de ingresso e afastamento, inicialmente controlados por programas, passaram a ser representados por duas tabelas no banco de dados. Nos atributos, foi identificada a inclusão de novos e mudanças de domínio, como aumento de tamanho e mudanças de tipos.

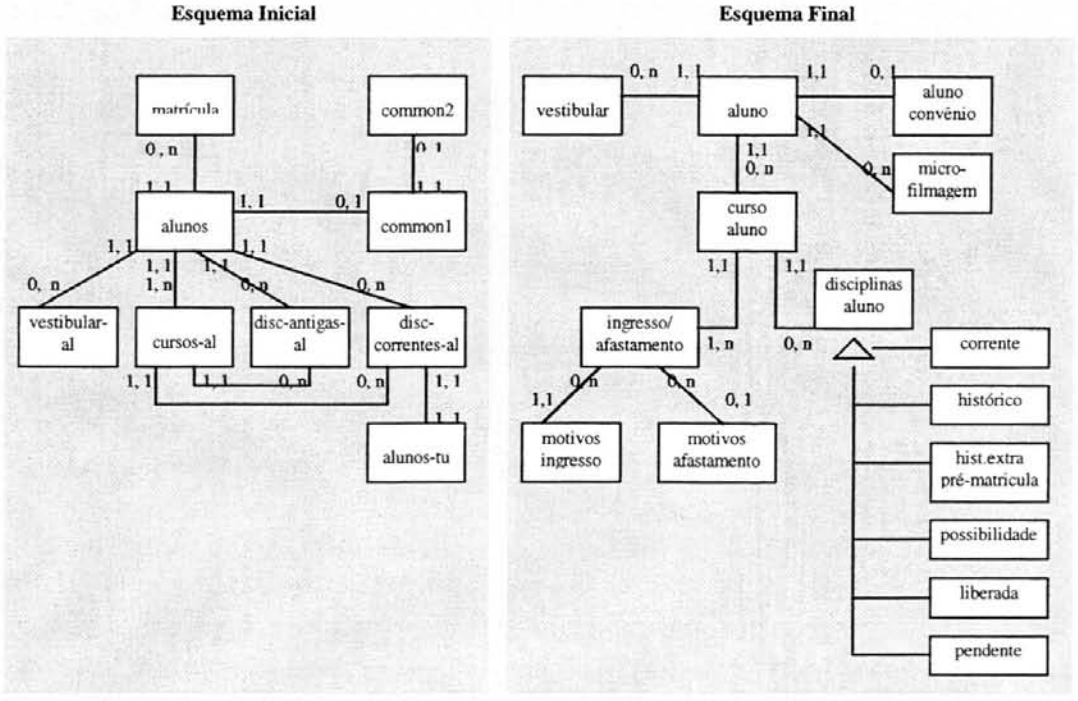


FIGURA 4.2 - Evolução do módulo "Alunos"

O módulo Comissão de Graduação, inicialmente conhecido por Comissão de Carreira, manteve basicamente a mesma semântica. Novos atributos foram incluídos e outros sofreram mudanças de domínio, como aumento do tamanho definido e alterações de tipo. Essa entidade pode ser observada tanto na Figura 4.3 quanto na Figura 4.4.

No módulo Cursos, com o objetivo de normalização, a entidade *Disciplinas do Currículo* foi dividida em duas novas entidades: *Dicur*, caracterizando as disciplinas do currículo, e *Semdc*, semestre recomendado para disciplina. Tecnicamente, na implementação atual, a entidade *Semdc* foi abandonada, porque, em todos os currículos, somente dois se valiam dessa característica. Novas entidades e relacionamentos foram incluídos, tais como colégio e vetor de cursos. A evolução do módulo de cursos pode ser observada na Figura 4.3.

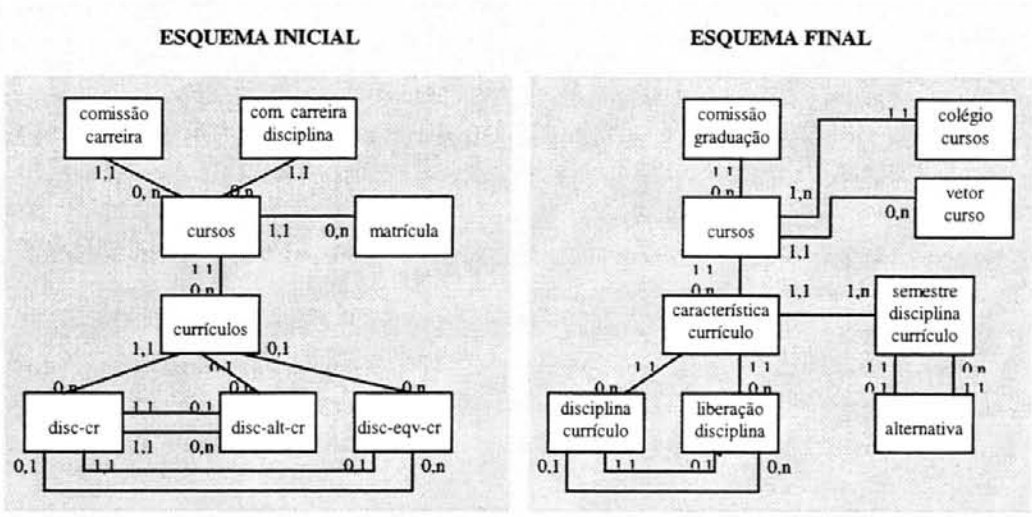


FIGURA 4.3 - Evolução do módulo "Cursos"

A semântica de Departamentos permaneceu inalterada (Figura 4.4). Alguns atributos foram incluídos e um auto-relacionamento acrescentado, sendo esse responsável pela conversão de código devido ao aumento no tamanho do atributo chave, com objetivo de aproveitamento da base de dados existente.

Em Disciplinas, a principal alteração ocorrida foi a criação de uma nova entidade, denominada *Pré-Requisitos*, formada por elementos provenientes da própria entidade *Disciplina*, eliminando assim, atributos multivalorados. A entidade *Disciplina* passou a ser composta pela união, *join*, de duas entidades com características comuns: *Disciplinas* e *Características da Disciplina*. A entidade *Comcares-di* passou a ser representada por um relacionamento entre as entidade *Disciplina* e *Comissão de Graduação*.

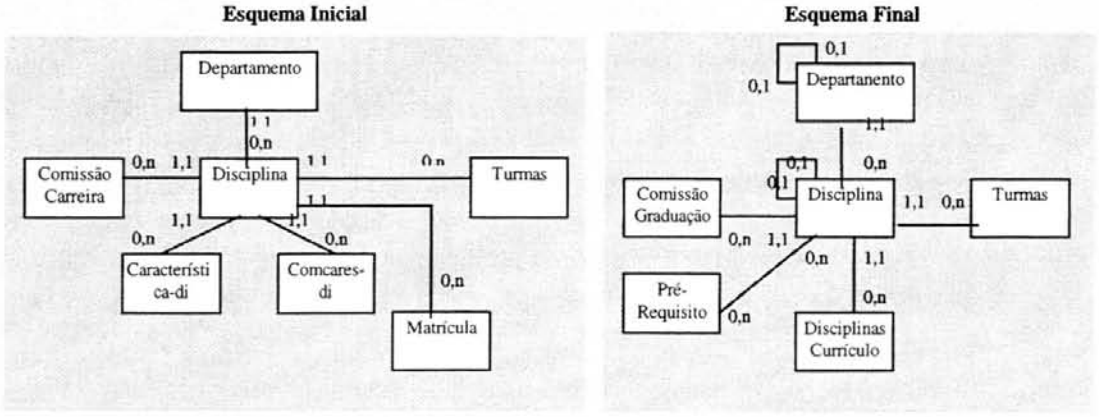


FIGURA 4.4 - Evolução do módulo "Disciplinas"

Em Edifícios (Figura 4.5) a mudança evidente foi a criação de um relacionamento entre as entidades *Turmas* e *Salas*, formando o *Horário das Turmas nas Salas*. Inicialmente, representado por uma entidade denominada *Salas-slot-ed*, composta por atributos multivalorados.

O módulo Histórico das Disciplinas, responsável pela representação das vagas oferecidas e ocupadas nos cursos, foi desativado, por não estar em uso desde a primeira implementação.

O módulo Matrícula foi eliminado, passando a ser representado por estruturas auxiliares, na forma de tabelas de apoio, tais como: *Solicitação de Matrícula em Curso2*, *Solicitação de Turma Não-Programada*, *Impressão de Comprovante*, *Controle de Matrícula* e *Controle de Pré-Matrícula*. Essa decisão foi tomada pela necessidade de normalização dos dados.

O módulo Turmas foi o que apresentou visíveis e significativas alterações quanto à semântica do sistema. Inicialmente, uma turma era identificada pelo código da turma, código da disciplina e semestre de vigência. A partir do segundo semestre de 1994, além dessas informações, passou a ser identificada pelo ano/semestre correspondente. Por conseguinte, o acervo dos dados passou a ser guardado, armazenando a história das turmas, visto que, a princípio, a cada ano, os dados referentes às turmas eram substituídos pelos novos valores. Houve também, uma pequena mudança entre os relacionamentos onde *Cursos da Turma*, inicialmente, representados por uma entidade passou a ser um relacionamento entre as entidades *Cursos* e *Turmas*. Da mesma forma, como mantinha um relacionamento com *Comissão de Graduação*, um novo relacionamento entre *Turmas* e *Comissão de Graduação* foi criado, gerando *Comissão de Graduação da Turma*.

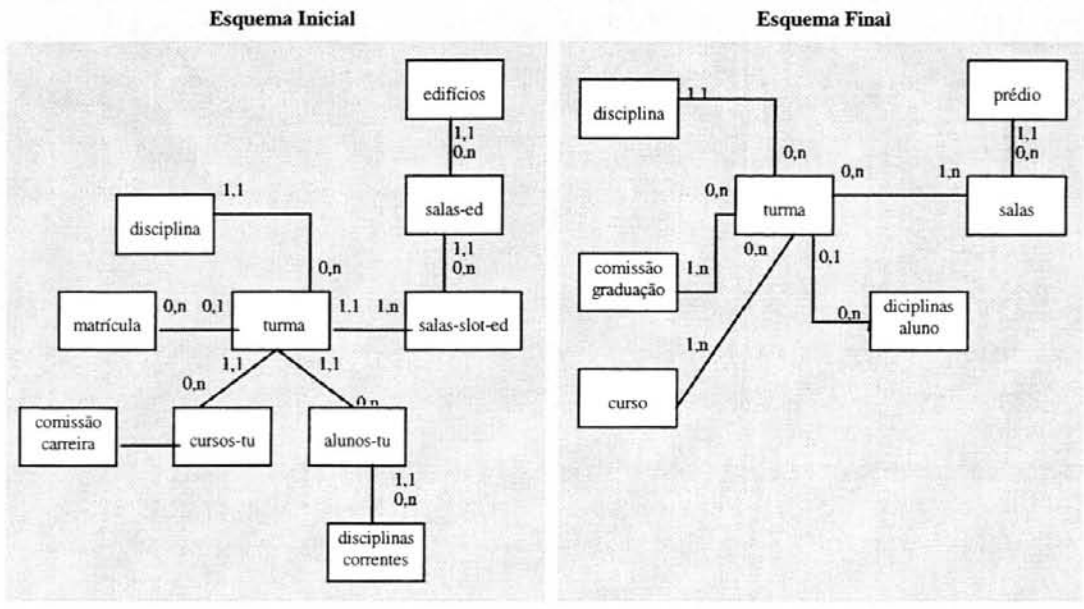


FIGURA 4.5 - Evolução do módulo "Turmas"

Além das modificações apresentadas, características relevantes, relacionadas à evolução do sistema de uma forma geral, são identificadas a seguir.

A maioria das entidades apresentaram inclusão de novos atributos, como também atributos existentes sofreram modificações em seus domínios, tais como alterações de tipo e aumento na definição de tamanho. A identificação do ano, nos atributos data, passou de duas para quatro posições. Essa providência está relacionada ao tratamento das datas a partir do ano 2000.

Grande parte das alterações realizadas no Sistema Discente, principalmente, a criação de novas entidades e inclusão de alguns relacionamentos, teve por objetivo solucionar problemas quanto ao armazenamento de atributos multivalorados, isto é, a normalização dos arquivos existentes, representando uma reorganização do banco de dados.

Todas as entidades e atributos que migraram de uma fase para outra tiveram seus nomes alterados por motivos de padronização. Outro fator que impulsionou esta decisão foi a limitação imposta pela ferramenta de implementação utilizada, Linc II, que permite, no máximo, o uso de dez caracteres na definição dos nomes. Esse procedimento implica tanto a verificação, quanto modificação das referências existentes para a entidade e/ou atributo. Surge, também, a necessidade de garantir a unicidade de nomes.

Até a fase de 1995, nada foi excluído. Isso deu-se pela falta de um total conhecimento a respeito da funcionalidade de todas as entidades e atributos, devido à existência de inúmeras peculiaridades e à grande extensão do sistema. Todo o trabalho realizado até 1995 proporcionou um amplo conhecimento do sistema, de forma que as eliminações estão sendo realizadas na fase atual de evolução. Nesse ponto, é possível salientar, que o tratamento com versões permite guardar todo o histórico da evolução, no qual tudo o que é excluído continua presente nas versões anteriores, não sendo extraído por completo do banco de dados.

Na passagem de uma fase para outra, a base de dados foi totalmente aproveitada. Isto foi possível porque as modificações caracterizaram-se por inclusões e nunca por reduções ou exclusões. Esse procedimento teve a pretensão de manter a consistência das informações armazenadas e respectiva correspondência dos dados com o esquema definido. Entretanto, surgiram alguns problemas que exigem a utilização de recursos adicionais para o total aproveitamento da base de dados.

Alguns programas de carga foram desenvolvidos com o intuito de converter os valores dos atributos que sofreram alterações em seus tipo. Por exemplo, o atributo responsável em indicar se o aluno está vinculado à instituição por convênio ou não possuía domínio "0" ou "1" na fase inicial do sistema. Após a evolução, seu domínio passou para "S" ou "N". Esses programas converteram os dados a fim de garantir a consistência com o novo esquema definido.

Houve ainda, a criação de auto-relacionamentos em três entidades. Na entidade *Cursos* são dois auto-relacionamentos: o de substituição e o de sucessão. O de substituição serve para ligar os cursos na conversão feita no primeiro semestre de 1995, onde o atributo passou de 4 para 5 posições. O de sucessão destina-se à readmissão de alunos após trancamento. Tanto na entidade *Departamento* quanto em *Disciplina*, os auto-relacionamentos introduzidos foram responsáveis pela conversão de atributos que tiveram seus tamanhos acrescidos. Em *Departamento* o atributo chave passou de 4 para 5 posições e, em *Disciplinas*, de 6 para 8.

Com relação aos programas, nada foi reaproveitado, tudo foi implementado novamente. Isso foi possível devido às facilidades proporcionadas pela ferramenta utilizada, Linc II, que permitiu a realização dessa tarefa em curto prazo e de forma eficiente. A única exceção foi a reutilização do programa "Recepção das Folhas de Conceito", que permanece implementado em Cobol, embora tenha sido adaptado para

fazer acesso ao novo banco de dados. Ele prevaleceu devido às peculiaridades existentes na recepção de folhas ópticas.

Outra mudança relevante foi a passagem de uma abordagem hierárquica para uma modelagem relacional. Por conseguinte, os apontadores de referência responsáveis pelas ligações físicas entre os registros, inicialmente definidos no esquema, deixaram de existir. A ferramenta Linc II proporcionou uma grande facilidade, à medida que controlou automaticamente todos os relacionamentos de integridade referencial.

4.4 Os Padrões de Alteração Identificados no Sistema de Controle Acadêmico

A análise do Sistema Discente permitiu a identificação de novos padrões de mudanças, não previstos na bibliografia examinada no capítulo 3 e apresentados a seguir.

4.4.1 Alterações em Entidades

4.4.1.1 Criação de uma Entidade

A operação de inclusão de uma nova entidade no banco de dados implica tanto sua própria criação quanto seus respectivos atributos. A princípio, a única restrição refere-se à validação de unicidade de nomes, ou seja, garantir a inexistência de mais de uma entidade com o mesmo nome.

A introdução de uma nova entidade vem, freqüentemente, acompanhada da inclusão de suas propriedades. No caso do Sistema Discente, por se tratar de um modelo relacional, apenas a inclusão de atributos foi identificada.

A composição de uma entidade pode se dar por várias formas distintas:

- Uma entidade pode ser formada pela inclusão de novos atributos, não definidos ainda no banco de dados. Nesse caso, a restrição existente refere-se à unicidade de nomes;
- Uma entidade pode ser composta por atributos existentes no banco de dados. Por exemplo, a entidade *Pré-Requisito* foi formada por atributos pertencentes, inicialmente, à entidade *Disciplina*. Nesse caso, a operação implica a exclusão do atributo da entidade inicial e posterior inclusão na nova entidade. Os valores armazenados na base de dados precisaram ser convertidos para que a migração fosse executada;
- Uma entidade pode ser constituída por atributos oriundos de mais de uma entidade. Por exemplo, a entidade *Alunos*, foi composta pela união dos atributos de três entidades: *Alunos*, *Common1* e *Common2*. Nesse caso, existe a necessidade de realizar uma união, *join*, a fim de eliminar possíveis redundâncias. Um problema observado no Sistema Discente, que representa um cuidado a ser tomado, refere-se à existência de atributos com nomes iguais, mas domínios diferentes, como, também, atributos com nomes distintos, mas domínios idênticos. Para tanto, é essencial um bom

conhecimento da aplicação, a fim de garantir a integridade do sistema frente às modificações.

4.4.1.2 Exclusão de uma Entidade

Esse tipo de alteração foi identificado somente na última fase de evolução, entretanto foi julgado como uma operação importante, pois existe sempre a necessidade de eliminar peculiaridades que não estão sendo utilizadas.

Embora seja uma operação fundamental, deve ser cuidadosamente analisada. A exclusão de uma entidade implica a eliminação de todos os seus atributos. Com isso, surge a necessidade de verificar todas as referências existentes para a entidade em questão e suas respectivas propriedades. Assim, de acordo com a aplicação, as referências existentes devem ser eliminadas juntamente com a entidade, ou redirecionadas para outra entidade e/ou atributo.

No caso do Sistema Discente, por se tratar de um modelo relacional, a eliminação de uma entidade implicou garantir os relacionamentos de integridade referencial. Foi preciso decidir se as entidades dependentes seriam eliminadas juntamente com a principal ou se o processo de exclusão seria interrompido. O mesmo aconteceu com os grafos de especialização/generalização. As entidades mais especializadas poderiam ser excluídas ou, simplesmente, reconectadas à entidade superior.

4.4.2 Alterações em Relacionamentos

4.4.2.1 Inclusão de Relacionamento

A inclusão de um relacionamento implica a existência de duas entidades, precisando assim, ser verificada sua presença no banco de dados. No caso de inexistência, deve-se assumir, primeiramente, o processo de inclusão de entidade.

A introdução de um relacionamento pode apresentar-se de duas formas: o relacionamento gera uma nova entidade ou apenas impõe uma relação de dependência entre as duas entidades existentes. No primeiro caso, tanto novos atributos podem ser incluídos, como também atributos presentes no banco de dados, utilizados. No segundo caso, é preciso apenas verificar a existência dos identificadores nas entidades que se relacionam.

4.4.2.2 Exclusão de Relacionamentos

A exclusão de um relacionamento representa uma alteração na semântica definida para o sistema, devendo ser cuidadosamente analisada.

É preciso averiguar se os atributos definidos para compor o relacionamento podem ser excluídos ou se eles apresentam outras funções dentro da entidade.

Uma verificação consiste em investigar se a exclusão do relacionamento não representa um rompimento na integridade referencial definida, pois poderá deixar a base de dados inválida. Nas fases iniciais, essa restrição precisava ser controlada pelo

implementador, mas a partir da fase 3, a ferramenta utilizada para o desenvolvimento do sistema passou a assegurar essa consistência.

4.4.2.3 Mudanças de Relacionamentos entre Entidades Existentes

A mudança de relacionamento entre entidades existentes ocasiona os problemas citados anteriormente, pois representa a exclusão de um relacionamento existente e sua posterior inclusão em outra entidade.

4.4.2.4 Introdução de Auto-relacionamentos

O objetivo desse tipo de alteração foi, simplesmente, executar o mapeamento entre atributos de uma fase para outra, após as modificações em domínios. Essa alteração não acarreta nenhum problema ao esquema definido, pois seu único objetivo é garantir a integridade das informações armazenadas.

4.4.3 Alterações em Atributos

4.4.3.1 União de Dois Atributos

Essa alteração caracteriza-se pelo agrupamento, união, do conteúdo de dois atributos em apenas um. Por exemplo, a matrícula do aluno, inicialmente, representada por dois atributos, *código*, com seis posições, e *dígito de controle*, com uma posição, passou a ser representada por um único atributo com sete posições.

O principal problema gerado por esse tipo de mudança é garantir a integridade da base de dados. No caso do Sistema Discente, os programas de carga foram responsáveis em realizar todas as conversões necessárias.

4.5 Requisitos que Poderiam ser Incorporados no Sistema de Controle Acadêmico da UFRGS

Inicialmente, por se tratar de um modelo hierárquico, grande parte das regras referentes à aplicação e informações armazenadas no banco de dados não estavam definidas no esquema, dificultando, ou até mesmo, tornando impossível, sua identificação. Surge a necessidade de mecanismos que auxiliem o projetista no processo de conhecimento da realidade da aplicação como um todo, a fim de garantir uma maior segurança e confiabilidade na definição das etapas de evolução. Um exemplo presente na evolução do Sistema Discente foi a impossibilidade de realização de exclusões nas fases iniciais, justamente pela falta de conhecimento seguro a respeito da importância e utilidade de determinadas informações. Torna-se visível, que com o aumento gradativo do conhecimento, no decorrer dos trabalhos, as eliminações começaram a ser realizadas. O problema aparente é a lentidão no processo de conhecimento. A existência de métodos que guiassem o usuário no processo de compreensão não implicaria a perda do conhecimento na substituição de um responsável.

Outra dificuldade encontrada foi quanto ao aproveitamento das informações presentes no banco de dados. A troca de esquema não pode ocasionar perda de

informações, pois eles representam todo o histórico da aplicação. Emerge a necessidade de estabelecer regras para garantir não apenas a validade dos dados armazenados, mas também assegurar a coerência do esquema no final da modificação.

Todas as transições efetuadas no Sistema de Controle Acadêmico da UFRGS 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, seria interessante a existência 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 grandemente as fases de evolução, a definição de padrões de modificações permitiria uma melhoria na qualidade do trabalho.

4.6 Considerações

A partir desse estudo de caso, algumas comparações podem ser feitas com o modelo de versões [GOL95a], utilizado como base para o desenvolvimento desse trabalho.

Segundo o estudo realizado, constatou-se que, para cada curso, existe um currículo vigente com validade ano/semestre, que, 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, pois, segundo [GOL95a], a versão corrente é utilizada quando o objeto é solicitado sem a especificação de uma de suas versões. No Sistema Discente, a cada ano, um currículo, preparado e selecionado pelo usuário, permanece atual para o curso. E, ainda, o fechamento do currículo pode ser comparado à passagem da versão para o estado consolidado, pois, após encerrado, não sofre mais alterações. Em [GOL95a], uma versão é considerada consolidada quando atingiu seu estado final de desenvolvimento, não podendo ser alterada nem removida.

A heurística utilizada na implementação dos relacionamentos de generalização/especialização foi herança por refinamento, onde os atributos migram para as entidades mais especializadas. Essa abordagem foi selecionada pela necessidade de limitar o número de tabelas, acomodando os dados na entidade mais especializada, 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, herança por extensão, em que as propriedades permanecem no nível em que foram definidas. Essa característica, presente no modelo de versões [GOL95a], facilita o manuseio de evolução de esquemas, possibilitando modificar arbitrariamente determinados níveis de abstração da estrutura de dados sem preocupar-se com os demais, garantindo, a coerência e 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 análise de esquemas paralelos, ajudando na escolha da melhor forma de se tratar um padrão de evolução em questão.

Versões 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 é possível identificar alterações realizadas em etapas intermediárias às fases definidas, devido à substituição das mudanças realizadas e conversão da base de dados.

Conforme citado anteriormente, os programas foram todos desenvolvidos novamente. Convém enfatizar que esse processo de implementação foi possível devido às inúmeras facilidades proporcionadas pela ferramenta de quarta geração utilizada, Linc II, que proporcionou um rápido, eficiente e confortável desenvolvimento.

Por fim, as modificações realizadas na transição entre uma etapa e outra tiveram grande impacto mas não afetaram de forma radical a semântica definida para o sistema. As alterações visaram a incluir melhorias e reorganizar o banco dados e não a mudar os objetivos propostos para o sistema. Ainda assim, as modificações procuraram manter o máximo possível a compatibilidade entre uma fase e outra, a fim de assegurar a consistência e garantir o aproveitamento das informações existentes.

5 Modelo de Dados e Modelo de Versões

O objetivo deste capítulo é apresentar o modelo de dados e versões nos quais se baseia esse modelo. As principais características são identificadas com vistas às necessidades impostas pelo mecanismo de evolução de esquemas.

O modelo de dados é baseado nos princípios fundamentais de orientação a objetos, cujas características são descritas a seguir, porém somente os aspectos relevantes a proposta de evolução de esquemas são considerados.

O modelo de versões é definido sobre a proposta de Golendziner, descrita em [GOL95a], que estende a um modelo de dados orientado a objetos conceitos e mecanismos que suportam a definição e manipulação de objetos, versões e configurações. As características desse modelo são revistas na seção 5.2, a fim de estabelecer as premissas nas quais se baseou a proposta, considerando os aspectos posteriormente utilizados pelo modelo de evolução de esquemas proposto.

5.1 O Modelo de Dados

O modelo de dados utilizado 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 [ATK89] e, de forma semelhante, presentes em sistemas de banco de dados, como, por exemplo, O2 [DEU91, O2T91a,O2T91b]. O modelo 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, entre outros.

5.1.1 Identidade de Objetos

As entidades do mundo real são modeladas como objetos e possuem uma existência própria independente de seu valor. Cada objeto possui um identificador (OID: *object identifier*), definido pelo sistema, que garante sua unicidade mantendo a integridade referencial e permitindo sua manipulação pelos programas aplicativos [ATK89].

5.1.2 Objetos Simples e Complexos

Objetos complexos são construídos a partir de objetos simples pela aplicação de construtores sobre eles, tais como tupla, conjunto ou lista. Uma hierarquia de objetos é formada onde o objeto raiz é denominado objeto composto e, seus valores, objetos componentes. As operações em objetos complexos são propagadas recursivamente em todos os componentes [ATK89].

5.1.3 Classes

Classes agrupam objetos com as mesmas propriedades e comportamento. Os objetos são construídos pela execução de operações e conectados à classe como sua extensão, isto é, um conjunto de objetos é instância de uma classe [ATK89].

5.1.4 Relacionamentos de Herança

A proposta admite tanto herança simples, permitindo que uma classe contenha apenas uma subclasse única e direta, quanto herança múltipla, admitindo que uma subclasse possua mais de uma superclasse. Embora essa característica seja considerada opcional pelo autores do manifesto *Atkinson* [ATK89], diversos bancos de dados orientados a objetos, como, por exemplo, O2 [DEU91, O2T91a,O2T91b], apresentam herança múltipla por permitir uma modelagem mais natural da realidade.

Ao contrário dos mecanismos presentes na literatura que utilizam herança por refinamento, onde as propriedades migram para os níveis mais especializados da hierarquia de herança, esse modelo adota o mecanismo de herança por extensão, onde as versões são admitidas nos vários níveis da hierarquia de herança. Esse mecanismo de herança é adotado pelo modelo de versões [GOL95a], onde uma definição mais exaustiva pode ser encontrada.

5.1.5 Encapsulamento

Encapsulamento vem com a necessidade de distinguir entre a especificação e a implementação de uma operação, ocultando essas informações do usuário e definindo o comportamento lógico das instâncias pertencentes à classe. Nesse caso, o usuário conhece apenas a interface da classe, definida no conjunto de métodos, que manipula as instâncias da classe.

Dessa forma, um método é um programa executável, especificado em uma classe, que descreve parte do comportamento dos objetos da classe. A definição de um método possui uma parte de interface (assinatura), onde um conjunto de operações aplicáveis sobre os objetos é especificado, e uma parte de implementação (corpo), que descreve, em alguma linguagem de programação, a codificação de cada operação.

Assim, o encapsulamento provê uma certa independência lógica de dados, possibilitando a modificação da implementação da classe sem alterar sua interface, protegendo os programas que as utilizam contra as mudanças realizadas na sua codificação.

5.2 O Modelo de Versões

5.2.1 O Conceito de Versão e Objeto Versionado

Uma versão é a descrição de um objeto em um determinado período de tempo, ou sob um certo ponto de vista, cujo registro é importante para a aplicação considerada. Uma versão é 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 formando um objeto versionado (Figura 5.1), que é também um objeto de primeira classe (possui um identificador único) que mantém informações a respeito de suas versões. Um objeto que possui versão é dito objeto versionado, possui características próprias, apresentando propriedades que podem ser comuns a todas as versões. Cada versão faz parte de um único objeto versionado.

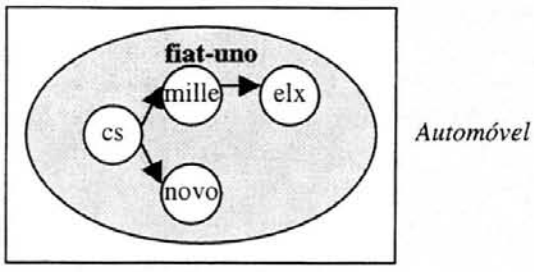


FIGURA 5.1 - Objeto Versionado e suas versões

Os objetos que possuem as mesmas propriedades e comportamentos são agrupados em classes. A característica de ser ou não versionado é de cada objeto e não da classe, pois uma classe pode conter tanto objetos versionados como não versionados.

Cada objeto versionado possui uma versão que é considerada a versão corrente. É mantida automaticamente pelo sistema como a última versão criada. No entanto, o usuário pode selecionar uma determinada versão como sendo a versão corrente de forma que permanece fixa mesmo com o surgimento de novas versões. A versão corrente é utilizada sempre que o objeto versionado é selecionado sem a especificação de uma de suas versões.

5.2.2 Propriedade das Versões

As versões de um objeto versionado estão relacionadas através de um relacionamento de derivação, formando um grafo acíclico dirigido. Uma versão pode ser criada como sucessora de uma existente ou derivada de várias outras. No primeiro caso, é uma cópia da antecessora. No segundo, é uma cópia da primeira indicada como antecessora, ficando sob responsabilidade do usuário a tarefa de extrair informações das outras versões a fim de que a nova seja uma combinação das diversas antecessoras.

Cada versão possui um *status* que reflete seu estágio de desenvolvimento e/ou consistência, podendo ser em trabalho, estável ou consolidada. Uma versão em trabalho é uma versão temporária podendo tanto sofrer alterações como ser removida. Uma versão estável não pode sofrer alterações, apenas ser compartilhada ou removida. E uma versão consolidada é aquela que atingiu seu estágio final não podendo ser alterada nem removida, somente compartilhada.

5.2.3 Composição e Objetos Complexos

Os objetos compostos podem ser construídos recursivamente formando uma hierarquia de agregação. Nesse caso, os atributos definidos para um objeto complexo contêm os OIDs dos objetos componentes.

Um objeto que apresenta versões pode ser componente de outro objeto. Referências a esses objetos podem ser estáticas ou dinâmicas. Uma referência estática indica uma versão específica do objeto e uma dinâmica faz referência ao objeto versionado, podendo indicar qualquer uma de suas versões. A Figura 5.2 mostra duas versões de um objeto composto da classe *Automóvel*. A versão *cs* possui uma referência estática para a primeira versão do objeto *motor-fiat*. O componente motor para a versão *cs* está completamente definido. A versão *novo* representa um automóvel sendo

projetado, pois a referência dinâmica indica que o motor ainda não foi definido, sabendo-se apenas que deve ser uma das versões *motor-fiat*.

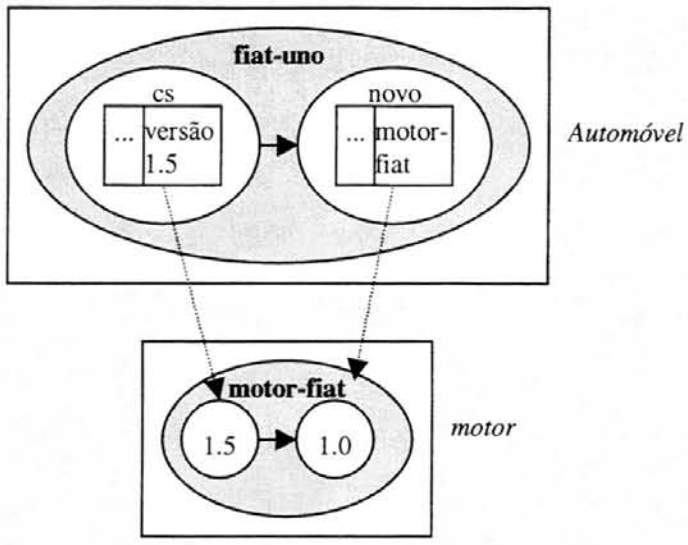


FIGURA 5.2 - Referências estáticas e dinâmicas

5.2.4 Hierarquia de Objetos e Versões

5.2.4.1 Herança

O modelo baseia-se no mecanismo de herança por extensão, em que as versões são admitidas em vários níveis da hierarquia de herança.

O mecanismo de herança por extensão está associado à idéia de protótipo, pois sendo 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.

Cada versão deve possuir, pelo menos, um ascendente que lhe corresponda. Dessa forma, quando uma versão é criada deve, obrigatoriamente, ser ligada a um ascendente.

5.2.4.2 Correspondência entre os Objetos e Versões

Como o modelo de versões proposto apresenta o mecanismo de herança por extensão, um objeto pode ser desenvolvido em um nível de abstração e posteriormente detalhado nos níveis inferiores da hierarquia de herança.

A representação de versões nos diversos níveis de hierarquia permite a existência de múltiplos ascendentes para um objeto (versionado, não versionado ou versão) em uma subclasse, devido à possibilidade de o objeto ascendente possuir múltiplas 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 correspondências podem ser classificadas em:

- 1:1 – cada versão na subclasse corresponde a exatamente uma versão na superclasse;

- 1:n – cada versão na subclasse pode corresponder a várias versões na superclasse e várias versões na superclasse podem corresponder somente a uma versão na subclasse;
- 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; e
- 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.

Quando é necessário buscar um objeto e seus atributos herdados, a busca inicia na classe mais especializada, sendo escolhido um ascendente para cada superclasse relacionada. O ascendente pode ser especificado pelo seu OID, ou através de um dos critérios pré-definidos: *recent* (mais recente), *first* (primeiro) ou *current* (corrente). O critério é usado sempre que houver mais de uma versão ascendente associada à versão (ou objeto) desejada(o).

5.2.5 Identificadores de Objeto

A correspondência entre os objetos ascendentes e descendentes na hierarquia é mantida pelo identificador do objeto, uma vez que uma entidade é representada em diferentes níveis da hierarquia de herança. O identificador possui um componente comum a todos os objetos que representa a entidade modelada. A estrutura proposta para o identificador é a seguinte:

OID = <id entidade, classe, número de versão>

Um objeto versionado possui número de versão nulo e um objeto não versionado possui número de versão igual a 1. Dessa forma, um objeto não versionado é identificado da mesma maneira que a primeira versão de um objeto versionado, podendo também evoluir para versionado sem exercer influência na identificação do objeto existente, que passa a ser a primeira versão.

5.2.6 Configurações

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.

5.3 Conceito de Versão Associado ao Modelo de Evolução de Esquemas

5.3.1 Derivação de Versões

A criação de uma nova versão de esquema, classe e/ou método 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. Assim, 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.

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.

5.3.2 Versões e Objeto Versionado

O modelo 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.

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

- versões de esquemas: são derivadas em decorrência de modificações realizadas na estrutura do esquema, como, por exemplo, a eliminação de uma classe desencadeia a derivação de uma versão completa do esquema representando a atualização aplicada;
- versões de classes: são geradas por alterações realizadas na definição das classes. Por exemplo, a introdução de uma propriedade em uma classe provoca a derivação de uma nova versão desta classe contendo a modificação efetuada;
- versões de métodos: são derivadas para registrar as modificações realizadas no código e definições dos métodos, entretanto inclusão e exclusão de métodos provoca a derivação de versões de classe, uma vez que a definição da classe é alterada;
- 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 classes e esquemas.

Sendo assim, uma versão de esquema especifica um esquema completo num dado instante de tempo e versões de classes servem como um ponto de partida para sua evolução durante o ciclo de vida do projeto.

5.3.3 Operações sobre as Versões

As versões de esquemas, classes, objetos e métodos possuem os mesmos *status* definidos pelo modelo de versões[GOL95a]: em trabalho, estável e consolidada.

Uma versão pode ser promovida de um estado para outro explicitamente pelo usuário, ou implicitamente pelo sistema. Uma versão é implicitamente promovida para o estado consolidado quando:

- apresentar versões sucessoras;
- apresentar um objeto na base de dados consolidado.

Versões em trabalho e estáveis podem ser alteradas, enquanto que versões consolidadas não. Alterações em versões consolidadas só podem ser realizadas gerando novas versões que representem essas atualizações. Nas versões em trabalho e estáveis, alterações podem ser realizadas a qualquer instante. Nesse caso, o usuário pode definir a criação de uma nova versão, consolidando a atual.

Remover uma versão significa eliminar esse estado da história de derivação de versões. Assim, a remoção de versões não é permitida, pois elas são geradas pela ocorrência de alterações para representar o estado evolucionário do esquema.

5.4 Considerações

Nesse capítulo foram apresentadas as principais características do modelo de dados e de versões [GOL95a] a ser utilizado ao longo desse trabalho e sobre o qual o modelo de evolução de esquemas foi proposto.

Considerando que o modelo de versões adotado foi proposto para versões de objetos, algumas características peculiares foram analisadas, considerando o contexto de evolução de esquemas. Assim, a derivação de versões abrange não somente os objetos da base de dados, mas também esquemas, classes e métodos. Entretanto, todos esses conceitos são tratados uniformemente como os objetos de forma que o modelo de versões pode ser perfeitamente utilizado no processo de retenção do histórico da evolução de esquemas.

6 Proposta de um Modelo de Evolução de Esquemas com Versões

Neste capítulo o modelo de evolução de esquemas é proposto como forma de guiar as alterações em bancos de dados orientados a objeto. 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.

O modelo tem por objetivo conduzir as modificações realizadas em sistemas de banco de dados orientados a objetos, mantendo o histórico das alterações através da derivação de versões. O estado anterior às transformações é mantido, permitindo a navegação retroativa e proativa entre as versões, para realização de operações de alteração e consulta. O modelo assegura ainda, a integridade das instâncias armazenadas no banco de dados em qualquer perspectiva de versão sob a qual são requeridas.

Nas seções subseqüentes, os requisitos propostos pelo modelo são detalhados. Restrições que visam a garantir a validade do esquema frente às modificações são apresentadas e a semântica das operações de atualização analisada e ilustrada. O mecanismo para adaptação das instâncias presentes no banco de dados é definido e, por fim, uma tabela compara as características do modelo definido com as propostas analisadas no capítulo 3.

6.2 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[3], 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.

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, tais como [AND91, BAN87, BRÈ96, FOR93a, ZIC91], adotam invariantes como forma de garantir a integridade perante as modificações. Esse modelo utiliza invariantes estruturais, comportamentais e de instanciação que asseguram, respectivamente, a integridade de classes e relacionamentos de agregação/generalização, validade do conjunto de métodos e consistência das instâncias armazenadas na base de dados com relação ao esquema definido.

---

[3] Os invariantes estruturais e comportamentais estão definidos em [BAN87] e os de instanciação, definidos em [FOR92]. Algumas adaptações são realizadas, pois o modelo adota o mecanismo de herança por extensão, diferindo das atuais propostas presentes na literatura.

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

- os relacionamentos de generalização devem resultar num grafo acíclico e conexo, com todas as classes ligando-se, no mínimo, à superclasse global, pré-definida no esquema;
- todas as classes existentes no esquema devem possuir nomes únicos;
- todos os atributos presentes em uma classe devem possuir nomes únicos, entretanto devido à utilização de herança por extensão, os conflitos de herança são resolvidos no instante em que a classe é solicitada;
- todos os atributos pertencentes ao conjunto de superclasses devem permanecer nos níveis em que foram definidos, exceto se houver uma redefinição do atributo nas subclasses;
- todos os atributos redefinidos nas subclasses devem possuir domínios mais especializados ou iguais ao de sua definição na superclasse; e
- todas as classes referidas como domínios de atributos devem existir.

Os Invariantes Comportamentais asseguram a integridade do conjunto de métodos declarados no esquema, estabelecendo que:

- todos os métodos existentes em uma classe devem possuir nomes únicos, sendo os conflitos de herança resolvidos no instante em que o método é solicitado pela classe;
- todos os métodos pertencentes ao conjunto de superclasses devem permanecer nos níveis onde foram definidos, exceto se houver redefinição do métodos na subclasse ou se a resolução de conflito de nomes excluir algum método; e
- todos os métodos redefinidos nas subclasses devem possuir comportamentos mais específicos ou idênticos ao de sua definição na superclasses.

Os Invariantes de Instanciação asseguram a integridade dos objetos armazenados no banco de dados em relação ao esquema definido, estabelecendo que:

- todos os objetos referidos devem estar presentes no banco de dados;
- os valores definidos para todos os atributos de uma versão de instância devem pertencer ao domínio definido para a versão de classe ao qual está associado ou são iguais 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.

6.3 Operações de Atualização

As operações de atualização realizáveis pelo modelo estão divididas em três blocos distintos: as modificações que alteram a estrutura das classes (seção 6.3.1); as modificações que alteram a estrutura do esquema (seção 6.3.2) e as modificações que alteram o comportamento das classes (seção 6.3.3).

6.3.1 Alterações na Estrutura das Classes: Atributos – Versões de Classes

A execução de alterações realizadas na estrutura das classes acarreta a derivação de versões de classe devido às modificações realizadas. Nesse caso, tanto a classe quanto os atributos devem estar presentes na mesma versão de esquema.

6.3.1.1 Incluir um Atributo

Permite a inclusão de um atributo, em uma determinada versão de classe, implicando a especificação de um domínio para o mesmo.

Por exemplo, a Figura 6.1[4] ilustra a introdução do atributo *área* na classe *Professor* (Professor, 1) e, por conseguinte, a derivação de uma nova versão para classe (Professor, 2).

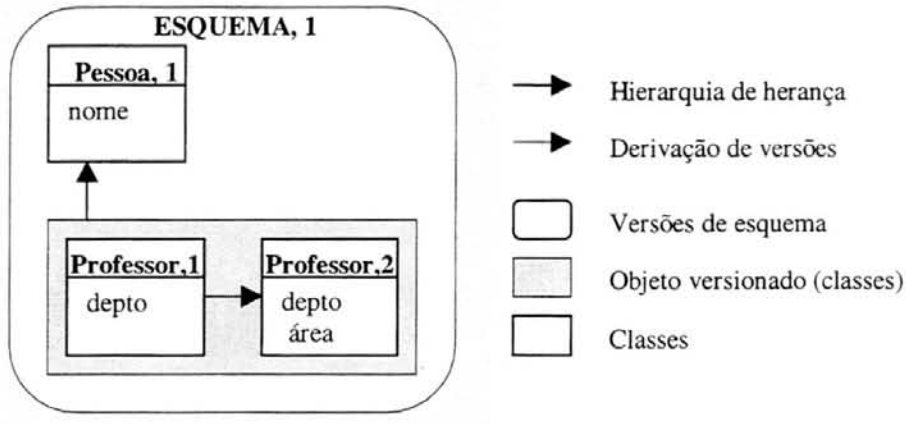


FIGURA 6.1 - Inclusão de atributo

A inclusão de um atributo deve garantir a unicidade de nomes, ou seja, quando existir, na versão da classe, um atributo com o mesmo nome daquele a ser inserido, a operação de atualização deve ser rejeitada e o usuário, notificado. Conflitos de nomes podem surgir na presença de herança múltipla; nesse caso, por *default*, o atributo selecionado é aquele presente na primeira superclasse definida para a versão de classe em questão. Entretanto, o usuário pode selecionar um atributo de outra superclasse. Devido à utilização de herança por extensão, a resolução do conflito é realizada no instante em que o objeto é requerido e não quando criado.

---

[4] A legenda definida para a Figura 6.1 é também utilizada para as figuras subseqüentes: Figura 6.2 a Figura 6.6, Figura 6.19 a Figura 6.21 e Figura 6.24.

6.3.1.2 Excluir um Atributo

Permite a exclusão de um atributo, definido em uma versão de classe.

Por exemplo, a Figura 6.2 ilustra a exclusão do atributo *área*, da classe *Professor* (Professor, 1) e, por conseguinte, a derivação de uma nova versão para classe (Professor, 2).

FIGURA 6.2 - Exclusão de atributo

6.3.1.3 Mudar o Nome de um Atributo

Permite trocar o nome de um atributo definido em uma versão de classe.

Por exemplo, a Figura 6.3 mostra a troca do nome do atributo *código*, presente na classe *Aluno* (Aluno, 1), pelo atributo *matrícula* e, em conseqüência, a derivação de uma nova versão para classe (Aluno, 2).

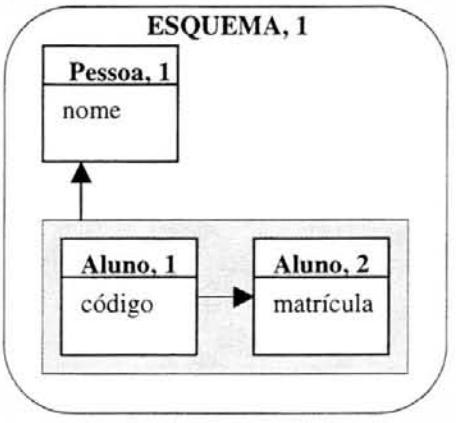


FIGURA 6.3 - Alteração no nome de um atributo

A alteração do nome de um atributo deve, obrigatoriamente, preservar a unicidade de nomes. Caso a classe possua um atributo com nome igual ao novo nome do atributo modificado, a operação de atualização deve ser recusada e o usuário, notificado.

6.3.1.4 Mudar o Domínio de um Atributo

Permite modificar o domínio de um atributo definido em uma versão de classe.

Por exemplo, a Figura 6.4 ilustra a mudança do domínio do atributo *bolsa*, presente na classe *Aluno* (Aluno, 1), do tipo numérico para o tipo alfanumérico e, por conseguinte, a derivação de uma nova versão para classe (Aluno, 2).

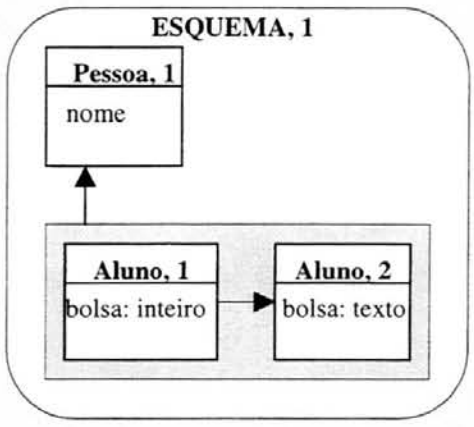


FIGURA 6.4 - Alteração no domínio de um atributo

6.3.1.5 Mudar o Valor Inicial de um Atributo

Permite alterar o valor inicial de um atributo, explicitamente definido no esquema, em uma versão de classe, acarretando a derivação de uma nova versão para a classe. Quando nenhum valor é especificado para o atributo, por *default*, os atributos são iniciados com valor nulo.

6.3.1.6 Incluir uma Ligação de Composição

Permite 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 assim estabelecida uma ligação de composição.

Por exemplo, a Figura 6.5 mostra 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 é gerada (Aluno, 2).

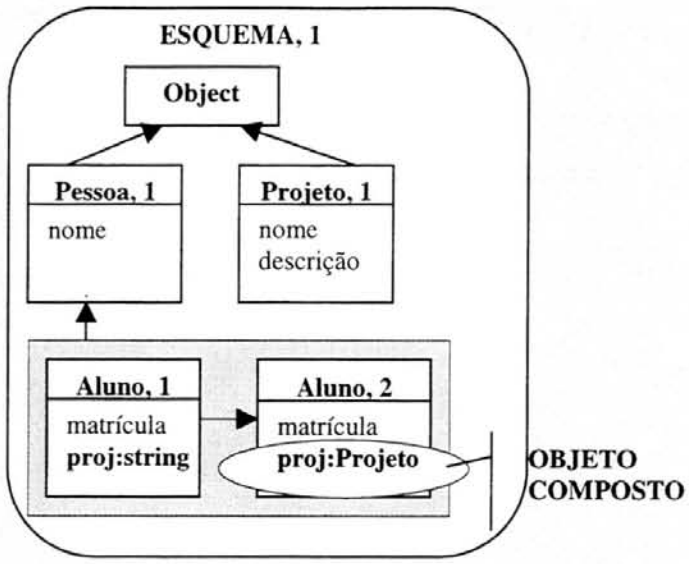


FIGURA 6.5 - Incluir uma ligação de composição

6.3.1.7 Excluir uma Ligação de Composição

Permite excluir uma ligação de composição. Um objeto complexo pode ser modificado para não-complexo, ou seja, uma propriedade que possui como domínio um objeto não-primitivo passa a apresentar um valor atômico, sendo desfeito o relacionamento de agregação.

Por exemplo, a Figura 6.6 ilustra a exclusão de um relacionamento de agregação, presente na primeira versão da classe *Aluno* (Aluno, 1). O atributo *projeto*, cujo domínio é composto por um objeto da classe *Projeto*, passa a apresentar um conteúdo atômico (alfanumérico) como domínio, sendo gerada uma nova versão para a classe (Aluno, 2).

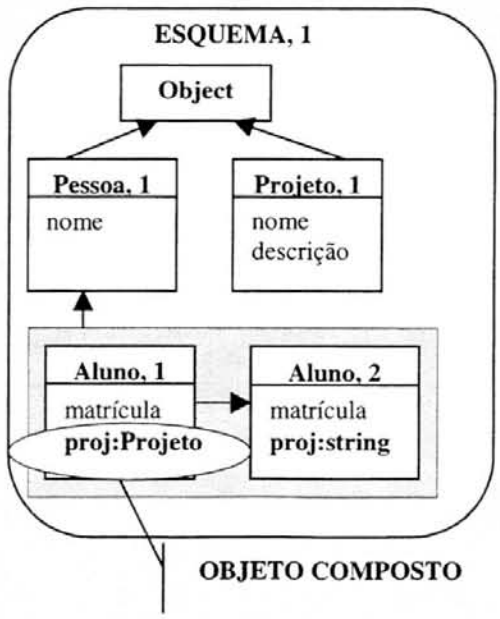


FIGURA 6.6 - Exclusão de ligação de composição

Essa operação elimina um relacionamento de agregação, porém os objetos presentes no banco de dados não são excluídos. Continuam existindo nas classes nas quais foram criados.

6.3.2 Alterações na Estrutura do Esquema: Classes – Versões de Esquema

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

6.3.2.1 Incluir uma Classe

Permite a inclusão de uma classe em uma determinada versão do esquema.

Por exemplo, a Figura 6.7[5] mostra a inclusão da classe *Aluno*, na primeira versão do esquema (Esquema, 1), e, por conseguinte, a derivação de uma nova versão (Esquema, 2).

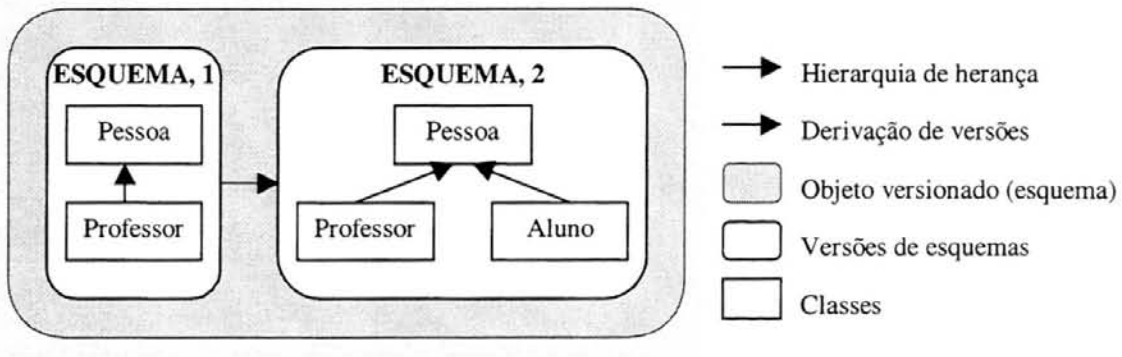


FIGURA 6.7 - Inclusão de uma classe

A inclusão de uma classe implica a seleção de sua posição no grafo de especialização: subclasse/superclasse. Essa escolha deve ser realizada pelo usuário no momento da inclusão da classe. Caso contrário, a nova classe é conectada à classe global, *Object*. A operação de inclusão de classes deve garantir a unicidade de nomes e não incluir um ciclo na hierarquia de herança. Senão a operação deve ser rejeitada e o usuário, notificado.

6.3.2.2 Excluir uma Classe

Permite a eliminação de uma classe presente em uma determinada versão do esquema.

Por exemplo, a Figura 6.8 ilustra a exclusão da classe *Aluno* da primeira versão do esquema (Esquema, 1) e, por conseqüência, a derivação de uma nova versão (Esquema, 2).

---

[5] A legenda definida para a Figura 6.7 é também utilizada para as seguintes figuras: Figura 6.8, Figura 6.10 a Figura 6.16, Figura 6.22 e Figura 6.23.

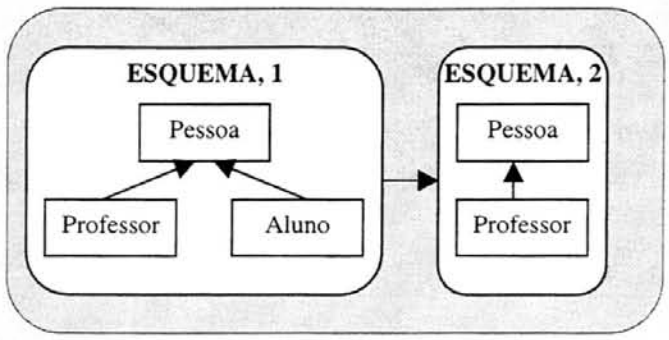


FIGURA 6.8 - Exclusão de uma classe

Existem duas formas para a exclusão de uma classe, considerando sua posição na hierarquia de herança: classe folha ou intermediária no grafo de especialização.

A exclusão de uma classe folha é um processo simples, bastando ao usuário indicar o nome da classe a ser excluída, conforme Figura 6.8. As propriedades definidas para a classe são juntamente eliminadas.

Para uma classe intermediária na hierarquia de herança, três procedimentos podem ser adotados, como a seguir:

(a) a eliminação de uma classe acarreta a exclusão de todas as suas subclasses, ou seja, é realizada uma exclusão em cascata (Figura 6.9(b));

(b) apenas a classe selecionada é excluída, sendo suas subclasses diretas reconectadas na superclasse direta da classe excluída. Na presença de herança múltipla, por *default*, as subclasses são reconectadas na primeira classe definida no esquema como superclasse. Nesse caso, o usuário pode selecionar outra superclasse (Figura 6.9(c));

(c) a subclasse direta da classe excluída é reconectada diretamente na classe raiz da hierarquia de herança, mantendo suas subclasses (Figura 6.9(d)).

Considerando os casos (a) e (b), as propriedades definidas na classe excluída podem ser eliminadas juntamente com a classe ou propagadas para a(s) subclasse(s) direta(s), preservando as informações existentes. Nesse caso, a migração das propriedades deve assegurar a restrição de unicidade de nome. Na presença de especialização, *override*, essa é automaticamente mantida pelo sistema, sendo o processo de migração ignorado e o usuário, notificado.

A Figura 6.9 ilustra a aplicação dos três procedimentos descritos para exclusão de uma classe intermediária. A classe *Empregado* é excluída do esquema. A Figura 6.9(a) apresenta o estado inicial do esquema, sendo indicada a classe excluída (*Empregado*). A Figura 6.9(b) exemplifica uma exclusão em cascata, Figura 6.9(c) reconexão das subclasses com a superclasse e Figura 6.9(d) reconexão com a classe *Object*.

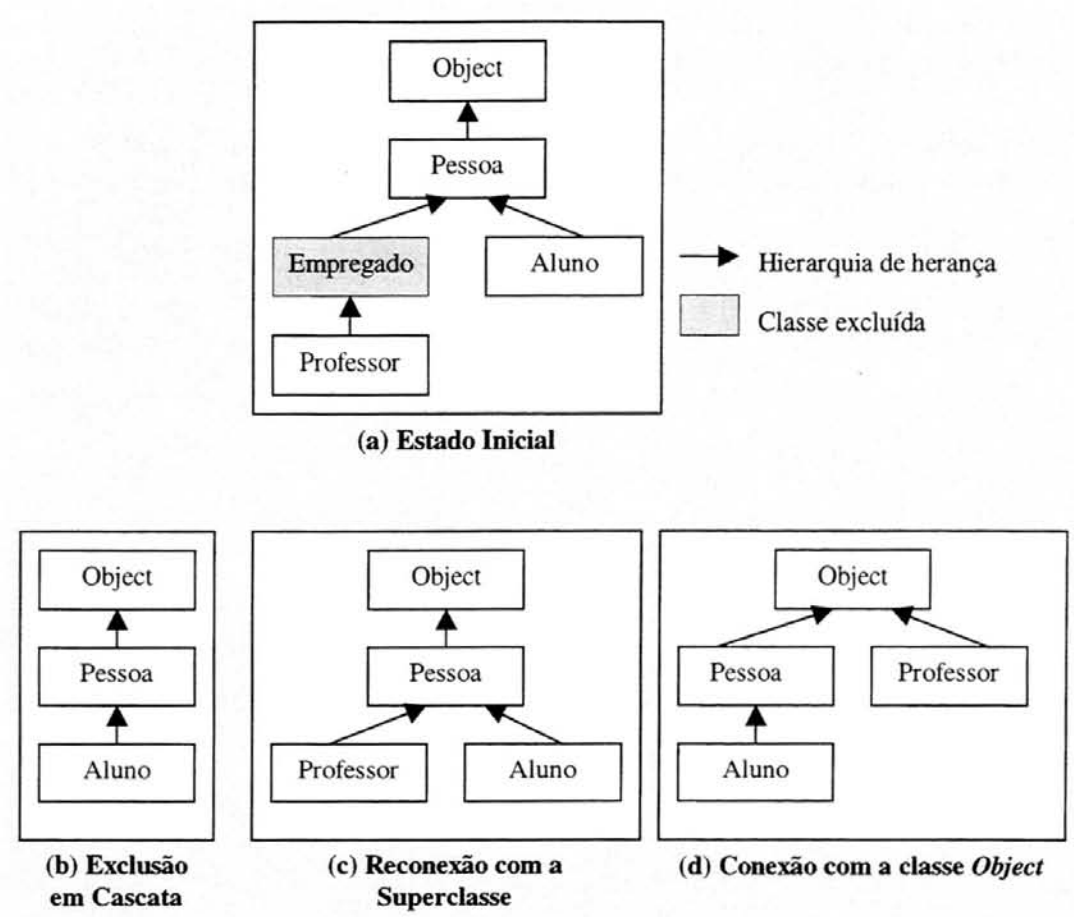


FIGURA 6.9 - Procedimentos para exclusão de classes intermediárias na hierarquia de herança

O processo de reestruturação da hierarquia de especialização deve ser selecionado pelo usuário. Caso contrário, o sistema adota por *default* o procedimento descrito em (a).

A fim de garantir a consistência do esquema, quando a classe excluída é referida por outra como uma agregação, a operação deve ser rejeitada e o usuário notificado.

6.3.2.3 Mudar o Nome de uma Classe

Permite substituir o nome de uma classe definida em uma versão de esquema.

Por exemplo, a Figura 6.10 exibe a classe *Aluno*, na primeira versão do esquema (Esquema, 1), que passa a ser denominada *Estudante* e uma nova versão é derivada (Esquema, 2).

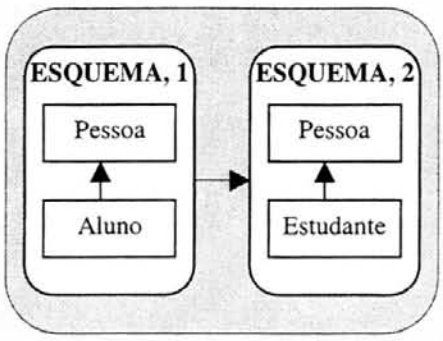


FIGURA 6.10 - Alteração no nome de uma classe

A troca do nome de uma classe deve, obrigatoriamente, manter a unicidade de nomes. Na presença de uma classe com nome igual ao novo nome da classe, a operação de atualização deve ser rejeitada e o usuário notificado.

6.3.2.4 Mudar a Ordem de Superclasses de uma Classe

Permite alterar a ordem de definição das superclasses de uma determinada classe, isto é, a disposição das definições de herança é trocada.

O problema resultante da execução dessa operação refere-se aos conflitos de nomes, na presença de herança múltipla. Como o mecanismo de herança por extensão é adotado, a resolução de conflitos de nome é realizada no instante em que a classe é requerida. Uma vez que a propriedade selecionada é aquela correspondente à primeira superclasse definida no esquema, uma mudança na ordenação das superclasses acarreta também a troca da propriedade quando a referência é redirecionada para os ascendentes, considerando a existência de nomes iguais em classes distintas referidas em ligações de herança múltipla.

Por exemplo, a Figura 6.11 ilustra a classe *Professor* contendo relacionamento de herança múltipla com as classes *Pessoa* e *Docente*. A classe *Pessoa* possui um atributo *nome* com domínio "*a*" e a classe *Docente*, um atributo *nome* com domínio "*b*", caracterizando dois atributos distintos. Duas situações são possíveis quanto à ordem de definição das superclasses da classe *Professor*, considerando o instante em que a referência é redirecionada para a classe ascendente:

- Classe *Pessoa* - definida como primeira superclasse - o atributo *nome* com domínio "*a*" é selecionado;
- Classe *Docente* - definida como primeira superclasse - o atributo *nome* com domínio "*b*" é selecionado.

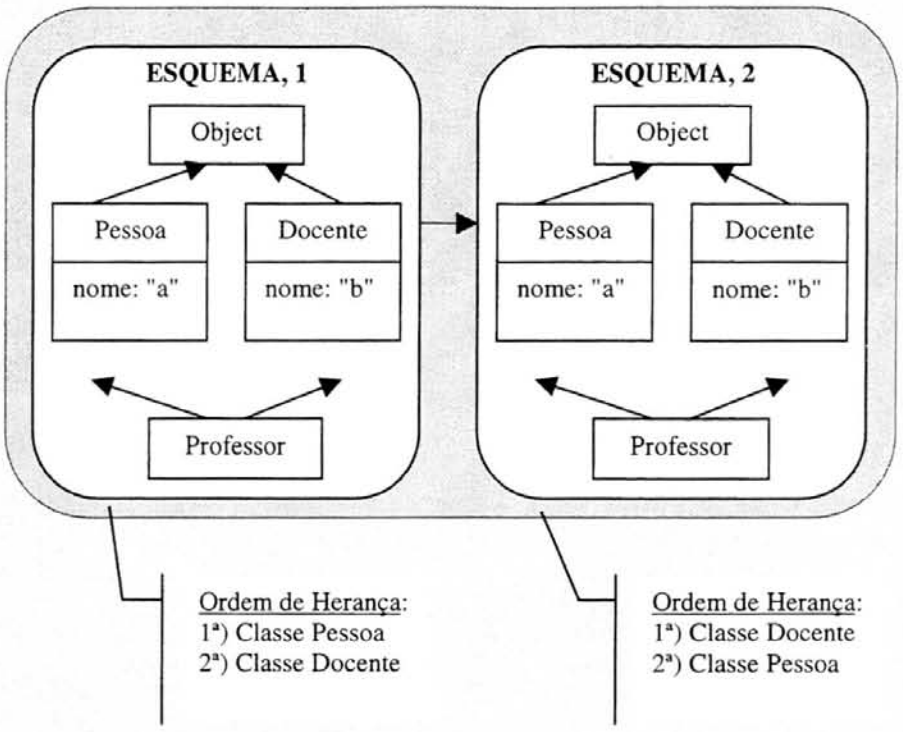


FIGURA 6.11 - Alteração na definição de superclasses - herança múltipla

6.3.2.5 Incluir uma Superclasse

Permite incluir uma classe na lista de superclasse de qualquer outra, porém nenhuma nova classe é criada.

A Figura 6.12 mostra a introdução da classe *Empregado*, definida no esquema como superclasse da classe *Professor*, alterando, assim, a estrutura do esquema (Esquema, 1). Em decorrência dessa modificação, uma nova versão do esquema é derivada (Esquema, 2).

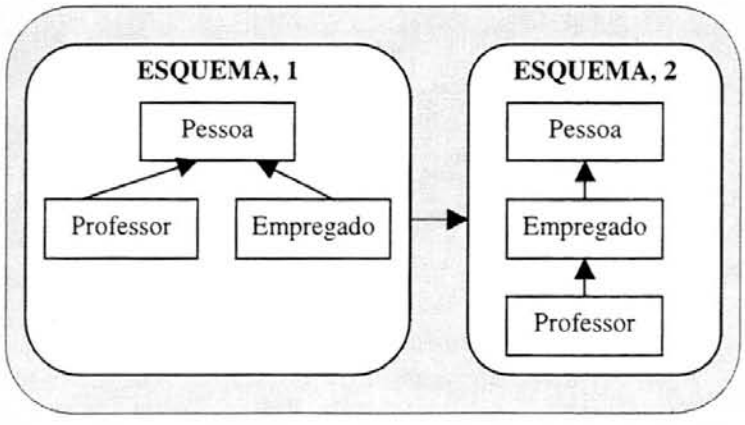


FIGURA 6.12 - Inserir uma superclasse

A restrição considerada é garantir a completa conexão das classes na hierarquia de herança. Na presença de ciclos, a operação é rejeitada e o usuário, notificado.

6.3.2.6 Excluir uma Superclasse

Permite remover uma classe da lista de superclasses de qualquer outra classe. Essa operação desfaz apenas a ligação de herança, sendo que as propriedades não são perdidas e a classe continua definida no esquema.

A Figura 6.13 ilustra a remoção da classe *Pessoa* da lista de superclasses da classe *Professor* na primeira versão do esquema (Esquema, 1) e a geração de uma nova versão (Esquema, 2).

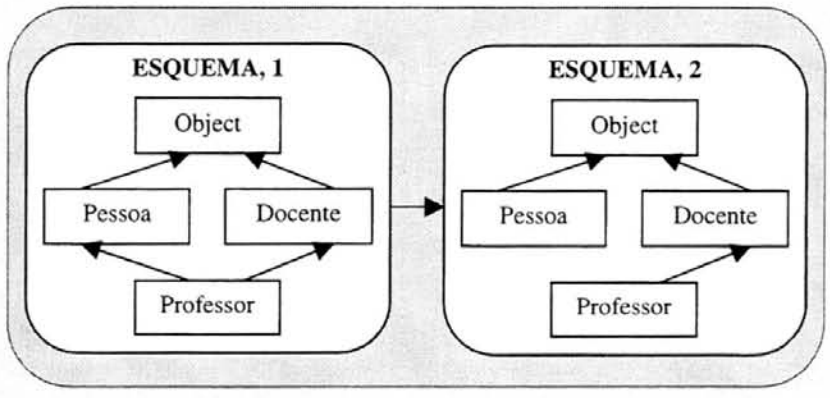


FIGURA 6.13 - Remover uma classe S da lista de superclasses de C

A fim de garantir a conectividade da hierarquia de herança, quando a superclasse é a única para a classe, esta é reconectada à classe raiz: *Object*. Quando a classe, da qual a superclasse está sendo removida, não é folha na hierarquia de especialização, dois procedimentos podem ser adotados. A classe é reconectada à superclasse Object juntamente com suas subclasses ou a subclasse imediata é reconectada à superclasse direta da classe em questão.

6.3.2.7 Criar uma Nova Classe como uma Superclasse Generalizando Outras Existentes

Permite a inclusão de uma classe como uma superclasse, generalizando outras existentes.

Por exemplo, a Figura 6.14 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*.

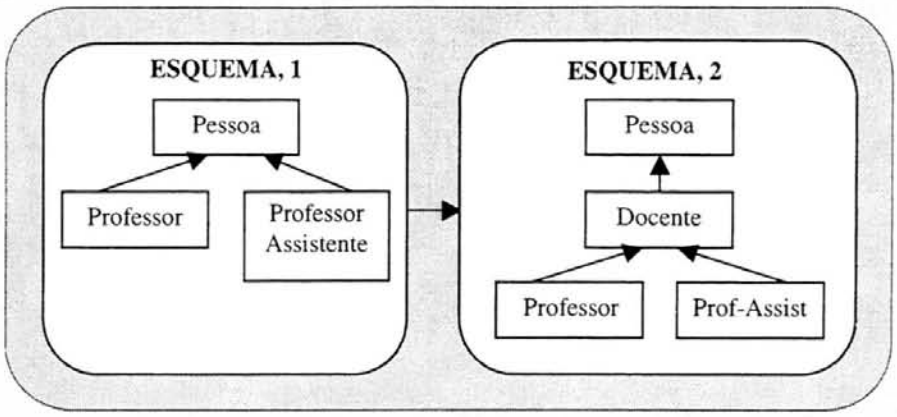


FIGURA 6.14 - Generalização de classes

O procedimento *default* 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 segue as mesmas restrições utilizadas para inclusão de uma classe (vide seção 6.3.2.1) e seu nome é fornecido pelo usuário durante a realização da operação.

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.

6.3.2.8 Decompor uma Classe em Novas Classes

Permite separar as propriedades de uma determinada versão de classe, distribuindo-as para outras classes.

Por exemplo, a Figura 6.15 mostra a classe *Aluno*, presente na primeira versão do esquema (Esquema, 1), sendo decomposta em duas novas classes: *Graduado* e *PósGraduado*, e a derivação de uma nova versão para o esquema (Esquema, 2).

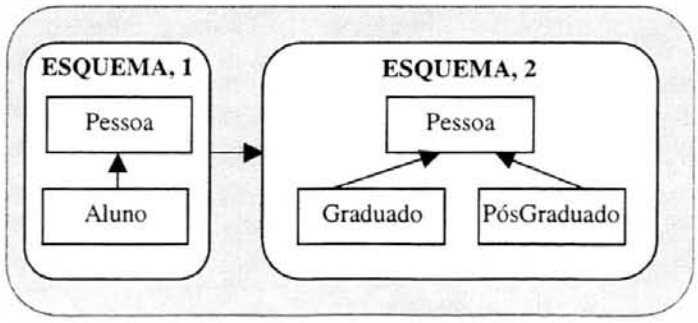


FIGURA 6.15 - Decomposição de classes

Devem ser indicados os atributos que migrarão, bem como suas classes destino. Quatro procedimentos podem ser adotados, conforme a seguir:

- a classe inicial é mantida e novas classes são criadas como especializações da classe fonte a fim de armazenar as propriedades em migração. Entretanto, essas classes podem ser conectadas em outra posição do grafo, bastando o usuário indicar a localização adequada;
- a classe inicial é mantida e as propriedades incorporadas em outras classes existentes na versão do esquema. Tanto as classes destino quanto a fonte são indicadas pelo usuário;
- a classe inicial é excluída e novas classes são geradas para incorporar as propriedades em migração. Por *default*, estas classes são conectadas à primeira superclasse definida para a classe excluída. Porém, outra posição no grafo de especialização pode ser indicada pelo usuário ou ainda as classes conectadas à classe global, *Object*;
- a classe inicial é excluída e as propriedades inseridas em outras classes existentes na versão do esquema. Essas classes destino devem ser indicadas pelo usuário.

Quanto à inclusão e exclusão de classes seguem os procedimentos descritos nas seções 6.3.2.1 e 6.3.2.2, respectivamente.

6.3.2.9 Unir Várias Classes em uma Nova Classe

Permite agrupar propriedades de diversas classes em uma outra classe resultante.

Por exemplo, a Figura 6.16 ilustra a união das classes *Professor* e *ProfessorAssistente* (Esquema, 1), em uma terceira classe, denominada *Docente* e, por conseguinte, a derivação de uma nova versão para o esquema (Esquema, 2).

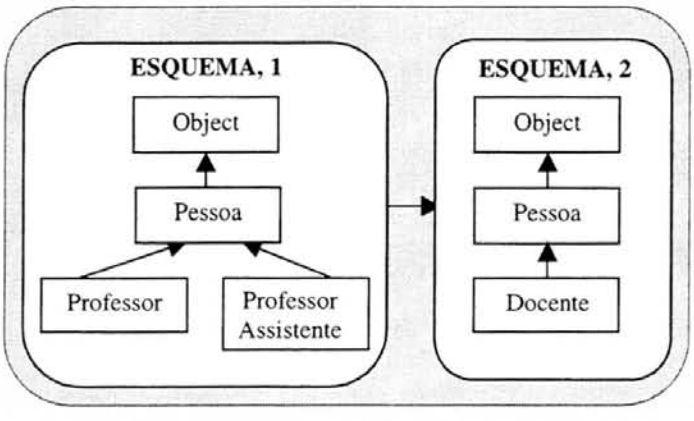


FIGURA 6.16 - União de classes

Inúmeras classes podem ser selecionadas para realização da união. Por *default*, as classes fontes são excluídas, a hierarquia de herança reestruturada e as propriedades incorporadas na nova classe criada. Entretanto, dois procedimentos podem ser realizados:

- quando as classes selecionadas estão posicionadas no mesmo nível na hierarquia de herança, a nova classe é criada como subclasse da superclasse das classes selecionadas;
- quando as classes selecionadas estão em níveis diferentes na hierarquia de herança, a nova classe é criada como subclasse da classe global: Object. Nesse caso, o usuário pode indicar uma outra posição para conexão da classe no grafo de herança.

Todas as propriedades são mantidas na nova classe. Na presença de conflitos de nomes o usuário deve selecionar uma das propriedades ou realizar uma renomeação. Caso contrário, as propriedades são distinguidas pela indicação da classe de origem em cada propriedade duplicada.

Quando as classes selecionadas possuem subclasses, seguem os procedimentos de reconexão de classe definidos na seção 6.3.2.2, referentes à exclusão de classe. E quanto à inclusão e exclusão de classes seguem os procedimentos descritos nas seções 6.3.2.1 e 6.3.2.2, respectivamente.

6.3.3 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, dois procedimentos adicionais são definidos para preservar essa consistência de comportamento.

O primeiro procedimento é manter associada a cada versão de esquema uma lista com informações a respeito dos métodos, sinalizando os atributos por eles utilizados. Por exemplo, na Figura 6.17 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 6.17 - Lista de utilização de métodos

Um método pode fazer referência a outro em sua implementação. Por exemplo, o método *M1* realiza chamadas aos métodos *M2* e *M3*, conforme ilustrado na Figura 6.18(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 a [ZIC91]. Esse grafo mantém informações a respeito de todos os métodos por ele referidos. Os vértices do grafo representam os métodos e as setas indicam seus dependentes. Essas informações são utilizadas pelo sistema quando alterações que prejudicam a execução dos métodos são realizadas. Por exemplo, um dos procedimentos quanto à 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 quando não estiver presente na implementação de outros métodos; caso contrário, a operação deve ser rejeitada e o usuário, notificado. Esse procedimento é realizado pela análise do Grafo de Dependência de Método. A Figura 6.18 ilustra o grafo da implementação do método *M1*.

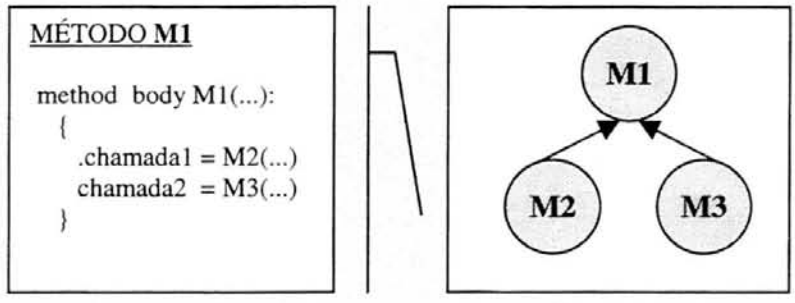


FIGURA 6.18 - Grafo de dependência de métodos

Assim, antes de realizar uma modificação em um método, o modelo impõe a verificação do Grafo de Dependência de Método e a alteração é realizada quando não provoca erros nos métodos existentes. Na presença de erros, a operação deve ser rejeitada e o usuário notificado.

6.3.3.1 Incluir um Método

Permite a inclusão de um método em uma determinada versão de classe. Essa operação implica a especificação dos parâmetros de entrada e retorno para o mesmo.

A inclusão de um método acarreta a derivação de uma nova versão para a classe devido à modificação realizada em sua estrutura. Por exemplo, a Figura 6.19 ilustra a inclusão do método *média* na classe *Aluno* (Aluno, 1) e, por conseguinte, a derivação de uma nova versão para a classe (Aluno, 2).

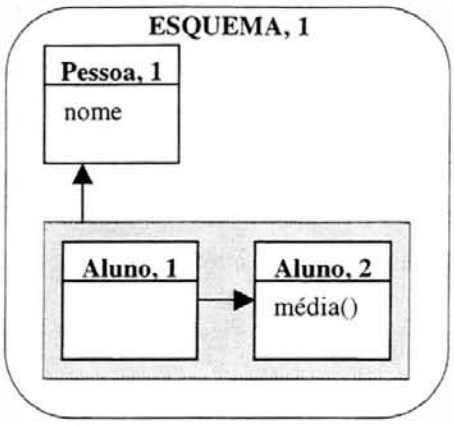


FIGURA 6.19 - Inclusão de métodos

A inclusão de um método deve garantir a unicidade de nomes, ou seja, quando existir na versão da classe um método com o mesmo nome daquele a ser inserido, a operação de atualização deve ser rejeitada e o usuário notificado. Conflitos de nomes podem surgir na presença de herança múltipla. O atributo selecionado deve ser aquele presente na primeira superclasse definida para a versão de classe. Devido à utilização de herança por extensão, a resolução do conflito é realizada no instante em que o objeto é solicitado e não quando criado.

Na inclusão de um método deve ser verificada a compatibilidade de assinatura a fim de evitar erros durante a execução. A classe em que o método está definido e os parâmetro de entrada e retorno devem existir. Os parâmetros precisam ainda apresentar tipos compatíveis. E, a assinatura dos métodos especializados nas subclasses devem ser compatíveis com os da superclasse.

Para garantir a consistência de comportamento, essa operação acarreta a atualização da lista de utilização de métodos e a sinalização dos atributos por ele utilizados, bem como a geração do grafo de dependência de método armazenando os métodos por ele requeridos, conforme descrito na seção 6.3.3.

6.3.3.2 Excluir um Método

Permite a exclusão de um método, definido em uma versão de classe.

A exclusão de um método provoca a derivação de uma nova versão para a classe devido às modificações ocorridas em sua estrutura. Por exemplo, a Figura 6.20 ilustra a exclusão do método *média*, da classe *Aluno* (Aluno, 1) e, por conseguinte, a derivação de uma nova versão para classe (Aluno, 2).

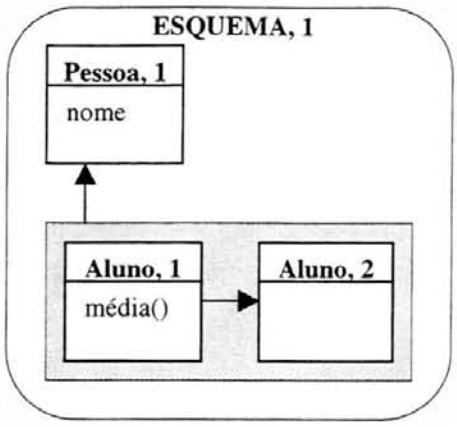


FIGURA 6.20 - Exclusão de métodos

Para garantir a consistência comportamental, a exclusão de um método implica a verificação do seu Grafo de Dependência (vide seção 6.3.3). Quando constatado que o método é requerido por outros métodos, a operação de exclusão deve ser rejeitada e o usuário notificado. Caso contrário, a operação é realizada, a Lista de Utilização de Métodos é atualizada e um novo Grafo de Dependência de Métodos é gerado, ambos na nova versão da classe.

6.3.3.3 Mudar o Nome de um Método

Permite trocar o nome de um método definido em uma versão de classe.

A mudança do nome de um método gera a derivação de uma nova versão para a classe devido à alteração realizada em sua estrutura. Por exemplo, a Figura 6.21 mostra a passagem do nome do método *média* para *cálculo*, na classe *Aluno* (Aluno,1) e a derivação de uma nova versão para a classe (Aluno, 2).

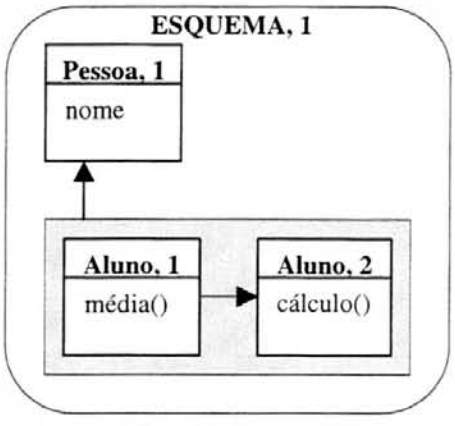


FIGURA 6.21 - Alteração no nome do método

A alteração do nome de um método deve garantir a unicidade de nomes. Caso a classe possua um método com nome igual ao novo nome do método, a operação de atualização deve ser rejeitada e o usuário, notificado.

Para garantir a consistência comportamental, essa operação implica a atualização do nome do método na Lista de Utilização de Métodos e no Grafo de Dependência na nova versão gerada para a classe.

6.3.3.4 Mudar o Código de um Método

Permite modificar a implementação de um método definido em uma versão de classe.

A mudança do código de um método provoca a derivação de uma nova versão para o método a fim de armazenar as modificações nele realizadas.

Essa modificação implica a criação de uma nova Lista de Utilização de Métodos e um novo Grafo de Dependência para a nova versão do método. Toda modificação no código dos métodos deve ser realizada pelo usuário.

6.3.3.5 Mudar o Domínio e/ou Contradomínio de um Método

Permite modificar o domínio e/ou contradomínio de um método, definido em uma versão de classe, resultando na derivação de uma nova versão para o método. São distinguidos três tipos de modificações: alteração no domínio do argumento de retorno, alteração no domínio dos argumentos de entrada e alteração na quantidade de argumentos.

A alteração do tipo de retorno é caracterizada pela troca do domínio definido para o parâmetro de retorno. Quando o argumento de retorno tem o domínio alterado para um tipo menos especializado, a execução do método torna-se inválida, pois o conteúdo do argumento resultante não está contido no escopo para ele definido. Nesse caso, o usuário deve ser notificado a fim de realizar as alterações necessárias, evitando, assim, erros de execução. Quando o domínio é modificado para um tipo compatível ou mais especializado, a execução do método prossegue normalmente.

A alteração de parâmetros é caracterizada pela troca do domínio definido para os argumentos de entrada. Quando o domínio dos parâmetros é alterado para um tipo mais especializado, a execução do método resulta em erros, pois em seu código o argumento não está contido no escopo para ele definido. Nesse caso, o usuário deve ser notificado para realizar as alterações necessárias no código do método, evitando, assim, erros de execução. Quando o domínio é alterado para um tipo compatível ou menos especializado, a execução do método prossegue normalmente.

A alteração do número de parâmetros permite eliminar ou inserir um argumento na assinatura do método. O usuário deve ser notificado para alterar as chamadas ao método a fim de evitar erro de execução.

6.3.3.6 Considerações: Alterações em Atributos e Classes

Como alterações em atributos e classes provocam a derivação de versões de classes e esquemas, uma nova versão pode afetar os métodos definidos na versão anterior. Por exemplo, a exclusão de um atributo, quando presente na implementação de um método, ocasiona erros de execução do método na nova versão da classe. Assim, alterações em atributos e classes devem também garantir a consistência comportamental.

As tabelas 6.1 e 6.2 apresentam as operações de modificação e suas implicações na atualização dos procedimentos que visam a preservar a consistência comportamental. A Tabela 6.1 apresenta as operações que alteram a estrutura das classes (seção 6.3.1) e a Tabela 6.2 as operações que alteram a estrutura do esquema (seção 6.3.2).

TABELA 6.1 - Procedimentos de atualização nas modificações na estrutura das classes

Operações de modificação	**Procedimentos de Atualização**	
	Lista de utilização de métodos (LUM)	**Implementação do código dos métodos**
Alterações na estrutura das classes		
Incluir atributo	- o atributo é incluído na LUM para a nova versão da classe	**
Excluir atributo	- o atributo é excluído somente quando não estiver presente na LUM para a versão inicial da classe mas não para nova versão gerada	**
Mudar nome atributo	- o nome do atributo é atualizado na LUM na nova versão gerada para classe	- o usuário deve alterar o nome dos atributos nos métodos que os utilizam
Mudar domínio atributo	**	- alteração na implementação dos métodos deve ser realizada pelo usuário
Mudar valor atributo	**	**
Incluir ligação composição	**	- alteração na implementação dos métodos deve ser realizada pelo usuário
Excluir ligação composição	**	- alteração na implementação dos métodos deve ser realizada pelo usuário

** nenhuma operação é realizada

TABELA 6.2 - Procedimentos de atualização nas modificações na estrutura do esquema

Operações de modificação	**Procedimentos de Atualização**
	Lista de utilização de métodos (LUM)
Alterações na estrutura das classes	
Incluir classe	- a classe é incluída na LUM na versão gerada para o esquema
Excluir classe	- a classe permanece na LUM da versão inicial do esquema mas não na nova versão derivada
Mudar nome classe	- o nome da classe é atualizado na LUM da nova versão gerada para o esquema
Mudar ordem superclasse	**
Incluir superclasse	**
Excluir superclasse	**
Generalizar classes	- as classes geradas são incluídas na LUM para nova versão do esquema - as propriedades que trocam de classe são também atualizadas na LUM para a nova versão do esquema
Decompor classe	- as classes geradas são incluídas na LUM para a nova versão do esquema - as classes excluídas permanecem na LUM da versão inicial do esquema mas não na nova versão gerada para o esquema - as propriedades que trocam de classe são também atualizadas na LUM para a nova versão do esquema
Unir classes	- as classes geradas são incluídas na LUM para a nova versão do esquema - as classes excluídas permanecem na LUM da versão inicial do esquema mas não na nova versão gerada para o esquema - as propriedades que trocam de classe são também atualizadas na LUM para a nova versão do esquema

** Nenhuma operação é realizada

6.4 Funções de Adaptação

Como o banco de dados é modificado e versões de esquemas e/ou classes são derivadas, é preciso garantir que as informações requisitadas sejam recuperadas de acordo com a versão sob a qual estão sendo solicitadas.

Para tanto, funções de adaptação são definidas a fim de adequar as informações presentes na base de dados com a requerida versão de esquema e/ou classe, acionadas no instante em que o objeto é recuperado.

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 e/ou classes 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.

6.4.1 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 comportamento comuns. Assim, surge a necessidade de estabelecer relacionamentos de equivalência entre as classes de diversas versões de esquema.

Funções de propagação são definidas entre duas versões de esquema estabelecendo relacionamentos de equivalência entre suas classes. Por exemplo, a Figura 6.22 ilustra uma alteração no nome de uma classe, classe *Aluno* (Esquema, 1) para *Estudante* (Esquema, 2), e, consequentemente, a derivação de uma nova versão para o esquema. Nesse caso, qualquer instância, criada para classe Aluno ou Estudante, pode ser recuperada nas duas versões de esquema.

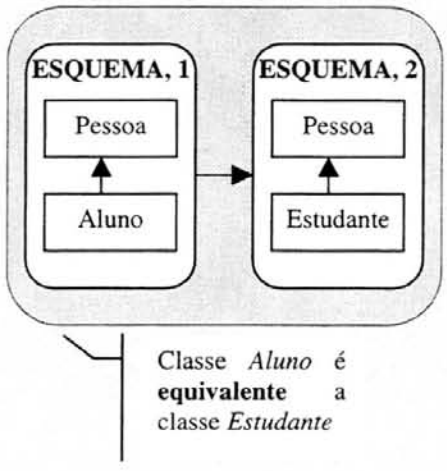


FIGURA 6.22 - Definição de Funções de Propagação

As Funções de Propagação apresentam duas finalidades quanto aos relacionamentos entre as classes em versões distintas de esquema: Igualdade – define igualdade de classes que possuem nomes diferentes mas propriedades e comportamento iguais; e Diferença – define diferença de classes que possuem nomes iguais mas propriedades e comportamento distinto.

Uma função de propagação é definida entre duas classes em versões distintas de esquema e é acionada no instante em que a versão 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. 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.

Como mostrado na Figura 6.23 entre a primeira (Esquema, 1) e segunda versão de esquema (Esquema, 2), duas funções são definidas: *update* e *backdate*. Um objeto gerado na primeira versão do esquema é adaptado para a segunda pela execução da função update e o processo inverso é realizado pela execução da função *backdate*.

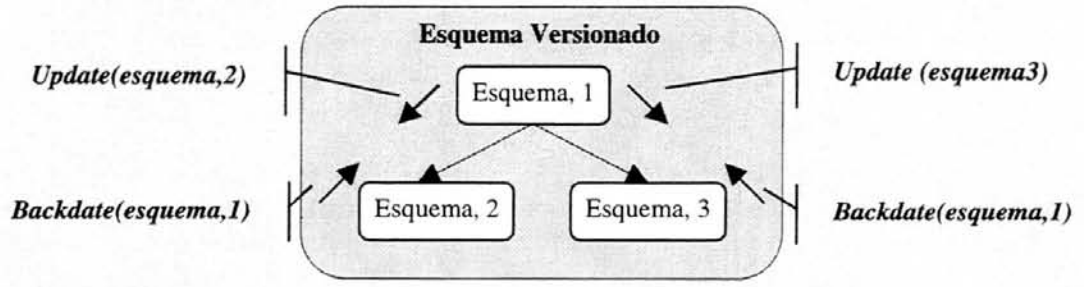


FIGURA 6.23 - Exemplo de funções de propagação

As funções de propagação são implementadas pelo usuário quando este julgar necessário, pois nem sempre alterações na estrutura das classes impõem necessidade quanto a definição de equivalências entre as classes. Por exemplo, a simples inclusão de uma classe não requer a definição de nenhuma equivalência com o esquema antecedente ou descendente.

Assim, as funções de propagação estabelecem relacionamentos semânticos entre classes em diversas versões de esquema, garantindo o comportamento consistente dos objetos em qualquer contexto de versão sob o qual são requeridos.

6.4.2 Funções de Conversão

Como a estrutura de uma classe pode ser modificada e, por conseguinte, novas versões são derivadas, 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 classe para outra, dentro de um mesmo esquema.

Uma função de conversão implica na 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 da classe, isto é, implementam operações que adaptam os objetos segundo a evolução de esquema. As funções *backdate* são associadas à nova versão de classe e descrevem o comportamento dos objetos na versão anterior, ou seja, antes de realizadas as modificações.

Por exemplo, a Figura 6.24 mostra a inclusão do atributo *área* na 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.

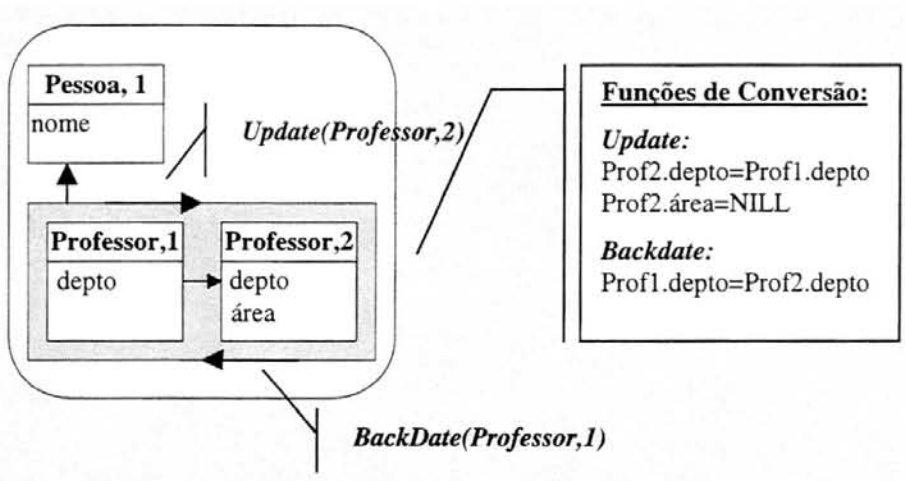


FIGURA 6.24 - Exemplo de uma função de conversão

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 versão de classe é derivada, caso o usuário não implemente a função de conversão, esta é definida pelo sistema, adotando o procedimento *default* estabelecido pelo modelo. A tabela 6.3 apresenta o procedimento padrão adotado para cada operação de atualização, quando a função não é especificada pelo usuário.

TABELA 6.3 - Procedimentos das funções de conversão

Primitivas de atualização	Funções *Update*	Funções *Backdate*
Inclusão de atributo	Inicia o atributo na nova versão da classe com valor nulo	Não propaga o conteúdo do atributo para a antiga versão da classe
Exclusão de atributo	Não propaga o conteúdo do atributo para nova versão da classe	Inicia o atributo na antiga versão da classe com valor nulo
Troca do nome do atributo	Ao atributo da nova versão da classe é atribuído o conteúdo do atributo da antiga versão	Ao atributo da antiga versão da classe é atribuído o conteúdo do atributo da nova versão
Alteração no domínio de atributo	Implementada pelo projetista para adequá-la conforme a modificação realizada	Implementada pelo projetista para adequá-la conforme a modificação realizada
Alteração no valor inicial de um atributo	Nenhuma operação é realizada	Nenhuma operação é realizada
Exclusão de uma ligação de composição	A ligação de composição é desfeita para a nova versão da classe	Ao objeto composto da antiga versão da classe é atribuído valor nulo
Inclusão de uma ligação de composição	Ao objeto composto da nova versão da classe é atribuído valor nulo	A ligação de composição é desfeita para a antiga versão da classe

6.5 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. O mecanismo de versões é a técnica adotada para adaptação das instâncias vigentes no banco de dados.

A utilização da técnica de versionamento de objetos na adaptação das instâncias 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 uma mesma classe e/ou esquema, permitindo a navegação retroativa e proativa entre as versões para realização de operações de alterações e consultas.

Quando um novo esquema é instanciado, versões de objetos são derivadas a fim de adequarem as instâncias atingidas pelas modificações à nova estrutura definida para a versão da classe e/ou esquema.

As versões dos objetos podem ser criadas gradativamente pelo usuário, seguindo os requisitos definidos pelo modelo de versões, ou geradas automaticamente pelo sistema como forma de adaptar as instâncias a cada versão esquema/classe correspondente.

Como projetos de bancos de dados são, em geral, extensos e complexos, a realização imediata da adaptação das instâncias pode provocar bloqueios nas aplicações em execução, restringindo as funcionalidades do modelo. A técnica de conversão adiada[6] é adotada, realizando a adaptação dos objetos somente quando o acesso a eles é requerido.

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

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

O processo de versionamento de objetos pode gerar uma grande quantidade de versões aumentando a complexidade de manipulação e armazenamento. As versões são armazenadas fisicamente (versões físicas) quando sua utilização é considerada freqüente, isto é, quando constatada a importância da disponibilidade para o sistema de banco de dados. E, em situação oposta, derivadas logicamente (versões lógicas).

A importância de uma versão é averiguada por mecanismos de métrica que identificam a necessidade da versão do objeto para os programas que os utilizam. As regras são apresentadas a seguir e estão definidas em [BEN95a, BEN95b]:

- DEFINIÇÃO 1 – Peso de uma Versão de Classe – identificado pela razão entre o número de programas associados à versão da classe e o total de programas definidos para o esquema. Tratando-se do esquema 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 importante, "pertinente", para os programas aplicativos quando o peso, calculado na definição 1, é maior ou igual ao valor de PT e obsoleta, caso contrário.

Quando um objeto é solicitado, por um programa ou consulta, sua estrutura é comparada à definição corrente do esquema e classe a que pertence, verificando, assim, se existe uma versão para o objeto. Caso não haja, um novo objeto é gerado. Na presença do objeto no banco de dados, a última versão é localizada e as funções de propagação acionadas para deixar o objeto em conformidade com sua definição. Em ambos os casos, antes da derivação da versão para o objeto o nível de pertinência é definido, averiguando se a instância a ser criada deve permanecer armazenada ou somente calculada.

---

[6] Na literatura, a técnica de conversão adiada está definida em [ZIC91, FER93, FER94a, FER94b] e também é adotada por [LAU97a, LAU97b], dentre outros.

Cada versão de objeto pertence a, pelo menos, um esquema real, físico, podendo adaptar-se a outras perspectivas de esquemas/classes. Assim, consultas podem ser realizadas utilizando mais de uma representação conceitual da base de dados.

6.6 Considerações

Nesse capítulo foi especificado o modelo de evolução de esquemas conceituais para banco de dados orientados a objetos com a utilização de versões. A tabela 6.4, colocada no final dessa seção, apresenta um resumo comparativo abordando as principais características envolvidas no processo. São consideradas as propostas apresentadas no capítulo 3 como também o modelo descrito nesse capítulo.

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. A utilização de versões para esquemas, classes 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.

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.

O cuidado com a excessiva proliferação de versões de objetos é tratado nas estratégias de propagação das instâncias através de um processo dinâmico em que o número de versões é limitado àquelas que são realmente necessárias à aplicação considerada. Desse modo, a área de armazenamento é reduzida e a manutenção dos relacionamentos entre as versões, facilitado.

Enfim, a utilização de versões no processo de evolução de esquemas apresenta uma vantagem significativa por permitir o armazenamento do histórico das modificações, apenas requerendo um nível maior de controle nas atividades de criação, manipulação e consulta às versões que habitam simultaneamente o esquema conceitual e a base de dados.

TABELA 6.2(a) Comparação do modelo proposto com os demais analisados

PROPOSTA DE EVOLUÇÃO DE ESQUEMAS COM VERSÕES							
Itens para comparação	***Won Kim***	***Farandole 2***	***M. R. Fornari***	***Erik Odberg***	***Boulean Benatallah***	***Stven-Eric Lautemann***	***Modelo Proposto***
Tipos de herança	Simples e múltipla	simples	simples e múltipla	simples e múltipla	simples e múltipla	simples e múltipla	simples e múltipla
Mecanismos de herança	por refinamento	por refinamento	por refinamento	por refinamento	por refinamento	por refinamento	por extensão
Objeto Versionado (*)	sem atributos próprios	sem atributos próprios	nada consta	nada consta	sem atributos próprios	nada consta	com atributos próprios
Estrutura dos relacionamentos (*)	Árvore	grafo acíclico	grafo acíclico	árvore	lista	grafo acíclico	grafo acíclico
Tipos e estados das versões (*)	Transitória, em trabalho	em trabalho, estáveis	em trabalho, estáveis	nada consta	nada consta	nada consta	em trabalho, estável, consolidada
Definição de versão corrente	Versão corrente é aquela em uso pela aplicação	versão corrente selecionada pelo usuário	última criada, podendo ser alterada pelo usuário	nada consta	última versão criada	nada consta	última criada, podendo ser alterada pelo usuário
Ligação de versões a um objeto composto (*)	Estática e dinâmica	estática e dinâmica	estática	estática	dinânica	nada consta	estática e dinâmica

(*) Extraído de [GOL95a]

TABELA 6.2(b) Comparação do modelo proposto com os demais analisados (continuação)

PROPOSTA DE EVOLUÇÃO DE ESQUEMAS COM VERSÕES							
Itens para comparação	***Won Kim***	***Farandole 2***	***M. R. Fornari***	***Erik Odberg***	***Boulean Benatallah***	***Stven-Eric Lautemann***	***Modelo Proposto***
Criação de versões (*)	Implícita	explícita	implícita	implícita	implícita e explícita	explícita	implícita e explícita
Exclusão de versões (*)	Apenas versões transitórias e/ou folhas na hierarquia de derivação	apenas para versões em trabalho	não permite	não permite	não permite	nada consta	versões folhas que não sejam versões consolidadas
Granularidade do versionamento	Esquemas e objetos	esquemas	classes, métodos e instâncias	esquemas e classes	esquemas, classes e objetos	esquemas e objetos	esquemas, classes, métodos e objetos
Alterações na definição das classes	- inclusão e exclusão de propriedades	inclusão e exclusão de relacionamentos de composição	- inclusão e exclusão de propriedades - modificação na definição das propriedades	- inclusão e exclusão de atributos	- inclusão e exclusão de propriedades - modificação na definição das propriedades - modificação no código dos métodos		- inclusão e exclusão de propriedades - modificação na definição das propriedades - inclusão e exclusão de ligação de composição - modificação no código dos métodos

(*) Extraído de [GOL95a]

TABELA 6.2(c) Comparação do modelo proposto com os demais analisados (continuação)

PROPOSTA DE EVOLUÇÃO DE ESQUEMAS COM VERSÕES							
Itens para comparação	***Won Kim***	***Farandole 2***	***M. R. Fornari***	***Erik Odberg***	***Boulean Benatallah***	***Stven-Eric Lautemann***	
Alteração na estrutura das classes	- inclusão e exclusão de classes - modificação no relacionamento de herança entre as classes	- inclusão e exclusão de classes - redefinição de superclasse e subclasse	- inclusão e exclusão de classes - inclusão e exclusão de superclasse e subclasse -mover propriedades da superclasse para subclasse e vice-versa	- inclusão e exclusão de classes - alterar superclasse das classes existentes	- inclusão e exclusão de classes - inclusão e exclusão de relacionamentos de herança	- inclusão, exclusão e renomeação de classes - inclusão e exclusão de superclasses	- inclusão, exclusão e renomeação de classes - inclusão e exclusão de superclasses - modificação no relacionamento de herança entre as classes -mover propriedades da superclasse para subclasse e vice-versa
Restrições quanto a alterações no esquema	não são verificadas	define regras que impõem restrições as alterações a fim de garantir a consistência da base de dados	define invariantes de esquema que asseguram a consistência da base de dados	não são verificadas	não são verificadas	não são verificadas	define invariantes de esquema que asseguram a consistência da base de dados

(*) Extraído de [GOL95a]

TABELA 6.2(d) Comparação do modelo proposto com os demais analisados (continuação)

PROPOSTA DE EVOLUÇÃO DE ESQUEMAS COM VERSÕES							
Itens para comparação	***Won Kim***	***Farandole 2***	***M. R. Fornari***	***Erik Odberg***	***Boulean Benatallah***	***Stven-Eric Lautemann***	***Modelo Proposto***
Métodos utilizados para adaptação das instâncias	Versionamento	compartilhamento das instâncias com todas as versões	versionamento	compartilhamento das instâncias com todas as versões	versionamento e emulação	versionamento	versionamento e emulação

(*) Extraído de [GOL95a]

7 Conclusões e Extensões Futuras

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 [AND91, BEN94, BEN95a, BEN95b, FOR93a, KIM88, LAU96a, LAU96b, LAU97a, LAU97b, ODB93, ODB94a, ODB94b, ODB94c, ODB94d], mas também através de técnicas de conversão [ZIC91], visões [MOT96], dentre outras.

Um modelo de evolução de esquemas foi proposto nesse trabalho como forma de guiar modificações em esquemas, classes, objetos e métodos com o objetivo de manter o histórico das alterações realizadas, tendo como base o modelo de versões definido em [GOL95a].

O modelo de versões de Golendziner [GOL95a] 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.

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 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 à transição.

Dentro desse contexto, foi definido um modelo de evolução de esquema que busca identificar e tratar os diversos aspectos relacionados à evolução de esquemas com versões, destacando as seguintes características:

- 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 quanto à 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.

Assim, o diferencial desse trabalho consiste em prover uma generalidade frente ao processo de evolução de esquemas conceituais, ao invés de dar ênfase a conceitos esparsos, investigando, explorando e identificando os vários aspectos envolvidos no processo de evolução de esquemas em bancos de dados orientados a objetos. [7]

Durante o desenvolvimento desse trabalho, foram identificados vários aspectos que podem ser melhor explorados ou estendidos, fora do âmbito dessa dissertação:

- foram propostos mecanismos para derivação de versões sempre que ocorrerem modificações no esquema do banco de dados. Essa forma de derivação pode ser estendida, permitindo ao usuário definir um conjunto de operações de modificação, sem que o sistema, automaticamente, gere novas versões para cada operação individualmente. Para tanto, mecanismos de controle de transações de alteração de esquema precisam ser estabelecidos, a fim de validar a consistência do esquema frente ao conjunto de modificações bem como garantir que todo histórico das alterações seja mantido;
- versões de objetos são derivadas somente para representar e acompanhar a mudança estrutural de esquemas e classes. Como extensão desse trabalho, poderia ser desenvolvido um mecanismo para controle da evolução dos objetos vigentes na base de dados e sua integração com o esquema definido;
- apesar de reconhecida a importância dos mecanismos de evolução de esquemas em sistemas de bancos de dados orientados a objetos, e o fato de que várias linhas de pesquisa têm expandido e explorado tais funcionalidades, percebe-se a ausência de um consenso, não havendo modelos completamente definidos ou totalmente implementados. Nesse trabalho foram propostos conceitos e mecanismos em um nível alto de abstração, podendo ser utilizados como base para futuras implementações.

Enfim, foi mostrado que o uso de versões 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 consulta às versões que habitam simultaneamente o esquema conceitual e a base de dados.

---

[7] [GAL98] artigo submetido e apresentado no Simpósio Brasileiro de Banco de Dados, consistindo um resultado positivo desse trabalho.

Bibliografia

[ALJ 95] AL-JADIR, L. et al. Evolution Features of the F2 OODBMS. In: INTERNATIONAL CONFERENCE ON DATABASE SYSTEMS FOR ADVANCED APPLICATIONS, 4., 1995, Singapore. **Proceedings...** [S.l.: s.n.], 1995.

[AND 91] ANDANY, J.; LÉONARD M.; PALISSER C. Management of schema evolution in databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES CONFERENCE, 17., 1991, Barcelona, Catalonia, Spain. **Proceedings...** [S.l.: s.n.], 1991.p.161-170.

[ATK 89] ATKINSON, M. et al. The Object-Oriented Database System Manifesto. In: INTERNATIONAL CONFERENCE ON DEDUCTIVE AND OBJECT-ORIENTED DATABASES, 1., 1989, Kyoto, Japan. **Proceedings...** [S.l.:s.n.], 1989. p.40-57.

[BAN 87] BANERJEE, Jay; KIM, Won. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1987, San Francisco, CA. **Proceedings...** New York: ACM Press, 1987. p.311-322.

[BEN 94] BENATALLAH, Boualem. Evolution de schéma et systèmes à objects: une synthèse. In: INFORSID, 1994, Aix-en-Provence, France. **Proceedings...** [S.l.:s.n.], 1994.

[BEN 95a] BENATALLAH, Boualem. **Sur le contrôle de versions pour l'évolution de schéma.** [S.l.:s.n.], 1995. Rapport ete Recherche.

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

[BRÈ 96] BRÈCHE, P. Advanced Primitives for Changing Schemas of Objects Databases. In: CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, 7., 1996, Heraklion, Crete. **Proceedings...** [S.l.]: Springer Verlag, 1996.

[CAM 96] CAMOLESI JÚNIOR, Luiz; TRAINA JÚNIOR, Caetano. Evolução de Esquemas de Dados - Um panorama Amplo de Aspectos Técnicos e Gerenciais. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 11., 1996, São Carlos, SP. **Anais...** São Carlos: ICMS/USP, 1996. p.1-19.

[CHE 94] CHEN, Jia-Lin. Schema evolution for object-based accounting database systems. In: INTERNATIONAL SYMPOSIUM ON OBJECT-ORIENTED METHODOLOGIES AND SYSTEMS, 1994. **Proceedings...** Palermo: Springer-Verlag, 1994. p.40-52.

[DEU 91] DEUX, O. et al. The O_2 System. **Communications of ACM**, New York, v.34, n.10, p. 35-48, Oct. 1991.

[DIT 88] DITTRICH, K. R.; LORIE, R. A. Version support for engeneering database systems. **IEEE Transactions on Software Engeneering**, New York, v.14, n.4, p. 429-437, Apr. 1988.

[FER 93] FERRANDINA, F.; ZICARI, R. Object Database Schema Evolution: are Lazy Updates always Equivalent to Immediate Updates? In: WORKSHOP ON SUPPORTING THE EVOLUTION OF CLASS DEFINITIONS, 1993, Washington, DC. **Proceedings...** [S.1.: s.n.], 1993.

[FER 94a] FERRANDINA, F.; MEYER, T.; ZICARI, R. Correctness of Lazy Database Updates for Object Database Systems. In: INTERNATIONAL WORKSHOP ON PERSISTENT OBJECT SYSTEMS, 6., 1994, Tarascon, France. **Proceedings...** Berlin: Springer-Verlag, 1994.

[FER 94b] FERRANDINA, F.; MEYER, T.; ZICARI, R. Implementing Lazy Database Updates for an Object Database System. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, 20., 1994, Santiago, Chile. **Proceedings...** [S.1.]: Morgan Kaufmann, 1994. p.261-272.

[FER 95] FERRANDINA, F.; FERRAN, G. Schema and Database Evolution in the O2 Object Database Systems. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, 21., 1995, Zürich, Switzerland. **Proceedings...** [S.1.]: Morgan Kaufmann, 1995. p.170-181.

[FER 96] LAUTEMANN, S.; FERRANDINA, Fabrizio. An Integrated Approach to Schema Evolution for Object Database. In: INTERNATIONAL CONFERENCE ON OBJECT-ORIENTED INFORMATION SYSTEMS, 3., December 1996, London, UK. **Proceedings...** [S.1.:s.n.,1996]. p.280-294.

[FOR 92] FORNARI, Miguel Rodrigues. **Um estudo sobre evolução de esquemas em banco de dados**: trabalho individual. Porto Alegre: CPGCC da UFRGS, 1992. 78p.

[FOR 93a] FORNARI, Miguel Rodrigues. **Evolução de esquemas em banco de dados orientados a objetos utilizando versões.** Porto Alegre: CPGCC da UFRGS, 1993. 131p. Dissertação de Mestrado.

[FOR 93b] 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., 1993, Campina Grande. **Anais...** Campina Grande: SBC/UFPb, 1993. p.113-127.

[FOR 94a] FORNARI, Miguel Rodrigues; GOLENDZINER, Lia Goldstein; WAGNER, Flávio Rech. Schema evolution in STAR framework. In: INTERNATIONAL IFIP WORKING CONFERENCE ON ELETRONIC DESIGN AUTOMATION FRAMEWORKS, 4., 1994, Gramado, Brasil. **Proceedings...** London: Chapman & Hall, 1994. p.45-54.

[FOR 94b] FORNARI, Miguel Rodrigues. **Gerente de Evolução de Esquemas para o ambiente STAR.** Porto Alegre: CPGCC da UFRGS, 1994. 73p. (RP-224).

[FOR 94c] FORNARI, Miguel Rodrigues; GOLENDZINER, Lia Goldstein; WAGNER, Flávio Rech. Evolução de esquemas para o ambiente STAR. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 14., SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, 21.,1994, Caxambu, MG. **Anais...** Belo Horizonte: UFMG, 1994. p.113-126.

[GAL 96] GALANTE, Renata de Matos. **Suporte à evolução de esquemas com o uso de versões**: trabalho individual. Porto Alegre: CPGCC da UFRGS, 1996. 60p. (TI-575).

[GAL 98] 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., Maringá-PR. **Anais...** Maringá: UEM-DIN, 1998. p.303-318.

[GOL 93a] GOLEDZINER, Lia Goldstein; SANTOS, Clesio Saraiva.Versões em Banco de Dados Orientados a Objetos. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 8., 1993, Campina Grande. **Anais...** Campina Grande: SBC/UFPb, 1993. p.7-20.

[GOL 93b] GOLEDZINER, Lia Goldstein.**Um estudo sobre Versões em Banco de Dados Orientados a Objetos.** Porto Alegre: CPGCC da UFRGS, 1993. 75p. (RP-213).

[GOL 95a] GOLEDZINER, Lia Goldstein. **Um modelo de versões para banco de dados orientados objetos.** Porto Alegre: CPGCC da UFRGS, 1995. 147p. Tese de doutorado.

[GOL 95b] GOLEDZINER, Lia Goldstein; SANTOS, Clesio Saraiva dos. Definição e manipulação de versões em banco de dados orientados a objetos. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 1995, Recife-PE, **Anais...** Recife: UFPE/DI, 1995. p.335-349.

[GOL 95c] GOLEDZINER, Lia Goldstein; SANTOS, Clesio Saraiva dos. Uma abordagem multi-nível para suporte de versões em banco de dados orientados a objetos. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 15., 1995, Canela, RS. **Anais...** Porto Alegre: SBC, 1995. p.1127-1138.

[GOL 95d] GOLEDZINER, Lia Goldstein; SANTOS, Clesio Saraiva dos. Versions and configurations in object-oriented database systems: a uniform treatment. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 7., 1995, Pune, Índia. **Proceedings...** New Delhi: Mcgraw-Hill, 1995. p.18-37.

[HEU 98] HEUSER, C. A. **Projeto de Banco de Dados.** Porto Alegre: Sagra Luzzato, 1998.

[JAD 95] AL-JADIR, Lina. et al. Evolution Features of the F2 OODBMS. In: INTERNATIONAL CONFERENCE ON DATABASE SYSTEMS FOR ADVANCED APPLICATIONS, 4., 1995, Singapore. **Proceedings...** [S.l.:s.n.] , 1995.

[JUN 86] JUNET, M.; FALQUET, G.; LÉONARD, M. ECRINS/86: An Extended Entity-Relationship Data Base Management System and its Semantic Query Language. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES CONFERENCE, 12., 1986. **Proceedings...** [S.1.: s.n.], 1986.

[KIM 88] KIM, Won; CHOU, Hong-Tai. Versions of schema for object-oriented databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 14., 1988, Los Angeles, Califórnia. **Proceedings...** [S.1.: s.n.,1988]. p.148-159.

[KIM 90] KIM, Won. **Introduction to Object-Oriented Databases.** Cambridge:MIT Press, 1990.

[KOR 93] KORTH, H. F.; SILBERSCHATZ, A. **Sistema de Bancos de Dados.** São Paulo: MAKRON Books, 1993. p.25-56.

[LAU 96a] LAUTEMANN, S. An Introduction to Schema Versioning in OODBMS. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 7., 1996, Zurik, Switzerland. **Proceedings...** [S.1.:s.n.], 1996.

[LAU 96b] LAUTEMANN, S. Integration of populated Schema Versions. In: ANUAL DATABASE CONFERENCE, 16., 1996. **Proceedings...** [S.1.:s.n.], 1996. p.147-158.

[LAU 97a] LAUTEMANN, S. Schema Versions in Object-Oriented Database Systems. INTERNATIONAL CONFERENCE ON DATABASE SYSTEMS FOR ADVANCED APPLICATION, 5., 1997, Melbourne, Austrália. **Proceedings...** [S.1.:s.n.], 1997. p.323-332.

[LAU 97b] LAUTEMANN, S. A Propagation Mechanism for Populated Schema Versions. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 13., 1997, Bermingham, UK. **Proceedings...** [S.1.:s.n.], 1997.

[MOT 96] 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.

[NAV 78] NAVATHE, S.B. **Schema Analysis for Database Restructuring.** San Jose, California, IBM Research Laboratory. New York: IBM, 1978.p.58. Research Report.

[NAV 92] NAVATHE, S. B.; BATINI, C.; CERI, S. **Conceptual Database Design:** an Entity-Relationship Approach. Redwood Clify. Califórnia: Addison Wesley, 1992. p.328-346.

[ODB 93] ODBERG, Erik. Schema Versioning and Class Hierarchy Modifications in Object-Oriented Databases. In: WORKSHOP ON SUPPORTING THE

EVOLUTION OF CLASS DEFINITIONS, 1993, Washington D.C., USA. **Proceedings...** [S.l.:s.n.], 1993.

[ODB 94a] ODBERG, Erik. Schema Modification Management for Object-Oriented Databases. In: CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, 6., 1994, Utrecht, The Netherlands. **Proceedings...** [S.l.]:Spring-Verlag, 1994.

[ODB 94b] ODBERG, Erik. Category Classes: Flexible Classification and Evolution in Object-Oriented Databases. In: CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, 6., 1994, Utrecht, The Netherlands. **Proceedings...** [S.l.]:Spring-Verlag, 1994. p.406-420.

[ODB 94c] ODBERG, Erik. A Global Perspective of Schema Modification Management for Object-Oriented Databases. In: INTERNATIONAL WORKSHOP ON PERSISTENT OBJECT SYSTEMS, 6., 1994, Tarascon, Provence, France. **Proceedings...** [S.l.:s.n.], 1994.

[ODB 94d] ODBERG, Erik. MultiPerspectives: The Classification Dimension of Schema Modification Management for Object-Oriented Databases. In: TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES AND SYSTEMS, 1994, Santa Barbara, California, USA. **Proceedings...** [S.l.:s.n.], 1994.

[O2T 91a] O_2 TECHNOLOGY. **The O_2 Application Designer's Manual**. Versailles: O_2 Technology, 1991.

[O2T 91b] O_2 TECHNOLOGY. **The O_2 Programmer's Manual**. Versailles: O_2 Technology, 1991.

[UNI 80] 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.

[UNI 97a] 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.

[UNI 97b] 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.

[WAG 91] WAGNER, F. R. et. al. STAR - Um ambiente para integração de ferramentas de projeto de sistemas digitais. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE MICROELETRÔNICA, 6., Belo Horizonte, 15-19 de julho de 1991. **Anais...** Belo Horizonte: SMICRO/UFMG, 1991. p.176-185.

[ZIC 91] ZICARI, Roberto. A framework for schema updates in an object-oriented database system. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 7., 1991, Kobe, Japan. **Proceedings...** Los Alamitos: IEEE, 1991. p.2-13.

Informática  UFRGS

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

" Um Modelo de Evolução de Esquemas Conceituais para Bancos de Dados Orientados a Objetos com o Emprego de Versões."

por

Renata de Matos Galante

Dissertação apresentada aos Senhores:

Prof. Dr. Duncan Dubugras Alcoba Ruiz (PUCRS)

Prof. Dr. Carlos Alberto Heuser

Profa. Dra. Nina Edelweiss

Vista e permitida a impressão.
Porto Alegre, 25 /03/99 .

Prof. Dr. Clesio Saraiva dos Santos,
Orientador.