Uma abordagem multi-nível para suporte a versões em bancos de dados orientados a objetos

Lia Goldstein Golendziner

Clesio Saraiva dos Santos*

Resumo

O uso de Sistemas de Bancos de Dados em aplicações não convencionais evidenciou requisitos não atendidos pelos sistemas tradicionais, motivando pesquisas em torno do Paradigma da Orientação a Objetos. Destaca-se o requisito de liberação do número de possíveis instâncias associadas a um mesmo objeto, para a representação do histórico de um objeto, ou para um tratamento flexível da evolução de esquemas, ou, ainda, para manutenção da consistência dos dados em utilização concorrente. Esta liberação conduziu ao conceito de *versão*. Este trabalho apresenta uma análise do conceito de versão frente aos demais conceitos já incorporados ao paradigma de orientação a objetos, discutindo a necessidade dos novos conceitos. São consideradas as relações entre versões, bem como os condicionamentos impostos pelas hierarquias de classes e tipos sobre as versões de objetos ascendentes e descendentes. É proposta uma arquitetura na qual é admitido o versionamento de objetos em qualquer nível da hierarquia de herança, em contraposição aos modelos que admitem apenas o versionamento nas folhas da hierarquia. É mostrado como o modelo proposto descreve com mais propriedade várias situações encontradas em aplicações.

Abstract

The use of Database Management Systems to support non conventional applications put in evidence a set of requirements not supported by the traditional database systems, and strongly motivated research work towards object oriented database systems. In this context, it was evident the need to liberate the number of instances associated to one database object, to represent the history of the object, to gain flexibility in schema evolution, or to maintain data consistency, in concurrent access environments, originating the version concept. This work presents an analysis of the version concept, when integrated with the already incorporated concepts of the object-oriented paradigm and discusses the need for new concepts. Aspects considered include: relationships between versions, as well as the conditions imposed by the class and type hierarchies over the versioning of objects and its ascendants and descendants. A multi-level architecture is proposed, which allows versioned objects to appear in any level of a type or class hierarchy. This approach is compared with the traditional one, in which versions appear only at the leaves of the hierarchies. It is shown how the proposed architecture allows modeling of many situations of real world applications in a more natural way.

* clesio@inf.ufrgs.br Instituto de Informática-UFRGS, Caixa Postal 15064
CEP 91501-970 Porto Alegre-RS

1. INTRODUÇÃO

Em bancos de dados orientados a objetos, entidades do mundo real são modeladas como objetos, apresentando um estado e um comportamento. Freqüentemente, há necessidade de guardar mais de um estado para um objeto, ocorrendo, em conseqüência, a incorporação do conceito de *versão*, como parte explícita da semântica do sistema. Uma versão representa um estado identificável de um objeto, considerado semanticamente significativo pelo usuário, e deve ser tratada uniformemente em um modelo de dados.

Muitos trabalhos de pesquisa têm sido desenvolvidos em relação a versões, procurando atender basicamente requisitos dos seguintes domínios de aplicações: aplicações de projeto de sistemas de engenharia (CAD-Computer Aided Design) e de software (neste caso, chamadas de CASE-Computer Aided Software Engineering) e de bancos de dados históricos.

Na área de aplicações de projeto, muitos trabalhos preocuparam-se com a representação de objetos de projeto [35, 2, 6, 26, 31, 17, 24, 18, 16], baseados em modelos semânticos, dos quais o de Entidades-Relacionamentos foi o mais utilizado. Os trabalhos [23, 32], apesar de ainda não abordarem o aspecto de versões, foram importantes pela introdução do conceito de objeto complexo e mecanismos associados: consulta a objetos complexos, transação de projeto, operações de *checkout* e *checkin*. Katz argumentou em [27] que muitas propostas apresentadas na área de suporte a aplicações de engenharia são semelhantes e propôs uma terminologia básica e uma coleção de mecanismos que devem estar presentes em qualquer abordagem para representar este tipo de informação.

Na área de CASE, a pesquisa aborda principalmente o aspecto de configurações de sistemas (ex: [4, 33]) e pouco foi feito utilizando sistemas de bancos de dados [3, 25, 13].

Em bancos de dados históricos, o objetivo é guardar informações sobre as entidades, organizadas em função do tempo. Diversos trabalhos foram realizados procurando acrescentar aspectos temporais a modelos de dados, inicialmente considerando o modelo relacional [41, 36, 9], e mais recentemente modelos de dados orientados a objetos [22, 42, 19].

Ultimamente, os trabalhos denotam uma preocupação em propor conceitos e mecanismos para controle de versões que possam ser incorporados a modelos de dados orientados a objetos. O objetivo é constituir um núcleo básico, que possa posteriormente ser refinado para atender às especificidades das diferentes classes de aplicações [8, 1, KIM 90, 39, 20, 14, 40]. Alguns trabalhos enfocam o uso de versões na evolução de esquema do banco de dados [8, 43, 11, 21].

Este trabalho concentra-se na gerência de versões no nível da aplicação, cujo objetivo é suportar a representação de informações dependentes de tempo ou de seqüenciamento, conforme definidas pelo usuário ou aplicação. A preocupação é com a integração do conceito de versão em bancos de dados orientados a objetos. São discutidos os conceitos novos que devem ser incorporados a um modelo de dados orientados a objetos. Ênfase especial é dada ao versionamento de objetos participantes de hierarquias de herança, bem como ao relacionamento entre as versões de objetos situadas em diferentes níveis de uma hierarquia.

É proposto um modelo de versões, no qual é permitido o versionamento de objetos em todos os níveis das hierarquias de herança, ao contrário do que ocorre na maioria dos

demais modelos, onde o versionamento de objetos é permitido somente nos folhas das hierarquias. É mostrado como estas extensões ao paradigma de orientação a objetos permitem a modelagem mais natural de muitas situações encontradas nas aplicações, em especial naqueles casos em que os objetos são gerados em um processo *top-down*.

O trabalho está organizado da seguinte maneira. A seção 2 apresenta os conceitos básicos relacionados a versões, considerando-se bancos de dados orientados a objetos. A seção 3 analisa o conceito de versão no contexto de hierarquias de classes e herança. A seção 4 apresenta alguns comentários finais, sendo a bibliografia relacionada na seção 5.

2. BASE CONCEITUAL

Nesta seção são apresentados os conceitos relativos a versões no contexto de bancos de dados orientados a objetos, que serão utilizados e referidos ao longo deste trabalho. É também discutida a necessidade do conceito de versão e de objeto versionado em adição aos demais conceitos já incorporados à idéia de orientação a objetos.

2.1 Versão e objeto versionado

Uma *versão* é a descrição de um objeto em um determinado momento de tempo, ou sob um determinado ponto de vista, cujo registro é importante para a aplicação considerada. Num modelo orientado a objetos, uma versão é um objeto de primeira classe. Como tal, possui seu próprio identificador de objeto (OID), o que permite que seja diretamente manipulada e consultada.

As versões de uma entidade do mundo real devem ficar agrupadas e constituem um *objeto versionado*, que é também um objeto de primeira classe (possui um OID), mantendo informações sobre as versões a ele associadas. Um objeto versionado pode apresentar propriedades que devem ser comuns a todas as suas versões. Cada versão só faz parte de um objeto versionado.

Como muitas vezes não é possível antecipar se objetos apresentarão versões ou não, objetos podem dinamicamente passar de não versionados a versionados.

Objetos (versionados ou não) que possuem as mesmas propriedades e comportamento podem ser agrupados em uma classe. A característica de ser ou não versionado é uma característica de cada objeto e não de sua classe. Assim, uma classe pode apresentar objetos não versionados e versionados.

Um automóvel sendo projetado pode ser considerado um objeto versionado, que é constituído de várias versões, representando as diversas etapas ao longo de todo o seu projeto. A figura 1 ilustra este exemplo, onde a notação usada é baseada na introduzida em [28].

Sendo objetos, tanto uma versão como um objeto versionado podem ser referidos por outros objetos, através do respectivo OID, bem como ser usados como argumentos de operações.

Cada objeto versionado possui uma versão que é considerada a *versão corrente*. A versão corrente é mantida automaticamente como a mais recentemente criada. O usuário pode especificar uma versão diferente para ser a corrente e, neste caso, a versão ficará fixa, não mudando com o surgimento de novas versões. A versão corrente é utilizada sempre

que o usuário solicitar uma operação sobre um objeto versionado sem especificar uma das suas versões.

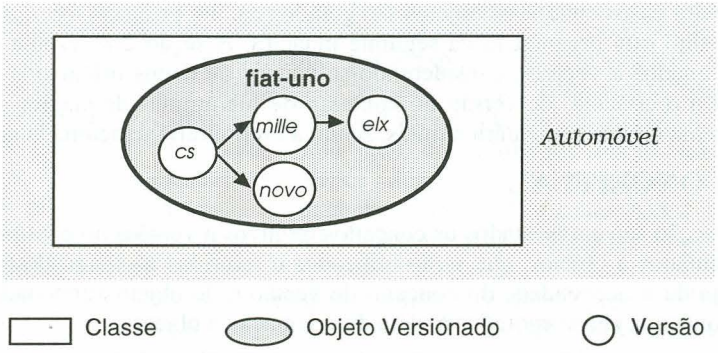

Figura 1 Objeto versionado e suas versões

2.2 PROPRIEDADES DE VERSÕES

Versões de um mesmo objeto versionado estão relacionadas através de um relacionamento de derivação, formando um grafo acíclico dirigido. Considerando a versão *mille* do exemplo, a versão *cs* é dita sua antecessora, e a versão *elx*, sua sucessora. Uma versão pode ter várias sucessoras e várias antecessoras.

Quando uma versão é criada como sucessora de uma já existente, ela é uma cópia de sua antecessora. Quando uma versão é criada como derivada de várias outras, ela é uma cópia da primeira indicada como antecessora, mas é mantido um relacionamento de derivação com as demais indicadas. Fica sob responsabilidade do usuário extrair as informações das outras versões, para que a nova versão seja uma combinação (*merge*) das diversas antecessoras, resultado de desenvolvimentos paralelos. A operação de combinar versões é uma operação difícil, não suportada até o momento por nenhum sistema que ofereça versões.

Cada versão tem um status, que reflete seu estágio de desenvolvimento e/ou consistência, podendo ser *em trabalho*, *liberada* ou *consolidada*, de maneira semelhante à definida em [29, 3]. Dependendo do status da versão, certas operações não são permitidas. Uma versão *em trabalho* é essencialmente uma versão temporária que ainda deve sofrer diversas alterações, antes de atingir um estado mais estável. Uma versão *liberada* é uma versão que já atingiu um estágio mais estável, de tal forma que pode ser compartilhada, e, conseqüentemente, não pode mais sofrer alterações, podendo, no entanto, ser removida. Uma versão *consolidada* é uma versão que chegou ao seu estágio final e, portanto, não pode mais ser alterada nem removida.

Uma versão é criada com status *em trabalho*. Quando uma versão é derivada de uma ou mais versões, suas antecessoras são automaticamente promovidas para *liberadas*, impedindo que sejam feitas modificações em versões que serviram de base para a construção de outras. O usuário pode promover explicitamente uma versão *em trabalho* para versão *liberada*, ou uma versão *liberada* para *consolidada*.

2.3 REFERÊNCIAS ESTÁTICA E DINÂMICA

Quando um objeto que apresenta versões é usado como componente de outro, referências a ele podem ser feitas de duas formas: referência a uma versão específica - chamada de *referência estática* - ou referência ao objeto versionado - chamada de *referência dinâmica* [29, 1] (ou genérica em [3]). Uma referência estática comporta-se como uma referência simples a um objeto, e o objeto composto é dito *estaticamente ligado* [29] à versão. Uma referência dinâmica significa que uma versão específica deve ser escolhida em tempo de execução para substituir a referência ao objeto versionado, e o objeto composto é dito *dinamicamente ligado* ao objeto versionado. Objetos compostos podem ser construídos recursivamente, formando uma *hierarquia de agregação.*

A substituição da referência a um objeto versionado pela referência a uma de suas versões é dita *resolução* da referência dinâmica. São necessários mecanismos para resolução de referências dinâmicas, que ocorre em dois momentos: 1) quando é solicitado o objeto referido (por exemplo, por uma operação **get_object**, em que o parâmetro é o OID do objeto versionado - vide seção 3.5) e 2) quando é construída uma configuração para o objeto composto.

No primeiro caso, é utilizada a *versão corrente* de um objeto. No processo de construção de configurações, são oferecidos diversos recursos para escolher uma versão associada a um objeto versionado. Por exemplo, a escolha pode ser baseada em critérios pré-definidos, como a primeira, a última, ou ainda em expressões envolvendo valores de atributos das versões. O processo de configuração não é objeto deste trabalho.

2.4 DISCUSSÃO: OBJETO VERSIONADO VERSUS CONJUNTO E CLASSE

O primeiro questionamento que se faz quando se trata de versões em modelos de dados orientados a objetos é se realmente o conceito de objeto versionado é um conceito novo requerido, ou se este pode ser substituído por algum outro já existente. Neste artigo, objeto versionado é considerado um novo conceito, não substituível por outro, pelas seguintes razões:

- uma *versão* é um representante de uma entidade do mundo real. Um *objeto versionado* agrupa todas as versões de uma mesma entidade do mundo real, de tal forma que cada uma das versões pode ser utilizada onde a entidade do mundo real é esperada;
- um objeto versionado deve possuir um OID, para permitir que seja referido, bem como utilizado como argumento de operações. Referências a um objeto versionado são necessárias em objetos compostos, em duas situações: a) quando o usuário deseja uma referência dinâmica, para que o sistema automaticamente defina a versão a ser utilizada como componente. Esta situação pode acontecer, por exemplo, quando o usuário quer que a versão componente seja sempre a última existente; b) quando o usuário quer selecionar manualmente o componente, mas, por alguma razão, não escolheu ainda a versão exata, ou a versão necessária ainda não existe. Em ambos os casos, o usuário fará uma referência a um objeto versionado, que será substituída, posteriormente, por uma referência a uma versão específica.

Poderia ser pensado que o conceito de objeto versionado é equivalente ao de conjunto ou de classe, já que estes conceitos também permitem o agrupamento de objetos com propriedades similares. Entretanto, existem diferenças substanciais.

Um conjunto de objetos é um objeto composto de outros do mesmo tipo, por exemplo, uma *equipe* de programadores é composta de diversos *programadores*. Existem duas diferenças, comparando o conceito de conjunto com o de objeto versionado:

1- o objeto *equipe* não é de mesma natureza que o objeto *programador*, enquanto um objeto versionado e qualquer uma de suas versões o são;

2- uma referência a um conjunto representa uma referência ao grupo correspondente de elementos, enquanto uma referência a um objeto versionado representa uma referência a uma de suas versões.

Uma classe agrupa objetos com as mesmas propriedades e comportamento. É um elemento do esquema do banco de dados e pode ser considerada como uma especificação de suas instâncias [5]. Uma classe pode aparecer na definição de outra classe, constituindo o domínio de um atributo da classe definida, ou na assinatura de uma operação, como domínio de um parâmetro. Nestes casos, a classe identifica os possíveis valores que podem ser assumidos pelo atributo ou parâmetro. Em tempo de execução, o atributo ou parâmetro conterá uma referência a uma das instâncias da classe, de forma similar ao que acontece com uma referência a um objeto versionado. No entanto, novamente existem diferenças quando se compara o conceito de classe (como forma de agrupar versões de uma entidade do mundo real) com o conceito de objeto versionado.

Consideremos a classe *Automóvel* (figura 2). Suponhamos agora que um novo automóvel *Y* deva ser projetado (não uma versão de **fiat-uno**), com a mesma estrutura e comportamento que um automóvel fiat-uno. Com o conceito de objeto versionado, a classe *Automóvel* apresenta um objeto versionado (**fiat-uno**), que é o agrupamento de todas as versões da entidade do mundo real fiat-uno; a operação necessária para representar o automóvel *Y* é a de criação de uma nova instância da classe *Automóvel*. Não havendo este conceito, a classe *Automóvel* estaria agrupando as versões de **fiat-uno** e a forma de definir a mesma estrutura e comportamento para um novo automóvel *Y* seria criando uma subclasse de *Automóvel*, para que as propriedades já definidas fossem herdadas.

Em sistemas onde classes não são objetos, uma classe não possui um OID e, conseqüentemente, não pode ser referida de um objeto composto. Além disto, objetos versionados são elementos da base de dados e a criação de um novo objeto implica apenas um acréscimo na extensão da base de dados. Subclasses são elementos do esquema da base de dados e a criação de uma nova subclasse é uma operação de modificação do esquema. Mesmo quando classes são objetos, e a criação de uma nova classe representa a criação usual de um novo objeto, pode ocorrer uma proliferação de classes, cada uma com relativamente poucas instâncias, o que não é desejável.

A segunda diferença envolve objetos compostos e é uma conseqüência da primeira. Suponhamos que existam dois objetos Oi e Oj com a mesma estrutura e comportamento, possuindo objetos versionados Vi e Vj, respectivamente, como componentes. Suponhamos ainda que Vi e Vj sejam de mesma natureza, e, portanto, podem ser instâncias da mesma classe V. Conseqüentemente, Oi e Oj podem ser instâncias da mesma classe, digamos O, que foi definida como tendo uma propriedade Y, cujo domínio é V. Se a representação de objetos versionados fosse através de classes, Vi e Vj seriam diferentes classes, fazendo com que Oi e Oj apresentassem diferentes estruturas, já que seus componentes são de classes distintas.

Uma outra diferença é relativa a classes, quando representam a coleção de suas instâncias. Neste sentido, uma classe está atuando como um conjunto e aplicam-se as mesmas diferenças mencionadas anteriormente, quando foram comparados os conceitos de conjunto e objeto versionado.

3 HIERARQUIAS DE OBJETOS E VERSÕES

Nesta seção são discutidas as repercussões advindas da possibilidade de versionamento de objetos em diferentes níveis de uma hierarquia de herança, bem como as vantagens que esta abordagem proporciona na modelagem de aplicações, onde a concentração das versões nas folhas da hierarquia conduz a distorções na captação da situação real.

3.1 HERANÇA

Herança é um dos conceitos básicos de modelos de dados orientados a objetos [5] e um dos mecanismos de reusabilidade. Existem duas maneiras pelas quais a herança pode ocorrer [7]: *refinamento* e *extensão*.

Refinamento modela o relacionamento *IS-A* entre uma classe e sua superclasse. Uma entidade do mundo real é modelada como um único objeto, que é instância da classe mais especializada da hierarquia de herança à qual pertence. Objetos de uma subclasse podem ser considerados como "casos especiais" de objetos da superclasse e podem ser utilizados onde objetos da superclasse são esperados. Por exemplo, pode ser definida uma classe *Veículo*, com subclasse *Automóvel*, como refinamento. Um objeto automóvel do mundo real será modelado como instância da classe *Automóvel*, podendo ser utilizado onde uma instância da classe *Veículo* é esperada. Este é o tipo de herança presente em vários sistemas orientados a objetos, tais como O_2 [15], ORION [30], GemStone [10] e KRISYS [34].

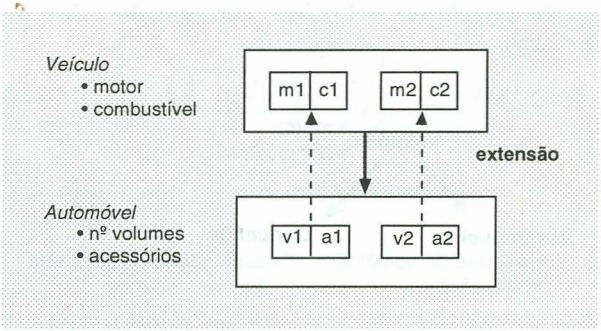

Figura 2 Herança por extensão: objeto na subclasse com ascendente na superclasse

Extensão está relacionada com a idéia de protótipos e aparece em modelos de dados como PEGASUS [7, 40] (extensão do EXTRA [12]), e modelo E [37, 38]. Cada objeto em uma subclasse C_2 possui um objeto associado na superclasse C_1, aqui denominado de *ascendente*. O objeto em C_2 é dito *descendente* do objeto em C_1 (em [7] o objeto em C_1 é denominado *protótipo* e o descendente é uma *extensão* do protótipo). Por exemplo, se a

classe *Automóvel* é definida como extensão da classe *Veículo*, cada objeto automóvel possui um objeto veículo como seu ascendente (figura 2). Este é o tipo de herança adotado neste trabalho, por ter-se mostrado mais adequado à modelagem de versões, conforme discutido nas seções seguintes.

3.2 CORRESPONDÊNCIAS ENTRE OBJETOS E VERSÕES

Quando a forma de herança utilizada é por refinamento, versões existem apenas na classe mais especializada da hierarquia de herança associada à entidade do mundo real sendo modelada [29, 3, 8]. No modelo proposto, onde a herança é por extensão[7], versões são admitidas em vários níveis da hierarquia de herança. Desta forma, a modelagem de entidades do mundo real pode ser feita em vários níveis de abstração, projetando ou modificando características de um objeto em um nível a cada vez.

Considerando o esquema definido na figura 3, o exemplo da figura 4 ilustra a modelagem de versões em mais de um nível da hierarquia de abstração. A entidade do mundo real **fiat-uno** é representada em dois níveis de abstração: no nível de *Veículo*, onde apresenta as propriedades *motor* e *combustível*, e no nível de *Automóvel*, onde apresenta as propriedades *nº volumes* e *acessórios*. Em cada um dos níveis, existem versões, associadas a objetos versionados. Cada versão deve possuir pelo menos um ascendente que lhe corresponda, isto é, quando uma versão é criada, obrigatoriamente deve ser ligada a um ascendente. Pode ocorrer que uma versão tenha mais de um ascendente, significando que as mesmas características definidas naquele nível podem ser ligadas a diferentes características no nível superior da hierarquia de herança. No exemplo, as mesmas características definidas para um **fiat-uno cs**, no nível *Automóvel*, podem ser ligadas a diferentes versões de **fiat-uno** no nível de *Veículo*, resultando em um **fiat-uno cs** a gasolina ou a álcool. Por outro lado, uma mesma versão em uma superclasse pode corresponder a mais de uma versão na subclasse. É o que acontece no exemplo (figura 4), onde uma versão de **fiat-uno** no nível de *Veículo* (ex: <1.0, A>) corresponde a duas versões no nível de *Automóvel* (<2,y> e <2,z>).

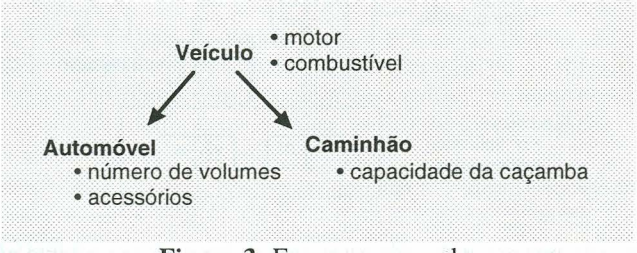

Figura 3 Esquema exemplo

Desta forma, o projeto de um novo automóvel pode ser desenvolvido em um nível de abstração e depois detalhado nos níveis inferiores da hierarquia de herança. No exemplo, um novo automóvel (ou caminhão) pode ser projetado inicialmente pensando-se em suas características a nível de *Veículo* e criando-se versões neste nível. Posteriormente, poderão

ser criadas versões no nível de *Automóvel*, relacionando-se cada versão criada com sua(s) versão(ões) ascendente(s).

Ficam assim estabelecidas *correspondências* (mapeamentos) entre versões de um objeto em uma classe e versões de seu ascendente na superclasse. Na figura 4, a correspondência é do tipo n:m, isto é, a cada versão na classe *Automóvel* podem corresponder n versões na superclasse *Veículo*, e a cada versão na classe *Veículo* podem corresponder m versões na subclasse *Automóvel*. A correspondência estabelece uma restrição de integridade, que é especificada quando da definição do relacionamento de herança entre uma classe e sua superclasse (definição do esquema), e deve ser mantida pelo sistema. Neste caso, quando a classe *Automóvel* é definida como subclasse de *Veículo*, define-se que a corrrespondência é n:m. A correspondência definida entre as versões em uma classe e aquelas em sua superclasse pode ser do tipo n:m (figura 4), 1:1, 1:n ou n:1.

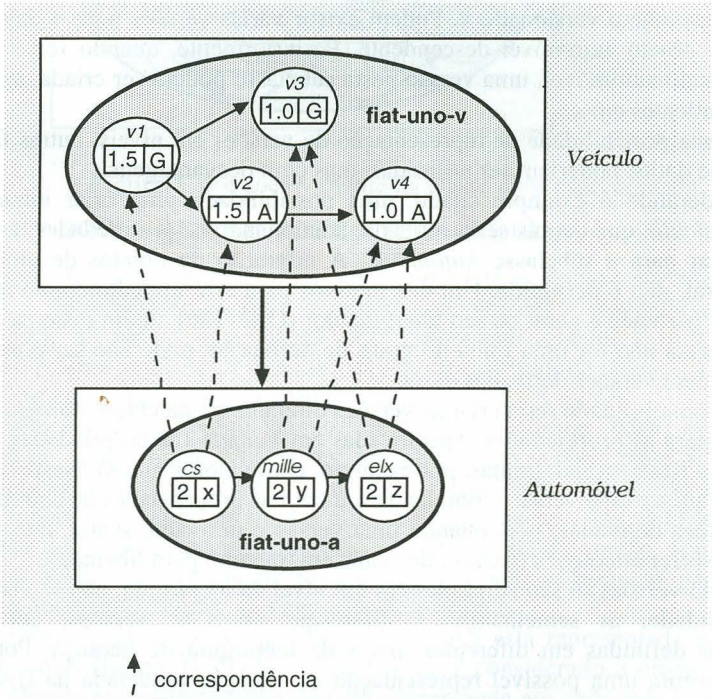

Figura 4 Versões representadas em mais de um nível da hierarquia de herança e suas correspondências

Quando é necessário buscar um objeto e todos os atributos herdados, a busca inicia na classe mais especializada, sendo escolhido um ascendente para cada superclasse relacionada. O ascendente pode ser explicitamente indicado, através de seu OID, ou especificado através de um dos critérios pré-definidos: **recent** (mais recente), **first**

(primeiro) ou **current** (corrente). O critério será usado sempre que houver mais de uma versão ascendente associada à versão (ou objeto) desejada(o).

Podem participar da hierarquia de herança objetos versionados e objetos não versionados, isto, é um objeto não versionado pode ser ascendente ou descendente de um objeto versionado. Objetos não versionados ou objetos versionados que não apresentam versões são considerados como uma versão, para efeitos de verificação de restrição de cardinalidade imposta pela correspondência.

3.3 DISCUSSÃO: REPRESENTAÇÃO DE VERSÕES NOS NÍVEIS

Com a possibilidade de representar versões de objetos em níveis distintos da hierarquia de herança, a modelagem de uma entidade do mundo real pode ser feita em vários níveis de abstração. Por exemplo, o projeto de um novo automóvel pode iniciar pelo projeto de suas características como veículo, sendo criada uma versão neste nível, associada a um objeto versionado **x**. Podem existir várias versões para o objeto **x**, mesmo sem existir o objeto automóvel descendente. Posteriormente, quando forem definidas as características do automóvel, uma versão nesta subclasse poderá ser criada, associada a um ou mais ascendentes em **x**.

Sem esta possibilidade de representação de versões em níveis, outras facilidades de um modelo de dados poderiam ser utilizadas, mas com desvantagens.

Considerando o exemplo citado, uma possibilidade seria criar inicialmente uma versão em *Veículo*, que depois seria refinada, adicionando-se propriedades de automóvel e deveria migrar para a subclasse *Automóvel*. A migração de objetos de uma classe para outra, em geral, não é permitida. Um dos motivos para este impedimento é que a classe à qual o objeto pertence é parte de seu identificador (OID) [30]. Além disto, se a versão que deve ser refinada não for uma folha no grafo de derivação, pode não ser possível removê-la, já que existem versões derivadas desta.

Outra possibilidade seria criar as versões diretamente na classe *Automóvel*, porém só com valores para as propriedades especificadas em *Veículo* (as propriedades especificadas em *Automóvel* ficariam indefinidas, por exemplo, com valores nulos). Neste caso, para que seja possível alterar uma versão, completando-a com as propriedades de automóvel, ela não deve ter versões derivadas, pois quando uma versão é derivada, sua(s) antecessora(s) não podem mais sofrer alterações (passam de status em trabalho para liberada).

Quando versões só são admitidas em um nível da hierarquia, não se visualiza, com a mesma facilidade, as semelhanças e diferenças entre as versões, com respeito a características definidas em diferentes níveis da hierarquia de herança. Por exemplo, a figura 5 apresenta uma possível representação da situação modelada na figura 4, porém com as versões representadas em um único nível.

Na representação de versões em níveis, obtém-se um agrupamento de versões de acordo com valores nelas presentes. Por exemplo, considerando a figura 4, a verificação de todas as versões de fiat-uno, no nível de Automóvel, que apresentem motor 1.0 e combustível gasolina, é feita buscando-se os descendentes da versão v2 (no nível de Automóvel). Versões em um nível podem ser consideradas como alternativas, para as quais são criadas versões nos níveis inferiores da hierarquia.

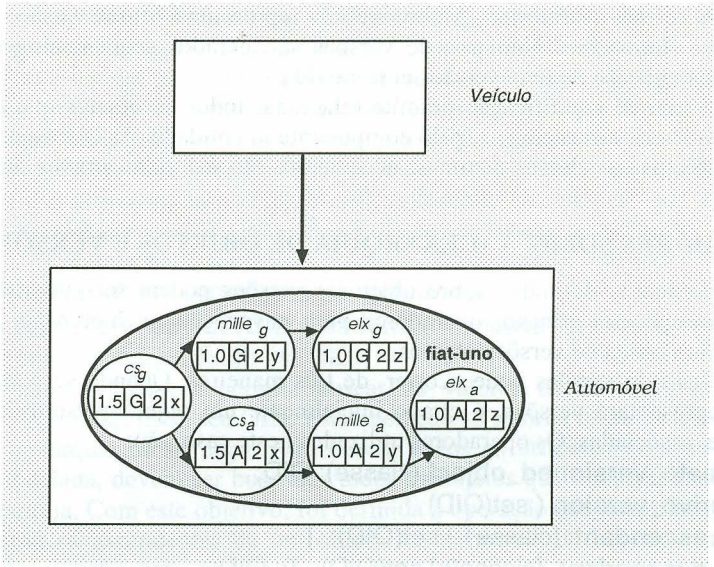

Figura 5 Versões somente na classe mais especializada

3.4 IDENTIFICADORES DE OBJETOS EM HIERARQUIAS

Uma entidade do mundo real é representada por vários objetos, em diferentes níveis da hierarquia. A correspondência entre os objetos ascendentes e descendentes é mantida pelo identificador do objeto, que possui um componente comum a todos os objetos representando a mesma entidade. A estrutura de um identificador de objeto 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. Assim, um objeto não versionado é identificado da mesma maneira que a primeira versão de um objeto versionado. A evolução de um objeto não versionado para versionado provoca a criação de um objeto versionado, não influindo na identificação do objeto existente, que passa a ser a primeira versão. Referências que existiam para o objeto não versionado passam a ser referências estáticas para a primeira versão do novo objeto versionado, não impactando o usuário.

Por exemplo, a entidade do mundo real fiat-uno está representada nos níveis de *Veículo* e *Automóvel* (figura 4). O objeto em *Veículo* é considerado o objeto raiz, e pode possuir descendentes em várias subclasses (um descendente em cada subclasse). No caso, existe um objeto descendente em *Automóvel*. As identificações destes objetos versionados são:

em *Veículo*: **<fiat-uno**, Veículo, nulo>

em *Automóvel*: **<fiat-uno**, Automóvel, nulo>,

onde **fiat-uno** está representando um identificador gerado pelo sistema para a entidade do mundo real fiat-uno.

Versões destes objetos possuem o componente número de versão com valores inteiros. Por exemplo: <fiat-uno, Veículo, 1> representa a primeira versão do objeto **fiat-**

uno na classe *Veículo* e <fiat-uno, Automóvel, 3> representa a última versão do objeto **fiat-uno** na classe *Automóvel*. Números de versões são gerados seqüencialmente e não são reaproveitados em caso de uma versão ser removida.

Esta forma de identificação permite relacionar todos os objetos que representam a mesma entidade em várias classes, pelo componente id entidade. As correspondências entre versões de objetos em classes distintas, no entanto, são mantidas através de atributos das versões.

3.5 OPERADORES SOBRE A HIERARQUIA DE OBJETOS E VERSÕES

Os operadores definidos sobre objetos e versões podem ser classificados em três tipos: operadores para criação, operadores para navegação na hierarquia de herança e operadores para busca de versões/objetos.

A criação de versões pode ocorrer de três maneiras. Quando se pode antever que um objeto apresentará versões, é criado inicialmente um objeto versionado e depois as versões a ele associadas. Os operadores utilizados neste caso, são:

create_versioned_object (classe): OID;
derive_version (set(OID)
[, **ascendant**: [classe1:] set(OID)...]
[, **descendant**: [classe2:] set(OID)...]): OID;

O operador **create_versioned_object** cria um objeto versionado, pertencente à classe indicada, sem nenhuma versão associada. A facilidade de criar um objeto versionado sem possuir ainda versões permite que o objeto versionado seja referido, por exemplo, para que um projeto seja realizado de forma *top-down*. O operador **derive_version** é utilizado posteriormente, para gerar versões. No caso de estar sendo criada a primeira versão, o parâmetro passado é o identificador do objeto versionado. Obrigatoriamente, uma versão deve ser ligada a um ascendente, que, se não for informado, será considerado como a versão corrente da superclasse. Opcionalmente, podem ser informados descendentes.

A segunda maneira de criar versões é derivar a partir de outra versão já existente e, desta forma, a nova versão é uma cópia desta. Pode ser informado um conjunto de versões como parâmetro, mas somente a primeira será copiada; as outras versões serão mantidas como antecessoras no grafo de derivação, mas é responsabilidade do usuário realizar as alterações necessárias na nova versão para que ela seja uma combinação das várias anteriores.

A terceira maneira é utilizada quando já existe um objeto (não versionado) e este deve passar a apresentar versões.Neste caso, a operação **derive_version** é aplicada sobre o objeto, e este passa a ser a primeira versão de um novo objeto versionado. A versão criada é cópia do objeto e passa a ser a segunda versão.

Os operadores para navegação na hierarquia de herança permitem que, a partir de um objeto, sejam buscados seus ascendentes e descendentes nas classes informadas. Os operadores são os seguintes:

get_ascendant (OID, classe[critério/"*"]): set (OID objeto ascendente);
get_descendant (OID, classe[critério/"*"]): set (OID objeto descendente);

Se há mais de uma versão ascendente para a versão solicitada, podem ser retornadas todas as versões (usando a opção *), ou apenas alguma, de acordo com um critério

fornecido. O critério poderá indicar uma escolha manual, sendo fornecido o OID da versão desejada, ou automática, informando uma das opções pré-definidas, para que o sistema selecione. Os critérios pré-definidos são: **recent**, **first** ou **current**. A opção **recent** indica que deverá ser escolhida a versão mais recente entre as correspondentes; a opção **first** indica que deverá ser a primeira (mais antiga) e a opção **current** indica que deverá ser a versão corrente. Se a versão corrente não for uma das ascendentes, um erro será acusado. O critério **recent** é o utilizado, caso nenhum seja informado. A operação **get_descendant** é análoga à **get_ascendant**, porém para os descendentes na subclasse definida.

A busca de objetos pode ser realizada através das seguintes operações:

get_object (OID): list(valores de atributos);
get_complete_object (OID [, **ascendant**: classe1 [: critério]
[, classe2 [: critério]]...]): list(OID);

O operador **get_object** permite buscar os valores de atributos de um objeto definidos em uma classe, dado seu OID. Esta operação só retorna os atributos definidos em um nível da hierarquia de herança. Para buscar todos os atributos de uma entidade do mundo real modelada, devem ser buscados todos os objetos que a representam, nos vários níveis da hierarquia. Com este objetivo, foi definida a operação **get_complete_object**. Esta operação retorna os ascendentes de um objeto na hierarquia de herança, um para cada superclasse. Opcionalmente, podem ser solicitados só os ascendentes de algumas superclasses, podendo ser especificadas as classes desejadas na cláusula **ascendant**. Se não especificado, são considerados todos os níveis até a raiz da hierarquia. Se o OID fornecido é de um objeto versionado, a operação é aplicada sobre a sua versão corrente.

Sempre que houver mais de uma versão ascendente correspondente, uma delas deverá ser selecionada através do critério fornecido, da mesma forma empregada nas operações **get_ascendant** e **get_descendant**.

A operação **get_complete_object**, em conjunto com a operação **get_object**, permite a busca de todos os atributos de uma entidade do mundo real representada neste modelo.

Além destas operações, estão definidas operações para navegação no grafo de derivação de versões (**get_first_version**, **get_last_version**, **get_successor**, **get_predecessor**, **get_versioned_object**).

4. CONSIDERAÇÕES FINAIS

Presentemente, há uma variada gama de aplicações para as quais o conceito de versão é considerado essencial. Entre estas encontram-se, por exemplo, aplicações de projeto e produção (CAD, CASE, CAM), automação de escritórios e hiperdocumentos.

Neste trabalho, o conceito de versão foi discutido no contexto de sistemas e modelos de bancos de dados orientados a objetos. Foi caracterizada a possibilidade da especificação de versões nos diferentes níveis de uma hierarquia de herança, liberando o projetista da necessidade de restringir todos os aspectos relativos ao versionamento de objetos ao nível mais baixo da hierarquia.

Foi mostrado como a liberação desta restrição possibilita a modelagem mais natural de situações reais, em particular aquelas onde se caracteriza um processo *top down* de construção de objetos de projeto, já que versões de objetos situados nos níveis mais altos da

hierarquia podem ser introduzidas mesmo antes da existência dos objetos situados nos níveis mais baixos (mais detalhados) da hierarquia.

Por outro lado, a possibilidade de versionamento nos vários níveis introduz a idéia de mapeamento entre versões correspondentes situadas em diferentes níveis das hierarquias de herança. Tais mapeamentos permitem uma representação mais concisa e natural das várias alternativas de configuração de objetos, pela escolha das versões mais adequadas em cada nível, sem a necessidade de representar explicitamente todas as combinações permitidas, como ocorre nos modelos onde as versões estão restritas às folhas da hierarquia.

Juntamente com a idéia de versões em vários níveis, foi apresentado um conjunto de operações que permite construir e explorar as representações obtidas a partir das aplicações. Elas permitem, entre outras ações, a navegação nas hierarquias definidas pelos mapeamentos entre versões, bem como a obtenção de configurações dos objetos versionados.

Como ficou evidenciado ao longo do trabalho, o conceito de versão é diferenciado dos demais conceitos presentes nos modelos e sistemas de bancos de dados, modelando um conjunto de situações que não seria adequadamente modelado de outra forma. Por outro lado, um grande número de questões relativas aos modelos e aos respectivos sistemas permanece em aberto, requerendo soluções, que poderão vir tanto daqueles que se dedicam às aplicações, quanto daqueles que desenvolvem pesquisas em bancos de dados.

Referências Bibliográficas

[1] AGRAWAL, R.; BUROFF, S.; GEHANI, N.; SHASHA, D. Object Versioning in Ode. In: Int. Conference on Data Engineering, 7., 1991, Kobe, Japan. Proceedings. p. 446-455.

[2] BATORY, D.S.; KIM, W. Modeling concepts for VLSI CAD objects. ACM TODS, v.10, n.3, p.322-346, Sept. 1985.

[3] BEECH, D.; MAHBOD, B. Generalized Version Control in an Object-Oriented Database. In: Int. Conference on Data Engineering, 1988, Los Angeles-EUA. Proceedings. p.14-22.

[4] BELKHATIR, N.; ESTUBLIER, J. Experience with a database of programs. Sigplan Notices, v.22, n.1, p.84-91, Jan 1987.

[5] BERTINO, Elisa; MARTINO, Lorenzo. Object-Oriented Database Systems: Concepts and Architectures. Workingham, England: Addison-Wesley Publishers Ltd., 1993.

[6] Berkel, T. et al. Modelling CAD-objects by abstraction. In: Int. Conference On Data And Knowledge Bases, 3., 1988, Jerusalem, Israel. Proceedings. p. 227-240.

[7] BILIRIS, A. Modeling design object relationships in PEGASUS. In: Int. Conference On Data Engineering, 6., 1990, Los Angeles-USA. Proceedings. p. 228-236.

[8] Björnerstedt, A. ; Hultén, C. Version control in an object-oriented architecture. In: Kim, W.; Lochovsky, F.H. (eds.). Object-Oriented Concepts, Databases, and Applications. ACM Press, chap. 18, p. 451-485, 1989.

[9] Blanken, H. Implementing version support for complex objects. Data & Knowledge Engineering , v.6, p. 1-25, 1991.

[10] BREITL, R. The Gemstone data management system. In: KIM, W.; LOCHOVSKY, F.H. (eds.). Object-Oriented Concepts, Databases, and Applications. ACM Press, p. 283-308, 1989.

[11] BYEON, K.J.; McLEOD, D. Towards the unification of views and versions for object databases. In: Int. Symp. on Object Technologies for Advance Software, Nov. 1993, Japan. Proceedings.

[12] CAREY, Michael J.; DeWITT, David J.; VANDENBERG, S. A data model and query language for EXODUS. In: ACM SIGMOD Conf., 1988, Chicago. Proceedings.

[13] Cellary, W.; VOSSEN, G.; Jomier, G. Multiversion object constellations: a new approach to support a designer's database work. Giessen, Universität Giessen, Nov. 1991. (Bericht Nr. 9105)

[14] DAVID, M.B.; SCHIEL, U. O esquema de versões do modelo temporal de objetos (TOM). In: Simpósio Brasileiro de Banco de Dados, 7., 1992, Porto Alegre. Anais. p. 339-350.

[15] DEUX et al. The story of O2. IEEE Transactions on Knowledge and Data Engineering, v.2, n.1, p.91-108, Mar. 1990.

[16] DIAS, E.Z.V.; MAGALHÃES, G.C. MVC: Um modelo para controle de versões e configurações. In: Simpósio Brasileiro de Engenharia de Software, 5., 1991. Anais. p.93-106.

[17] Dittrich, K.R.; Gotthard, W.; Lockemann, P.C. DAMOKLES-a database system for software engineering environments. Springer-Verlag, Lecture Notes in Computer Science 244, p. 353-371.

[18] DITTRICH, K.; LORIE, R. Version support for engineering database systems. IEEE Transactions on Software Engineering, v.14, n.4, p. 429-437, Apr. 1988.

[19] EDELWEISS, N.; OLIVEIRA, J.P.M. de; PERNICI, B. An Object-Oriented Temporal Model. In: CAISE'93, 5., Paris, 1993. Proceedings.p.397-415. (Lecture Notes in Computer Science, n.685).

[20] Fauvet, M.C. Versions and histories in Object-Oriented Applications. In: Simpósio Brasileiro de Banco de Dados, 7., Porto Alegre. Anais. p. 319-337.

[21] FORNARI, M.R.; GOLENDZINER, L.G. Evolução de esquemas utilizando versões em bancos de dados orientados a objetos. In: Simpósio Brasileiro de Banco de Dados, 8. Anais. p. 113-127.

[22] GREENSPAN, S.J.; BORGIDA, A.; MYLOPOULOS, J. A Requirements modeling language and its logic. In: M.L. BRODIE; J. MYLOPOULOS (eds.) On Knowledge Base Systems. Springer-Verlag, New York, 1986. p.471-502.

[23] HASKIN, R.L.; LORIE, R.A. On extending the functions of a relational database system. In: ACM SIGMOD conf., 1982, Orlando. Proceedings. p.207-212.

[24] HUDSON, S.E.; KING, R. Object-oriented database support for software environments. In: ACM SIGMOD conf., May 27-29, 1987, San Francisco, CA. Proceedings. p.491-503.

[25] HUDSON, S.E.; KING, R. The Cactis project: database support for software environments. IEEE Transactions on Software Engineering. New York, v.14, n.6, p. 709-719, June 1988.

[26] KATZ, R.; CHANG, E.; BHATEJA, R. Version modelling concepts for computer aided databases. In: ACM SIGMOD conf., May 28-30, 1986, Washington, D.C. Proceedings. p.379-386.

[27] Katz, R.H. Toward a unified framework for version modeling in engineering databases. ACM Computing Surveys, New York , v.22, n.4 , p. 375-408, Dec. 1990.

[28] Kim, W.; Banerjee, J.; Chou, H.T.; Garza, J.F. ; Woelk, D. Composite object support in an object-oriented database system. In: OOPSLA'87, October 4-8. Proceedings. p. 118-125.

[29] Kim, W.; Bertino, E. ; Garza, J.F. Composite objects revisited. In: ACM SIGMOD conf., May 31- June 2, 1989, Oregon. Proceedings. p.337-347.

[30] KIM, W. et al. Features of the ORION Object-Oriented Database System. In: KIM, W.; LOCHOVSKY, F.H. (eds.). Object-Oriented Concepts, Databases, and Applications. ACM Press, chap. 11, p. 251-282, 1989.

[31] KLAHOLD, P.; SCHLAGETER, G.; WILKES, W. A general model for version management in databases. In: VLDB'86, 12., Aug. 1986, Kyoto, Japão. Proceedings. p. 319-327.

[32] LORIE, R.L.; PLOUFFE, W. Complex objects and their use in design transactions. In: ACM SIGMOD conf., June, 1983, San Jose, Calif. Proceedings. p.115-122.

[33] MAHLER, A.; LAMPEN, A. An integrated toolset for engineering software configurations. In: ACM-SIGSOFT-SIGPLAN Symposium on Practical Software Development Environments, Boston, MA, 1988. Proceedings.

[34] MATTOS, Nelson Mendonça. An approach to knowledge base management-requirements, knowledge representation and design issues. Kaiserslautern, University of Kaiserslautern, 1989. (Lecture Notes in Artificial Intelligence 513, Springer, 1991).

[35] McLEOD, D.; NARAYANASWAMY, K.; BAPA RAO, K. An approach to information management for CAD/VLSI applications. In: ACM CONF. on Databases for Engineering Applications, May 1983, San Jose, CA. Proceedings. p.39-50.

[36] ROWE, L.A.; STONEBRAKER, M.R. The POSTGRES data model. In: VLDB'87, 13., Sept. 1987, Brighton, Ingl. Proceedings. p. 83-96.

[37] SANTOS, Clesio Saraiva dos; OLIVEIRA, José Palazzo Moreira de; CASTILHO, José Mauro Volkmer de. O modelo E. In: Seminário Integrado de Software e Hardware, 13., jul. 19-25, 1986, Olinda. Anais. Recife, SBC/UFPE, 1986. p.342-350.

[38] SANTOS, Clesio Saraiva dos. O modelo E revisado. Porto Alegre: CPGCC da UFRGS, 1989. (Relatório de Pesquisa 119)

[39] SCIORE, E. Using annotations to support multiple kinds of versioning in an object-oriented database system. ACM TODS, New York, v.16, n.3, p.417-438, Sept. 1991.

[40] SCIORE, E. Versioning and configuration management in an object-oriented data model. VLDB Journal, v.3, p. 77-106, Jan. 1994.

[41] SNODGRASS, R.; AHN, I. A taxonomy of time in databases. In: ACM SIGMOD conf., May 1985, Austin. Proceedings.

[42] WUU, G.T.D. ; DAYAL, U. A uniform model for temporal and versioned object-oriented databases. In: Temporal Databases: Theory, design and implementation. Bridge Parkway: Benjamin/Cummings, 1993. p. 230-247.

[43] ZDONIK, S. Object-Oriented Type Evolution. In: BANCILHON, F; BUNEMAN, P.(eds). Advances in Database Programming Languages. Reading, Mass.: Addison-Wesley, p.277-288, 1990.

Artigo originalmente publicado em: CONGRESSO da SOCIEDADE BRASILEIRA de COMPUTAÇÃO, 15., 29 jul.-4 ago. 1995, Canela-RS. **Anais**. Porto Alegre: SBC, 1995. p. 1127-1138