Modelo Temporal de Versões

Mirella Moura Moro 1

Nina Edelweiss 1

Clesio Saraiva dos Santos 1

Resumo: Esse trabalho apresenta uma alternativa para a união de dados temporais e um modelo de versões. O resultado, o Modelo Temporal de Versões – TVM, armazena as versões de objetos e, para cada versão, o histórico dos valores de seus atributos e relacionamentos dinâmicos. O TVM é ideal para modelar sistemas que evoluem no tempo e que necessitam gerenciar alternativas de projeto como versões. Uma interface para modelar as classes no Modelo também é apresentada.

Palavras-chave: Modelagem conceitual, Modelo Temporal, Modelo de Versões

Abstract: This work presents an alternative for the union of temporal data and a version model. The result, the Temporal Versions Model, is able to store object versions and, for each version, the history of its dynamic attributes and relationships values. TVM is ideal for modeling time-evolving systems that need to manage design alternatives as versions. An interface for modeling TVM classes is also presented.

1 Inst. de Informática, UFRGS - Av. Bento Gonçalves, 9500- PoA-RS Cx.P.:15064 CEP 91501-970
{mirella, nina, clesio @inf.ufrgs.br}

1 Introdução

Em aplicações de projeto, diferentes alternativas ou versões de um projeto são mantidas no banco de dados. Historicamente, as primeiras pesquisas relacionadas a versões se encontram nas áreas de CAD (*Computer Aided Design*), CASE (*Computer Aided Software Engineering*) e SCM (*Software Configuration Management*) [4],[6],[11]. Posteriormente, esse conceito foi se estendendo para outros domínios de aplicação, tais como projetos concorrentes, evolução de esquemas, XML e versões de documentos [3].

Uma versão descreve um objeto em um período de tempo ou sob um certo ponto de vista. Embora a utilização de versões armazene as alternativas de projeto, nem todo o histórico das alterações realizadas sobre os dados é registrado. Modificações importantes podem ter sido realizadas, influindo de alguma maneira no desenvolvimento geral, sem que seja possível o acesso posterior a todos os valores passados. Além disso, as informações relativas ao tempo no qual as especificações e alterações foram realizadas não são armazenadas. O usuário pode estar interessado em recuperar o estado do projeto relativo a um período ou data específica, incluindo dados que foram descartados por qualquer motivo. Esse tipo de recuperação não é possível apenas com o uso de versionamento, pois o histórico completo somente é acessível através de um modelo de dados temporal.

Um modelo de dados temporal especifica os aspectos estáticos e dinâmicos da aplicação. Por definição, um banco de dados temporal deve seguir o princípio de que todos dados excluídos pelo usuário são mantidos a fim de preservar o histórico completo. Diversos modelos temporais foram propostos nos últimos anos, sendo a maioria extensão de modelos existentes [5],[7],[26].

Gerenciar versões (unindo, separando, descartando e justificando possibilidades erradas) tende a diminuir custos e aumentar a qualidade de produtos finais. Apresentar versões com seus respectivos aspectos temporais adiciona a vantagem de obter informações de projeto relacionadas a períodos específicos. O objetivo deste trabalho é definir um modelo que utilize as características de versões e tempo para permitir o armazenamento das versões de um objeto e, para cada uma de suas versões, o histórico de seu tempo de vida e das alterações feitas em seus atributos e relacionamentos dinâmicos. Esse modelo é denominado Modelo Temporal de Versões (TVM – *Temporal Versions Model*) [14],[15].

O uso de um modelo de dados semanticamente rico não requer necessariamente a existência de um SGBD (Sistema Gerenciador de Banco de Dados) próprio. A tendência é implementar o modelo sobre bancos convencionais, através do mapeamento das informações temporais. Como objetivo complementar, neste trabalho é proposto também um Ambiente para o funcionamento do TVM sobre um banco de dados convencional.

O trabalho está organizado da seguinte maneira. A seção 2 apresenta o Modelo Temporal de Versões. O ambiente para o TVM é ilustrado na seção 3. A seção 4 mostra um estudo de caso. Finalmente, a seção 5 apresenta as conclusões.

2 Modelo Temporal de Versões

O conceito de versão permite que o usuário mantenha diferentes alternativas de projeto. Versões podem ser definidas como estágios distintos de um objeto em desenvolvimento, em diferentes estados, que compartilham algumas características comuns. Há dois tipos de versões, de acordo com o nível no qual o versionamento é aplicado: versões de classes (evolução das classes), e versões de instância (modificações das propriedades dentro das instâncias). Somente o segundo tipo de versionamento é considerado no TVM.

Versionamento de instâncias implica que diferentes versões do mesmo objeto diferem apenas pelos valores de algumas de suas propriedades. Entretanto, somente o versionamento não é suficiente para armazenar o histórico de todas as alterações nas instâncias com versões. Somente os valores identificados explicitamente pelo usuário como nova versão são armazenados. A adição da dimensão temporal proporciona o armazenamento de toda a evolução de valores de atributos e relacionamentos das instâncias. O modelo proposto (TVM) é baseado nos conceitos de versionamento de instância e tempo, permitindo o armazenamento de versões de objeto e, para cada versão, seu tempo de vida e o histórico das alterações realizadas em seus elementos.

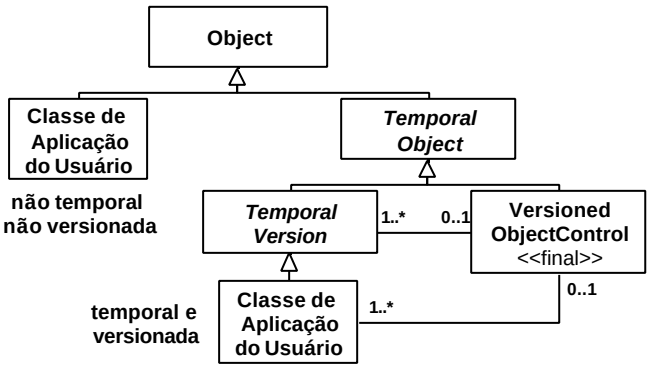

Figura 1. Hierarquia de classes do TVM com as classes de aplicação

A Figura 1 apresenta a hierarquia base do TVM. Basicamente, a hierarquia do TVM permite especificar dois tipos de classes da aplicação:

- classe de aplicação não temporal e não versionável – definida como subclasse de *Object*. Sua modelagem pode ser usada para contemplar classes que representam tabelas de uma base já existente ou classes auxiliares nas quais os conceitos de tempo e versão não são necessários;
- classe de aplicação temporal versionada – definida como subclasse de *TemporalVersion*. Possui um relacionamento de associação com a classe *VersionedObjectControl*. Seus atributos e relacionamentos podem ser definidos como estáticos ou temporais.

Verificando essa hierarquia, a possibilidade de criar classes temporais diretamente de *TemporalObject* poderia ser cogitada. Essa hipótese foi descartada porque o objetivo principal é modelar e proporcionar ambas características de tempo e versões nos objetos. O Modelo permite somente os dois tipos de classe descritos. Além disso, a derivação de versões é realizada explicitamente pelo usuário, e é o usuário quem define quais atributos e relacionamentos terão seus históricos armazenados. Em outras palavras, o Modelo deixa a critério do usuário a utilização de somente tempo, somente versões, ou ambos conceitos simultaneamente.

As classes *TemporalObject* e *TemporalVersion* são *classes abstratas*, não podendo ser instanciadas diretamente. A classe *VersionedObjectControl* só pode ser instanciada pelo sistema gerenciador com o intuito de administrar os objetos que possuem versões. Além disso, essa classe é denominada *classe final*, não sendo permitido especializá-la em subclasses.

A classe *TemporalObject* possui métodos para gerenciar os rótulos temporais. A classe *TemporalVersion* contém os atributos *configuration* e *status* (que informam se a versão pertence a uma configuração e seu status, respectivamente) e os atributos para navegação na hierarquia de extensão (*ascendant* e *descendant*) e na hierarquia de derivação (*predecessor* e *successor*). Finalmente, a classe *VersionedObjectControl* contém os atributos para informar a versão corrente, os números de configurações e versões, a primeira e a última versão, o número da próxima versão, e se o usuário especificou uma versão como corrente ou não. Todas as classes contêm métodos para obter e atualizar os atributos e métodos com funções específicas, detalhadas na referência [17].

No TVM, o tempo é associado a objetos, versões, atributos e relacionamentos, permitindo uma melhor e mais flexível modelagem da realidade. O usuário pode definir atributos e relacionamentos como estáticos (quando não possuem a variação dos valores armazenada) ou temporais (todas as alterações nos valores são armazenadas, criando seus históricos). Uma classe pode apresentar atributos e relacionamentos de ambos os tipos.

Um objeto temporal versionado contém uma linha de tempo para cada versão. O fato de muitas versões do mesmo objeto poderem coexistir gera a possibilidade de duas ordens no tempo: (i) tempo ramificado para um objeto, devido às diferentes linhas de tempo de cada versão originada da linha do objeto; e (ii) tempo linear para cada versão. A variação temporal é discreta, e a temporalidade é representada no modelo através do rótulo de tempo elemento temporal (conjunto de intervalos temporais), bitemporal (o intervalo de validade armazena o histórico da evolução da realidade, enquanto o intervalo de transação armazena a seqüência dos estados das classes temporais na base) e implícito.

Versões de uma mesma entidade devem ser mantidas juntas. O agrupamento de versões de um mesmo objeto constitue um objeto versionado. Instâncias de classes podem ser: um objeto versionado, uma versão, ou um objeto sem versões. Resumindo, a Figura 2 apresenta graficamente os objetos instanciados da classe de aplicação temporal versionada *CL* (subclasse de *TemporalVersion*). São apresentados um objeto sem versões (*Obj1*) e um objeto versionado (*Obj2*) com suas respectivas versões (*V1*, *V2*, *V3*, *V4*).

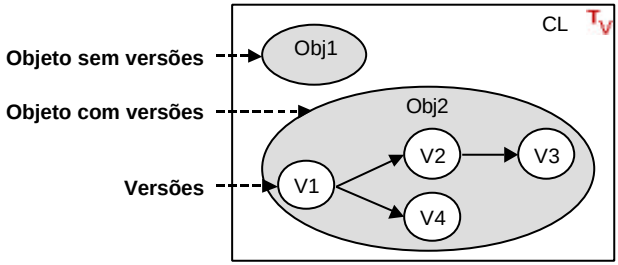

Figura 2. Objetos instanciados de uma subclasse de *TemporalVersion*

Versões em um objeto versionado são relacionadas através de um relacionamento de derivação, definindo um grafo acíclico dirigido. Em um objeto versionado existe sempre uma versão corrente, mantida pelo sistema como a mais recente à medida em que novas versões vão sendo criadas. Se o usuário especificar uma outra versão como sendo a corrente, essa permanece fixa. Sempre que o usuário envia uma mensagem a um objeto versionado sem especificar a versão, a versão corrente é utilizada.

Durante seu tempo de vida, versões podem passar por diferentes *status.* As transições entre eles e os eventos que ocasionam tais transições estão representados no diagrama de estados na Figura 3. Cada *status* define o conjunto de operações que podem ser aplicadas sobre a versão, mantendo a consistência da evolução armazenada, conforme a Tabela 1.

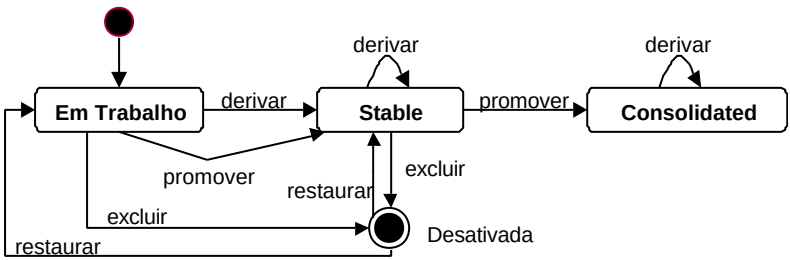

Figura 3. Diagrama dos *status* de uma versão

Tabela 1. Operações que podem ser executadas em cada *status* das versões

Operações / *Status*	***Em Trabalho***	***Estável***	***Consolidada***	***Desativada***
Derivar	**sim**	**sim**	**sim**	–
Promover	**sim**	**sim**	–	–
Alterar	**sim**	**não**	**não**	**não**
Excluir	**sim**	**sim**	**não**	**não**
Consultar	**sim**	**sim**	**sim**	**sim**
Compartilhar	**não**	**sim**	**sim**	**não**
Restaurar	–	–	–	**sim**

Uma das características mais importantes do Modelo Temporal de Versões é a definição de herança por extensão, na qual as versões são admitidas nos vários níveis da hierarquia de herança. Com esse recurso, um objeto pode ser desenvolvido em um nível de abstração e posteriormente detalhado nos níveis inferiores da hierarquia. Isso possibilita que a modelagem de entidades do mundo real seja feita em vários níveis, projetando ou modificando características de um objeto em uma camada de cada vez.

Deve-se evitar confusões com a herança adotada nas linguagens e modelos orientados a objetos, denominada de herança por refinamento. A primeira diferença entre essas heranças está na finalidade da modelagem. Na especificação de sistemas ou na modelagem de dados, essas heranças são definidas em tempo de análise ou projeto. A herança por refinamento traz as vantagens de extensibilidade, pela redefinição de estado e comportamento nas subclasses durante o projeto, e de reuso de código na implementação. A herança por extensão é definida quando é detectada a necessidade de instanciar a entidade em diferentes níveis de detalhamento (em tempo de execução), como explicado anteriormente.

A segunda diferença está no aspecto de funcionamento das hierarquias. Quando um tipo *Y* refina um tipo *X*, uma instância de *Y* mantém seu próprio armazenamento de todos os valores definidos por *Y* e *X*. Quando um tipo *Y* extende *X*, a seguinte afirmativa é verdadeira: para toda instância *y* de *Y* há uma instância *x* em *X* cujos valores dos atributos são herdados (compartilhados) por *y*. Nesse caso, acrescenta-se um ponteiro em *y* que referencia a respectiva instância *x*, conforme apresentado na Figura 4. Esse conceito é apresentado por Biliris, em [2].

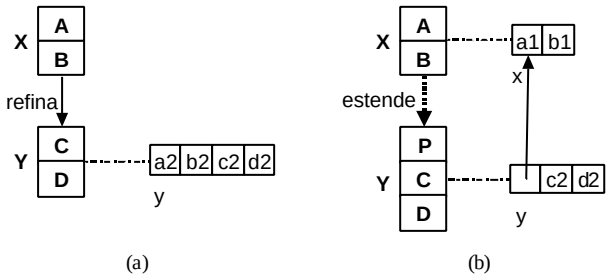

Figura 4. (a) Refinamento: instâncias de *Y* mantêm seu próprio armazenamento para os atributos definidos em *Y* e *X*. (b) Extensão: todos os atributos de *x* são compartilhados por *y*.

A linguagem de definição de classes para o TVM está completamente especificada em [14]. A Figura 5 apresenta uma simplificação dessa linguagem. Ela permite a definição de todas as características do Modelo: classes temporais versionadas (*HasVersions*), herança (*inherit*), classes agregadas (*aggregate_of*), atributos e relacionamentos temporais (*temporal*), relacionamentos inversos (*inverseRelationshipName*) e operações. A seção 4.2 ilustra a utilização dessa linguagem em uma aplicação.

O TVM possui muitas outras características que são apresentadas nas referências [17],[18],[19],[20].

```
Class className [ HasVersions ] [ inherit className ]
  [[ temporal ] aggregate_of [n] className ( by value | by reference )
  {, [ temporal ] aggregate_of [n] className ( by value | by reference )} ]
  ( [ Properties:
        [ temporal ] attributeName : attributeDomain ;
        { [ temporal ] attributeName : attributeDomain ; } ]
    [ Relationship:
        [  temporal ] relationshipName (1:1 | 1:n | n:1 | n:n)
                [ inverse inverseRelationshipName ] relatedClassName ;
        {[ temporal ] relationshipName (1:1 | 1:n | n:1 | n:n)
                [ inverse inverseRelationshipName ] relatedClassName; } ]
    [ Operations: { operationDefinitions } ] ) ;
```

Figura 5. Linguagem de Definição de Classes do TVM

A referência [18] apresenta um estudo comparativo com os modelos OODAPLEX [30] e TVOO [22]. Em resumo, o TVM propõe uma hierarquia como base para especificação de classes, enquanto o TVOO utiliza um *framework* formal, e a extensão do OODAPLEX trabalha com uma hierarquia de tipos abstratos temporais. Tanto o TVM quanto o OODAPLEX permitem objetos temporais versionados entre os objetos normais, enquanto o TVOO especifica que todos os objetos são temporais. TVM e OODAPLEX possuem ambos os tempos de validade e transação, enquanto o TVOO apresenta apenas o tempo de transação. Considerando versões, cada alteração gera nova versão no TVOO, enquanto o usuário define explicitamente novas versões no TVM. Adicionalmente, o TVM armazena todas as alterações dos atributos e relacionamentos definidos pelo usuário como temporais. Essas últimas características são importantes para limitar o espaço de dados e fornecer um desempenho melhor. Finalmente, os três modelos possuem exclusão lógica sendo que objetos excluídos não podem ser reativados no TVOO.

É importante notar que esses três modelos gerenciam versões de dados armazenados em bases de dados, sendo que no TVM é o usuário quem define novas versões. Existem aplicações que controlam versões de arquivos ou documentos através do sistema operacional ou de protocolos *web*. Entre elas citam-se: WebDav, padrão para autoria colaborativa na *web* [29]; DeltaV, versionamento sob a *web* estendendo o WebDav [13]; CVS II (*Concurrent Versions System*), revisão de arquivos e liberação de código fonte em ambiente de multi-desenvolvedores [1]; e RCS (*Revision Control System*) e seu sucessor RCE (*Revision Control Engine*), armazenamento, recuperação, log e identificação de revisão de arquivos (em particular programas, documentação e dados teste) [27],[28]. É necessário que essa diferenciação fique bem clara, pois é comum confundir esses dois tipos de uso de versões.

3 Ambiente Temporal de Versões

Estabelecer uma arquitetura de SGBD integrada para implementar um modelo de dados que estende a SQL (*Structured Query Language*) com suporte a tempo é uma tarefa de alto custo, que somente os maiores fabricantes de banco de dados podem financiar. Então, o fato de já existir um SGBD que gerencie grandes quantidades de dados sugere uma

alternativa melhor: prover suporte para aplicações de tempo e versões embutido por meio de uma camada entre um SGBD existente e a aplicação [5].

Seguindo essa idéia, esse trabalho propõe o Ambiente Temporal de Versões (Figura 6) para implementar o TVM sobre um banco de dados convencional. O Ambiente Temporal de Versões possui um conjunto de interfaces para modelar classes, gerenciar e consultar instâncias. Especificamente, na interface para a definição de classes, o usuário pode especificar suas classes e mapear a especificação para a Linguagem de Definição de Classes do TVM e SQL, a fim de criar o sistema no banco de dados. A Figura 7 ilustra essa interface e a parte de um exemplo simples. Ao lado esquerdo está a estrutura em forma de árvore que apresenta classes, atributos, relacionamentos e operações. Ao lado direito está a linguagem de definição do TVM para a aplicação.

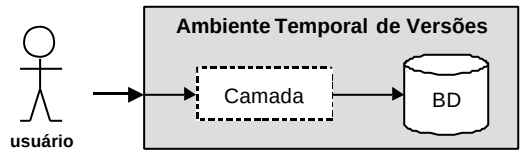

Figura 6. Ambiente Temporal de Versões

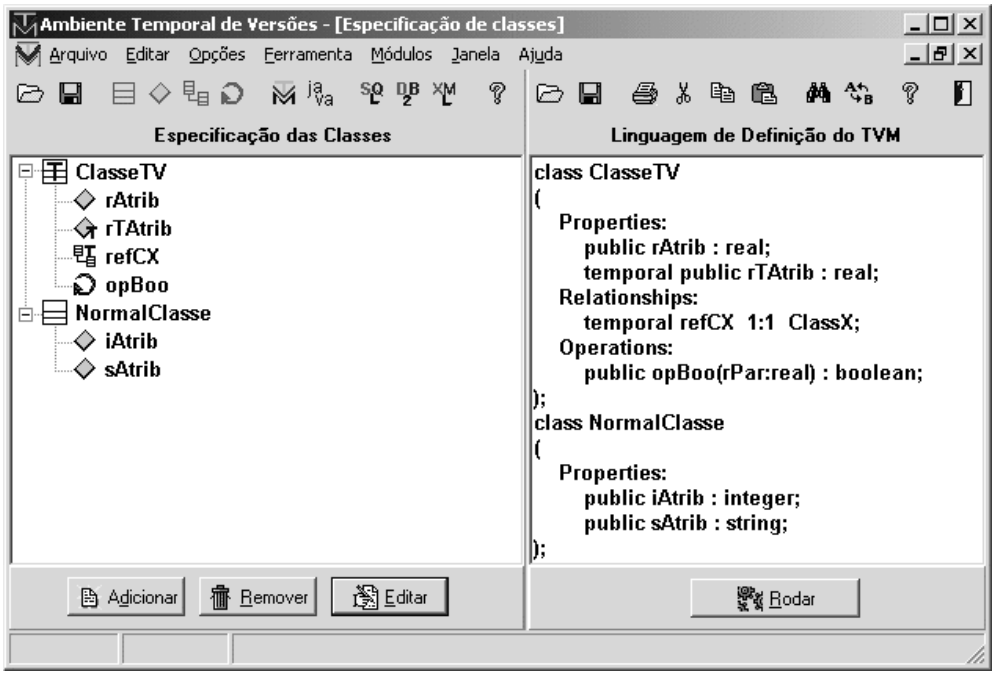

Figura 7. Ambiente Temporal de Versões: Ferramenta de Especificação de Classes

O usuário especifica os detalhes de cada elemento em interfaces próprias. Os detalhes da classe são equivalentes às características apresentadas na linguagem de definição: nome da classe, tipo (normal ou temporal versionada), herança, correspondência, classe abstrata ou final, e agregação. De mesmo modo, para a especificação de atributos são apresentados os campos: nome do atributo, tipo, visibilidade, escopo, valor padrão e se o mesmo é temporal. Para relacionamentos são apresentados: nome do relacionamento, nome da classe relacionada, nome do relacionamento inverso, cardinalidade e se o relacionamento é temporal. Finalmente, para operações são apresentandos: nome da operação, visibilidade, escopo, tipo de especialização, parâmetros e tipo retornado.

4 Estudo de Caso

Nessa seção, um estudo de caso é apresentado em três partes: representação do mapeamento no DB2, modelagem da aplicação e representação gráfica das instâncias.

4.1 Mapeamento para o DB2

Como experiência para a implementação do mapeamento do TVM em um banco de dados objeto-relacional, o banco de dados *DB2 Universal Database* (*UDB*) foi escolhido para ser base para o funcionamento do TVM. Um dos motivos que levaram à escolha desse SGBD é o fato de ser o mais próximo ao padrão SQL-92 no seu suporte aos tipos de dados temporais [25]. O mapeamento das classes do TVM é dividido em duas partes: representação da hierarquia do modelo e mapeamento das classes de aplicação [17].

Com base nos métodos *Observer* e *Mutator*, no uso de funções definidas pelo usuário (UDF), procedimentos armazenados e gatilhos (*triggers*), é possível generalizar o mapeamento do Modelo para o DB2:

- operações de recuperação de valores dos atributos – respectivo método *Observer*;
- operações para atualização de valores de atributos e relacionamentos normais – respectivo método *Mutator*;
- operações com retorno *set* – geralmente, os valores de retorno são resultados de uma consulta SQL. Desse modo, essas operações podem retornar os valores direto da consulta, sem armazenar em alguma estrutura;
- classes temporais versionadas – devem ser definidos gatilhos para as operações a fim de manter o gerenciamento dos controles das versões;
- atributos e relacionamentos temporais – devem ser definidos gatilhos associados às três operações de manipulação (inserir, atualizar e excluir) para realizar os controles sobre os rótulos temporais de cada elemento temporal.

A classe *Object* é mapeada para um tipo estruturado não instanciável (*not instantiable*), com o atributo *tvOID*. A classe *TemporalObject* é mapeada para um tipo estruturado não instanciável, contendo o atributo *alive*. A classe *TemporalVersion* é mapeada para um tipo estruturado não instanciável, contendo os atributos definidos no modelo. A classe *VersionedObjectControl* é renomeada para *VersionedObjCtrl* e mapeada para um tipo estruturado contendo todos os atributos definidos no modelo. As informações a respeito dos objetos versionados são armazenadas em uma única tabela chamada *VOC*, criada a partir do tipo estruturado *VersionedObjCtrl* adicionando colunas para os identificadores da entidade e da classe. Desse modo, toda a classe da aplicação temporal versionada contém um atributo para representar o seu relacionamento com essa classe (*refVOC*).

4.2 Modelagem de uma aplicação

Essa parte do estudo de caso consiste da modelagem de uma empresa de criação de *websites*. Além dos *sites* de seus clientes, a empresa mantém as páginas profissionais de seus empregados. Cada *website* é composto de uma ou mais páginas, sendo que uma delas deve ser considerada como página inicial. Por exemplo, o cliente pode querer manter em páginas diferentes seus dados pessoais, suas informações profissionais, uma relação de seus *sites* preferidos, e assim por diante, mas todas referenciadas a partir da mesma página principal.

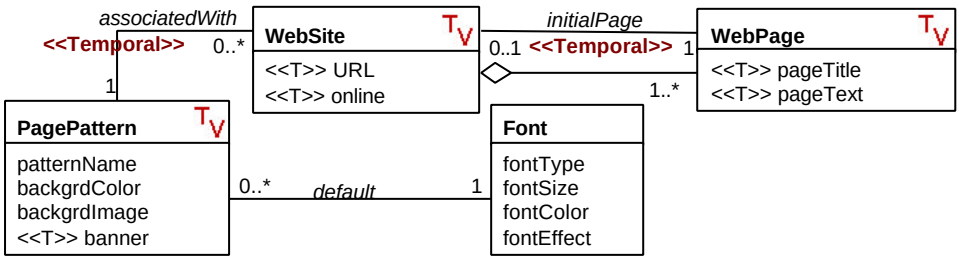

Figura 8. Diagrama de Classes do estudo de caso

```
Class WebSite HasVersions
  aggregate_of n WebPage by reference
    ( Properties:
      temporal URL : string ;
      temporal online : boolean;
    Relationship:
      temporal initialPage 1:1 WebPage;
      temporal associatedWith 1:1 PagePattern;
  ) ;
Class WebPage HasVersions
  ( Properties:
      temporal pageTitle : string;
      temporal pageText : text; ) ;
```

```
Class PagePattern HasVersions
  ( Properties:
      patternName : string;
      backgrdColor : string;
      backgrdImage : string;
      temporal banner : string;
    Relationship:
      default 1:1 Font; ) ;
Class Font
  ( Properties:
      fontType : string;
      fontSize : integer;
      fontColor : string;
      fontEffect : string; ) ;
```

Figura 9. Linguagem de especificação de classes do estudo de caso

Os *websites* também possuem um padrão de página associado composto pela cor e imagem de fundo da página, por um *banner* com propagandas, e pelas especificações *default* da fonte. Esse padrão é usado como controle do *layout* da página de todos os empregados. Segundo especificação da empresa, esse padrão varia conforme as estações do ano e datas comemorativas. Por exemplo, no mês de aniversário da empresa, a imagem de fundo apresenta um bolo com velas acesas, e o *banner* mostra as ofertas especiais do mês (e desse mesmo modo nas épocas de Páscoa, carnaval, datas cívicas, Natal, etc). A Figura 8 apresenta apenas as classes principais modeladas num diagrama de classes, e a Figura 9 apresenta a modelagem na linguagem de especificação de classes do Modelo.

A empresa tem várias possibilidades de usar as informações armazenadas na base de dados construída a partir desse modelo. Uma das mais importantes é a chance de descobrir padrões e perfis dos clientes através de técnicas de mineração de dados (*data mining*). Esse exemplo usa o conceito de versionamento para ilustrar principalmente diferentes versões definidas de acordo com o tempo no qual são válidas. Essa mesma especificação pode ser usada para modelar outros aspectos. Por exemplo, o projetista pode criar diferentes versões de acordo com a tecnologia presente dentro das páginas. Então, pode-se ter versões com *HTML*, *XML*, *java script*, *asp*, *php*, *flash*, etc.

4.3 Representação gráfica das instâncias

A Figura 10 apresenta a classe *PagePattern* com seus objetos versionados *Primavera*, *Verao*, *Outono* e *Inverno*. O objeto versionado *Verao* é apresentado com suas versões: *Verao* (9,15,1) é o padrão de página para o verão; *Natal* (9,15,2) e *NatalB* (9,15,3) foram derivadas como alternativas para o período de Natal; *NovoAno* (9,15,4) foi derivada como versão do novo ano; e *NovoMilenio* (9,15,5) foi derivada da versão de ano novo, como alternativa para o novo milênio. Observa-se que a mudança no valor do atributo *banner* (*natalAd* para *23DezAd*) não cria nova versão, apenas gera mais um valor para o seu histórico.

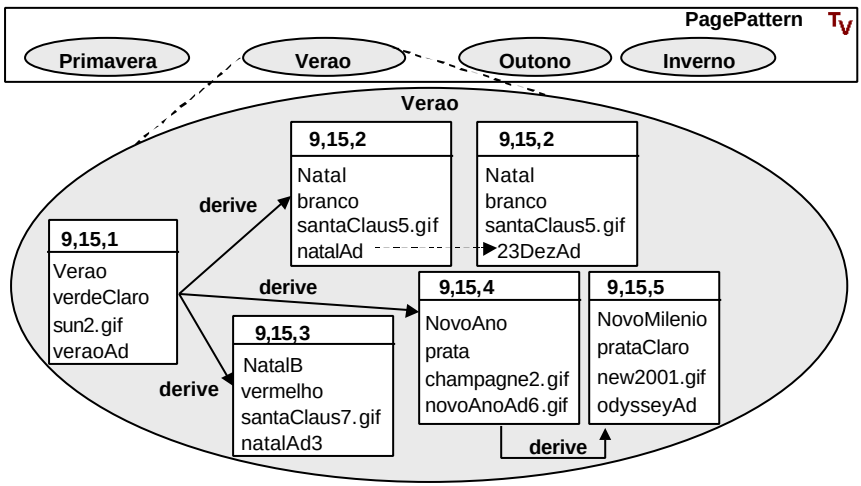

Figura 10. Representação gráfica do objeto versionado Verao

A Figura 11 ilustra a representação gráfica da evolução do relacionamento *associatedWith* (entre as classes *WebSite* e *PagePattern*) para o *website* de um dos empregados. A versão de *WebSite* é sempre a mesma (2,8,1), porém a associação com o padrão de página muda de acordo com a data comemorativa dentro do período de verão. A Tabela 2 apresenta os valores do rótulo temporal para o relacionamento, sendo que toda a variação foi definida pelo usuário no início do mês de dezembro.

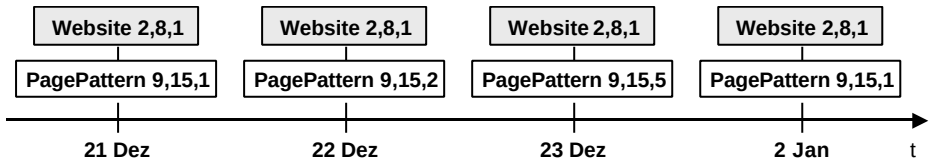

Figura 11. Relacionamento temporal *associatedWith*

Tabela 2. Variação temporal do relacionamento *associatedWith*

PagePattern* no relacionamento *associatedWith	**Tempo validade inicial**	**Tempo validade final**	**Tempo transação inicial**	**Tempo transação final**
9,15,1	21/12/2000	21/12/2000	01/12/2000	null
9,15,2	22/12/2000	25/12/2000	01/12/2000	null
9,15,4	26/12/2000	01/01/2001	01/12/2000	null
9,15,1	02/01/2001	null	01/12/2000	null

5 Conclusão

Esse trabalho propõe o Modelo Temporal de Versões (TVM) para ser usado como alternativa de modelagem de banco de dados. O TVM é ideal para modelar sistemas que evoluem no tempo e que necessitam gerenciar diferentes alternativas na forma de versões. Da união de um modelo de versões com tempo, três vantagens são oferecidas: (i) armazenamento de alternativas de projeto; (ii) armazenamento da história dos dados evolutivos; (iii) possibilidade de reconstruir o estado da base em qualquer data passada, sem usar operações complexas de *backup* e *recovery*. O Modelo apresenta seu diferencial nas características de relacionamento de herança por extensão, ordem temporal ramificada e por permitir classes sem tempo e versões entre as classes temporais versionadas.

O Modelo é definido com riqueza de detalhes que incluem: regras de integridade temporal, exclusão lógica, hierarquia de tipos temporais, diagrama de estado das versões, hierarquia de classes base com seus atributos, relacionamentos, operações e funcionamento, herança por extensão, configuração, relacionamentos entre classes normais e temporais versionadas, linguagem de definição de classes. Devido à limitação de espaço, apenas as principais características estão detalhadas neste trabalho. Também é apresentado um resumo do estudo que compara o TVM com outros dois modelos existentes.

É proposta a arquitetura de uma camada intermediária para o funcionamento do TVM sobre um SGBD convencional. Essa camada é responsável pelo gerenciamento das características específicas do Modelo. Um ambiente (Ambiente Temporal de Versões) é projetado para encapsular a camada e o SGBD, tornando-os transparentes ao usuário. Entre as demais funcionalidades do Ambiente, esse trabalho detalha a Ferramenta de Apoio à Especificação de classes. Através de sua interface, o usuário especifica classes, atributos, relacionamentos e operações sem ser necessário o conhecimento prévio da linguagem de definição do TVM. A partir da especificação, a ferramenta pode gerar o script para criação na base de dados em SQL.

Mesmo com a especificação do Modelo, algumas questões ou sugestões de aprimoramento continuam em aberto: construção de novos tipos literais e estruturados a partir da linguagem de definição, linguagem de manipulação de dados, regras para garantir as propriedades ACID (atomicidade, consistência, isolamento, durabilidade), extensões possíveis da ferramenta de apoio à especificação de classes, novas possibilidades de relacionamento temporal entre classes normais e temporais versionadas, operações para gerar configurações e para restaurar objetos logicamente excluídos, especificação de uma operação para criar uma nova entidade baseada em uma já existente, entre outras.

Outros trabalhos estão em desenvolvimento com base no Modelo Temporal de Versões, entre eles: extensão para o padrão ODMG [9],[10], especificação do Modelo e da linguagem de consulta em XML [12],[24], extensão do Modelo para o suporte à evolução de esquemas [8],[23], extensão e mapeamento da linguagem de consulta para SQL [16],[21], e implementação de um *extender* para o DB2 com as características do TVM.

Referências Bibliográficas

[1] Berliner, B., "CVS II: Parallelizing Software Development", *Procs. the USENIX Winter 1990 Technical Conference*, USENIX Association, Berkeley, pp. 341-352, 1990.

[2] Biliris, A., "Modeling Design Object Relationships in PEGASUS", *Procs. Intl. Conf. Data Engineering* - ICDE, USA, IEEE Computer Society, pp. 228-236, 1990.

[3] Chien, S-Y., Tsotras, V.J., Zaniolo C., "Efficient Management of Multiversion Documents by Object Referencing", *Procs. Intl. Conf. Very Large Data Bases* - VLDB, Italy, pp. 291-300, 2001.

[4] Conradi, R., Westfechtel, B., "Version Models for Software Configuration Management", *ACM Computing Surveys*, vol. 30, pp.232-282, no. 2, June 1998.

[5] Edelweiss, N., Hübler, P., Moro, M.M., Demartini, G., "A Temporal Database Management System Implemented on Top of a Conventional Database", *Procs. Intl. Conf. Chilean Computer Science Society*, Chile, pp. 58-67, 2000.

[6] Estublier, J,. "Software Configuration Management: A Roadmap", *Procs. Intl. Conf. Software Engineering - Future of SE Track*, Ireland, pp.279-289, 2000.

[7] Etzion, O., Jajodia, S., Sripada, E., *Temporal Databases: Research and Practice*, Springer-Verlag, Lecture Notes in Computer Science (LNCS), vol. 1300, 1998.

[8] Galante, R.M., Roma, A.B.S., Jantsch, A., Edelweiss, N., Santos, C.S., "Dynamic Schema Evolution Management using Version in Temporal Object-Oriented Databases", *Intl. Conf. Database and Expert Systems Applications* – DEXA, France, 2002. A ser publicado.

[9] Gelatti, P.C., *Extensão do Padrão ODMG para Suportar Tempo e Versões*. Dissertação de Mestrado, Porto Alegre: PPGC UFRGS, 2002.

[10] Gelatti, P.C., Santos, C.S. dos, Edelweiss, N. "Uma Extensão do Padrão ODMG para Suportar Tempo e Versões", *Simpósio Brasileiro de Banco de Dados*, Gramado - RS, 2002.

[11] Golendziner, L.G., Santos, C.S. dos, "Versions and configurations in object-oriented database systems: a uniform treatment", *Procs. Intl. Conf. Management of Data*, India, pp. 18-37, 1995.

[12] Gomes, C.H.P., *Extensão de uma Linguagem de Consulta para Documentos XML com Características de Tempo e de Versão*. Dissertação de Mestrado, Porto Alegre: PPGC UFRGS, 2002.

[13] Hunt, J.J, Reuter, J., "Using the Web for Document Versioning: An Implementation Report for DeltaV", *Procs. Intl. Conf. Software Engineering*, Canadá, pp. 507-513.

[14] Moro, M.M. *Modelo Temporal de Versões*. Dissertação de Mestrado, Porto Alegre: PPGC UFRGS, 2001.

[15] Moro, M.M., Edelweiss, N., Santos, C. S. dos, "Temporal Versions Model", XV Workshop de Teses e Dissertações - CTD 2002, *Anais do XXII Congresso da Sociedade Brasileira de Computação* - SBC, Brasil, v. 3, pp.33-37, 2002.

[16] Moro, M.M., Gelatti, P.C., Gomes, C.H.P., Rossetti, L.L.F., Zaupa, A.P., Edelweiss, N., Santos, C.S. dos, *Linguagem de Consultas para o Modelo Temporal de Versões*, Relatório de Pesquisa, R.P. 308, Porto Alegre: UFRGS, 2001.

[17] Moro, M.M., Saggiorato, S.M., Edelweiss, N., Santos, C.S. dos, "Adding Time to an Object-Oriented Versions Model", *Procs. Intl. Conf. Database and Expert Systems Applications* - DEXA, Germany, LNCS vol. 2113, pp. 805-814, 2001.

[18] Moro, M.M., Saggiorato, S.M., Edelweiss, N., Santos, C.S. dos, "Dynamic Systems Specification using Versions and Time", *Procs. Intl. Database Engineering and Applications Symposium* - IDEAS, France, IEEE Press, pp. 99-107, 2001.

[19] Moro, M.M., Saggiorato, S.M., Edelweiss, N., Santos, C.S. dos, "A Temporal Versions Model for Time-Evolving Systems Specification", *Procs. Intl. Conf. Software Engineering & Knowledge Engineering* - SEKE, Argentina, pp.252-259, 2001.

[20] Moro, M.M., Saggiorato, S.M., Edelweiss, N., Santos, C.S. dos, "Um Modelo Temporal de Versões para Especificação de Aplicações", *Iberoamerican Work. on Requirements Engineering and Software Environments*, Costa Rica, pp.336-347, 2001.

[21] Moro, M.M., Zaupa, A.P., Edelweiss, N., Santos, C.S dos, "TVQL – Temporal Versioned Query Language", *Intl. Conf. Database and Expert Systems Applications* - DEXA, France, 2002.

[22] Rodríguez, L., Ogata, H., Yano, Y., "TVOO: A Temporal Versioned Object-Oriented data model", *Information Sciences*, Elsevier, vol. 114, pp. 281-300, 1999.

[23] Roma, A.B.S., Jantsch, A., Galante, R.M., Edelweiss, N., Santos, C.S., "Gerenciamento Temporal de Versões para Evolução de Esquemas em BDOO", *Anais do Simpósio Brasileiro de Banco de Dados*, Rio de Janeiro - RJ, pp.110-124, 2001.

[24] Rossetti, L.L.F., *Um Meta Esquema para Especificação do Modelo Temporal de Versões em XML*, Dissertação de Mestrado, Porto Alegre: PPGC UFRGS, 2002.

[25] Snodgrass, R.T. *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufmann Publishers, 2000.

[26] Tansel, C.G. et al. *Temporal Databases - Theory, Design and Implementation*, Benjamin/Cummings, 1993.

[27] Tichy, W.F. "RCS – A System for Version Control", *Software - Practice and Experience*, vol. 5, 1985, no. 7, pp. 637-654, July 1985.

[28] Tichy, W.F., *RCE Homepage*, disponível em http://www.ipd.uka.de/~RCE/.

[29] Whitehead, J., Meredith, W. "WEBDAV: IETF Standard for Collaborative Authoring on the Web", *IEEE Internet Computing*, September/October 1998, pp. 34-40.

[30] Wuu, G.T.J, Dayal, U. "A Uniform Model for Temporal and Versioned Object-Oriented Databases", in *Temporal Databases: Theory, Design, and Implementation*, A.Tansel et al (ed), Benjamin/Cummings, pp. 230-247, 1993.