Implementando uma Camada de Persistência Semidinâmica

José Roberto Motta Garcia
Mestrando
Programa de Graduação da CAP / INPE / SJC
garcia@cptec.inpe.br

Dr. Nilson Sant'Anna
Orientador
Programa de Graduação da CAP / INPE / SJC
nilson@lac.inpe.br

Resumo

Este trabalho visa descrever e implementar uma camada de persistência semidinâmica, para atuar entre uma aplicação desenvolvida com base em Programação Orientada ao Objeto e um banco de dados relacional, de modo que, os objetos da aplicação possam ser armazenados e recuperados do banco de dados, sem que o desenvolvedor necessite se preocupar com a codificação da sua persistência. Esta camada será desenvolvida com base na linguagem Java e fará uso de um mapeamento objeto-relacional semidinâmico para associar as informações da aplicação e do banco de dados.

Abstract

The goal of this paper is to describe and implement a semidynamic persistent layer to work between an Object-Oriented Application and a Relational Database, in order to store and retrieve the application´s objects to and from the database without developer´s need of coding it´s persistence. This layer will be developed based on Java and will use a semidynamic object-relational mapping to associate information between the application and the database.

1. Introdução

Nos dias atuais, muitas empresas deixam o sucesso da empresa ou de determinado produto ou serviço a cargo de seus programas de computador [26]. Com os complexos escopos das aplicações, se torna necessário que o desenvolvedor se especialize em determinados domínios de uma organização, não sendo conveniente que sua esfera de atuação seja perturbada.

O número de aplicações combinando Programação Orientada ao Objeto (POO) [5][7][21] e Bancos de Dados Relacionais (BDR) tem crescido muito por vários motivos:

- Orientação ao Objeto (OO) oferece potencial efetivo para solucionar problemas da tecnologia da informação [6][34].
- Amadurecimento dos Sistemas Gerenciadores de Banco de Dados Orientados ao Objeto (SGBDOO) [11][8][29][33] não tem acontecido nas mesmas proporções que os bancos relacionais [29].
- Modelo Relacional (MR) [29] continua sendo a arquitetura mais confiável e madura quando se trata de armazenamento de dados.

Possuir a habilidade de utilizar O.O. e BDR juntos, torna-se primordial para preservar investimentos em sistemas de armazenamento de informações e, ao mesmo tempo, adequar as aplicações da organização aos complexos domínios cada vez mais presentes.

A O.O. e o M.R. são paradigmas de programação diferentes. Quando os objetos precisam ser armazenados em bancos de dados relacionais, o vácuo entre os dois aspectos precisa ser preenchido.

Desenvolvedores costumam perder muito tempo criando maneiras de tornar os objetos [5] das aplicações persistidos no banco de dados. Objetos consistem em dados e comportamentos e, freqüentemente, possuem herança enquanto bancos de dados relacionais são compostos de tabelas e relacionamentos.

Para corrigir este problema é preciso construir um circuito (o objetivo deste trabalho) que se coloque entre eles para que possam se comunicar.

Este trabalho visa descrever e implementar uma camada de persistência [4][17][19][23][27][28][36] de dados entre estas duas tecnologias, para atuar entre uma aplicação orientada ao objeto e um banco de dados relacional, de modo que os objetos da aplicação possam ser armazenados e recuperados de um banco que segue o modelo relacional sem que o desenvolvedor necessite se preocupar com a implementação da persistência.

Esta camada terá um Mapeamento Objeto-Relacional [2][18][19][23][27][36] semidinâmico, pois será formada por uma estrutura estática de associações, além de um mecanismo ativo de consulta ao Dicionário de Dados (DD) [29], que será responsável por delinear o estado do esquema do banco de dados [29] em tempo de execução.

O núcleo desta camada será desenvolvido com base na linguagem Java™ [14][28][31] e, sua estrutura, será modelada utilizando *Unified Modeling Language* (UML™) [5][25], que é uma linguagem generalizada para projetar aplicações orientadas ao objeto. O mapeamento objeto-relacional terá sua estrutura estática implementada em *Extensible Markup Language* (XML) [30] e seu mecanismo ativo implementado através de instruções *Structured Query Language* (SQL) que acessarão o Dicionário de Dados via *Java Database Connectivity* (JDBC™) [16].

2. Camada de persistência

A camada de persistência isola o desenvolvedor dos detalhes de implementação da persistência de dados e o protege das alterações do esquema. Ler e escrever objetos no banco de dados requer operações básicas de criar, ler, atualizar e apagar. Embora cada objeto da aplicação possa ter sua própria interface de acesso ao banco, o sistema teria mais usabilidade e manutenabilidade se houvesse um conjunto único de operações que todos os objetos pudessem usar.

Essas operações serão sintetizadas como simples instruções, uma para cada operação. Exemplo:

Ao invés da seguinte instrução SQL:

```
sql = "insert into " + DEFAULT_TABLE + "
(CDC_ID, CDC_NAME, CDC_NOTE,
KRN_DICAPA_KDC_ID) VALUES(?,?,?,?)";
```

cada objeto terá uma chamada ao seu próprio método:

```
objeto.create();
```

A camada de persistência que será implementada neste trabalho mapeia os objetos aos mecanismos de persistência (banco de dados) de maneira que pequenas alterações no esquema do banco não afetem o código fonte da aplicação e, portanto, não necessite ser recriada.

Vantagem: desenvolvedor alheio ao esquema do BD. Desvantagem: impacto no desempenho.

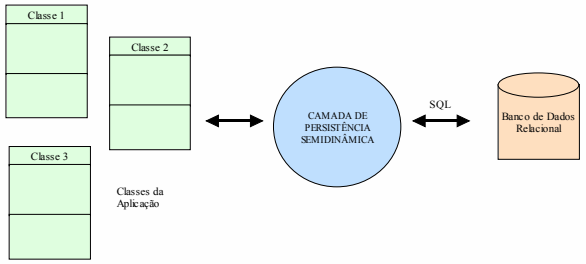

Figura 2.1 – Visão geral da camada de persistência dinâmica

3. Mapeamento objeto-relacional

O paradigma de OO é baseado em construir aplicações a partir de objetos que possuem dados e comportamentos, enquanto que o paradigma relacional é baseado no armazenamento de dados e relacionamentos em tabelas. Esta diferença fundamental resulta numa combinação inadequada entre os dois escopos. Um Mapeamento Objeto-Relacional é uma maneira de associar objetos a tabelas do banco de dados. Uma maneira de implementar um bom mapeamento objeto-relacional é entender bem as duas arquiteturas e suas diferenças e, fazer escolhas corretas baseadas neste conhecimento. Para haver um mapeamento completo é necessário: Mapear os atributos às colunas, as classes às tabelas e os relacionamentos.

3.1 Mapear atributos às colunas

Nem todos os atributos devem ser persistidos, como é o caso de atributos que armazenam totais. Além disso, às vezes, um atributo pode ser mapeado para várias colunas, como é o caso de endereço. Por isso, o atributo de uma classe irá ser mapeado para zero ou mais colunas no banco de dados relacional.

3.2 Mapear classes às tabelas

As classes são mapeadas às tabelas, mas, em geral, não diretamente. Excetuando-se em bancos simples, raramente teremos uma razão de 1:1 ao mapear classes às tabelas. Isto ocorre por causa das características distintas em cada paradigma. A herança é um conceito inexistente no modelo relacional e por isso requer um cuidado especial de implementação.

3.3 Mapear relacionamentos

Haverá a necessidade de mapear também os relacionamentos entre os objetos, de modo que seja permitido recuperá-los. Há dois tipos de relacionamentos que devem ser levados em consideração: a agregação e a associação [29].

O Mapeamento Objeto-Relacional será descrito num arquivo XML em que o Java possui facilidades de tratamento [32]. Um trecho de um arquivo exemplo é mostrado abaixo:

```
<class-name>teste.Manager</class-name>
<table-name>manager</table-name>
<database-name>Database</database-name>
<attribute>
        <attribute-name>id</attribute-name>
        <column-name>id</column-name>
        <key>foreign</key>
        <reference>id</reference>
</attribute>
```

4. Por que uma camada semidinâmica?

Como foi mencionado na introdução deste trabalho, haverá uma estrutura estática, que é o arquivo de mapeamento em XML, mostrado no capítulo anterior e um mecanismo dinâmico para recuperar os metadados (definição ou descrição dos dados), responsável pela parte ativa da camada de persistência.

Analisando melhor o arquivo XML mostrado, pode-se notar claramente os relacionamentos entre os atributos e as colunas (esta é uma das vantagens de se usar XML, pois as marcações são construídas de acordo com as necessidades).

Mas e se surgir necessidade de manipular o tipo do atributo ou até mesmo seu tamanho para tomar certas decisões no domínio da aplicação? Há algumas hipóteses de resolução do mapeamento que devem ser analisadas:

- Basear mapeamento somente no Dicionário de Dados: Exige que os nomes dos atributos sejam exatamente idênticos aos nomes das colunas. Pois não teríamos como fazer a associação entre eles. Não mapeia determinados casos em que há necessidade de conversão de valores, e.g. converter "S" para *True*.
- Basear mapeamento somente no arquivo XML: Permite que os nomes dos atributos sejam independentes dos nomes das colunas. Mapeia casos em que há necessidade de conversão de valores. Necessita alteração no arquivo XML em qualquer situação em que o esquema do banco de dados seja alterado.
- Basear em mapeamento híbrido (XML + DD): Permite que os nomes dos atributos sejam independentes dos nomes das colunas. Mapeia casos em que há necessidade de conversão de valores. Somente alterações de impacto forçam a alteração do arquivo XML.

Este trabalho aplicará a última resolução, acima descrita e para isso será implementado da seguinte maneira:

- Uma classe única de mapeamento será criada para todo o sistema, vista por todos os usuários. Para isso será empregado o Padrão de Desenvolvimento *Singleton* [13].
- Esta classe ficará ativa no Servidor de Aplicação (servidor que roda a aplicação) que permitirá com que todos os clientes tenham acesso [1][9][10].
- Haverá um mecanismo que controlará a necessidade de atualização das informações do dicionário de dados. Este mecanismo deverá atualizar a estrutura ativa do mapeamento à medida que uma atualização no esquema seja realizada.

5. Arquiteturas suportadas

Há muitas arquiteturas atuantes no mercado e, além disso, muitas organizações estão convertendo suas arquiteturas para a que melhor se enquadra no seu perfil [1][9][10][12] e a camada de persistência, por princípio [4], deve atuar dentro de todas elas.

As figuras a seguir mostram a localização da camada de persistência nas diferentes arquiteturas existentes:

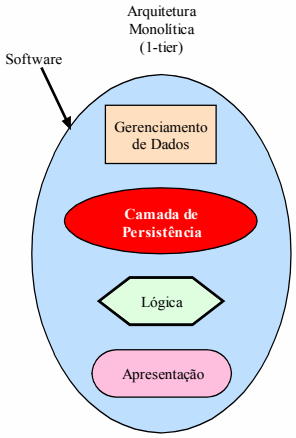

Figura 5.1 – Arquitetura monolítica

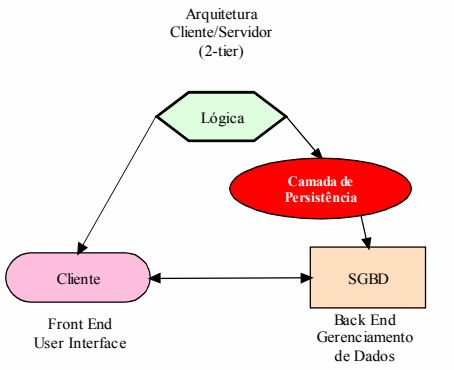

Figura 5.2 – Arquitetura cliente/servidor

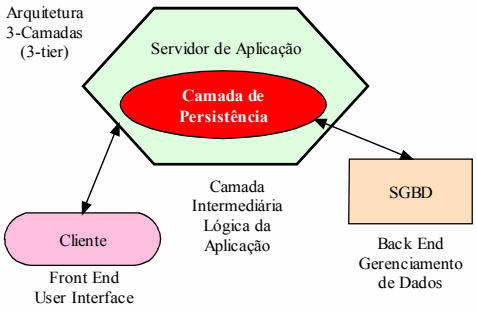

Figura 5.3 – Arquitetura 3-camadas

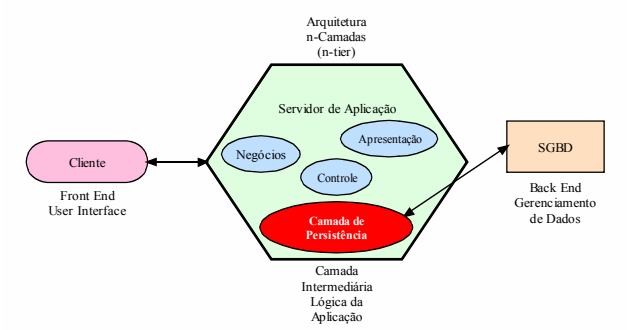

Figura 5.4 – Arquitetura n-camadas

6. Resultados esperados

Como este trabalho espera-se descrever e implementar uma camada de persistência semidinâmica, atuante em diversas arquiteturas de aplicação, que elimine a necessidade do analista de negócios se preocupar com detalhes de implementação dos mecanismos de armazenamento e recuperação das informações do banco de dados.

7. Referências bibliográficas

[1] Amaraju, S.; Buest, C.; et al.. Professional Java Server Programming J2EE – 1.3 Edition. Wrox Press, 2001.

[2] Ambler, S. W. Mapping Objects to Relational Databases. [online]. <URL: http://www.AmbySoft.com>. set. 2002.

[3] Ambler, S. W. Object Concurrency Control. [online]. <URL: http://www.sdmagazine.com>. set. 2002.

[4] Ambler, S. W. The Design of a Robust Persistent Layer for Relational Databases. [online]. <URL: http://www.AmbySoft.com>. set. 2002.

[5] Ambler, S. W. The Object Primer – The Application Developer´s Guide to Object Orientation and the UML. Cambridge University Press, 2001, Second Edition.

[6] Biddle, R.; Tempero, E.; Andreae, P. Object-oriented Programming and Reusability. [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[7] Borba, P. Where are the laws for Object-Oriented Programming? [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[8] Chaudhri, A. B. Object Database Management Systems: An Overview. [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[9] Conallen, J. Building Web Applications with UML. Addison-Wesley, 1999.

[10] Deitel, H. M.; Deitel, P.J.; Santry, S.E. Advanced java 2 Platform: How to Program. Prentice-Hall, 2001.

[11] Dobrovnik, M.; Eder, J. View Concepts for Object-Oriented Databases. [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[12] Fong, J.; Hui, R. Application of Middleware in the Three Tier Client/Server Database Design Methodology. Journal of the Brazilian Computer Society, v. 6, n. 1, p. 50-65, Jul. 1999.

[13] Gamma, E.; Helm, R.; et al.. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[14] Gosling. J.; McGilton, H. The Java Language Environment – A White Paper. [online]. <URL: http://java.sun.com>. set. 2002.

[15] Hofmann, C.; et al. The Fields of Software Architecture [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[16] JDBC API Specification [online]. <URL: http://www.java.sun.com>. set. 2002.

[17] Joukhadar, K. Object-To-Relational Database-Mapping Tool Optimizes Java Persistence. [online]. <URL: http://www.informationweek.com>. set. 2002.

[18] Keller, W. Mapping Objects to Tables – A Pattern Language. [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[19] Keller, W. Objects/Relational Access Layers – A Roadmap, Missing Links and More Patterns. [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[20] Keller, W. Three Layer Architecture. [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[21] Kiczales, G.; Lamping J.; Mendhekar A.; et al. Aspect-Oriented Programming. [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[22] Kramer, D. The Java Platform – A White Paper. [online]. <URL: http://java.sun.com>. set. 2002.

[23] Massoni, T.; Alves, V.; Soares, S.; Borba, P. PDC – Persistent Data Collections Patterns [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[24] McLaugblin, B. Java and XML – Solutions to Real Worls Problems. O'Reilly, 2001.

[25] OMG Unified Modelling Language Specification. [online]. <URL: http://www.omg.org/uml>. set. 2002.

[26] Pressman, R. S. Software Engineering: A Practitioner´s Approach. McGraw Hill, 1987, Second Edition.

[27] Remanathan, C. Providing Object-Oriented Access to Existing Relational Databases. Mississipi. 118p. Dissertation (Doctor of Philosophy in Computer Science) – Mississipi State University, 1997.

[28] Saljoughy, A. Object Persistence and Java. [online]. <URL: http://www.javaworld.com>.

[29] Silberschatz, A.; Korth, H. F.; Sudarshan, S. Sistema de Banco de Dados. Makron Books,1999, Terceira Edição.

[30] Sun Microsystems, Inc. A Quick introduction to XML. [online]. <URL: http://www.java.sun.com>. set. 2002.

[31] Sun Microsystems, Inc. The Java Tutorial – A Practical Guide for Programmers. [online]. <URL: http://java.sun.com>. set. 2002.

[32] Sun Microsystems, Inc. The Java Web Services Tutorial. [online]. <URL: http://www.java.sun.com>. set. 2002.

[33] Tari, Z.; Craske, G.; Bukhres, O. Teaching Object-Oriented Database Concepts. [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[34] Vermeer, M. W. W.; Apers, P. M. G. Object-oriented views of relational databases incorporating behaviour. [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.

[35] Victoriano, B. A. D.; Garcia, C.C. Produzindo Monografia. Publisher Brazil, 1996, Segunda Edição.

[36] Yoder, J. W.; Johnson, R. E.; Wilson, Q. D. Connecting Business Objects to Relational Databases. [online]. <URL: http://citeseer.nj.nec.com/cs>. set. 2002.