MOR: Uma Ferramenta para o Mapeamento Objeto-Relacional em Java

Leonardo Gresta Paulino Murta
Gustavo Olanda Veronese
Cláudia Maria Lima Werner
{murta, veronese, werner}@cos.ufrj.br
COPPE/UFRJ – Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Caixa Postal 68511 – CEP. 21945-970
Rio de Janeiro – Brasil

Abstract

This work presents an approach to map Java objects into relational databases. This approach is transparent because does not require any extra code. It uses the reflection mechanism to know what information exists inside the objects and the Java Database Connection (JDBC) to connect with the different databases.

1 Introdução

Na última década as linguagens de programação orientadas a objetos foram largamente difundidas, sendo grande a quantidade de sistemas que têm sido desenvolvidos em linguagens que seguem este paradigma. Em paralelo, os sistemas gerenciadores de bancos de dados orientados a objetos surgiram, porém com pouca força em aplicações convencionais, seja pelo fato de que os bancos relacionais já estão consolidados há tempos, dado a diversidade de opções no mercado, ou pelo aproveitamento das grandes bases de dados constituídas pelas organizações durante anos utilizando modelos relacionais.

O processo de armazenamento, suportado pelos sistemas de gerenciamento de banco de dados orientados a objetos (SGBD-OO), segue de maneira transparente, sem que sejam necessárias conversões. Porém, a popularização desses sistemas é pouca, devido à falta de robustez e desempenho, em comparação com os SGBDs relacionais existentes.

É necessário, então, dispor de mecanismos que mapeiem objetos em memória para meios persistentes relacionais e que forneçam transparência ao usuário. Este processo não é imediato e as linguagens OO não fornecem atualmente suporte direto para o mesmo.

O objetivo principal deste trabalho é prover uma abordagem para o armazenamento de objetos Java em bancos de dados relacionais, através de um processo transparente e de fácil utilização.

Para fornecer transparência ao processo de armazenamento, foi utilizado o mecanismo de reflexão da linguagem, o que torna desnecessário qualquer entendimento por parte do desenvolvedor do mecanismo de mapeamento de tipos Java – SGBD. Também foi utilizada a ponte JDBC-ODBC, o que permite acesso a qualquer SGBD que forneça suporte a ODBC.

Este trabalho está organizado em 5 seções. Esta seção exibe a motivação e os objetivos desse trabalho. A Seção 2 fornece uma descrição de outras abordagens para solucionar esse problema. A Seção 3 descreve o contexto em que foi proposto esse trabalho. A Seção 4 detalha a arquitetura da solução proposta e, finalmente, a Seção 5 destaca as contribuições, limitações e trabalhos futuros.

2 Abordagens para o Armazenamento de Objetos

O mapeamento objeto relacional pode ser feito tanto manualmente quanto automaticamente. O processo de mapeamento objeto relacional manual é trabalhoso, pois

torna-se necessário definir, para cada classe do sistema, como esta será armazenada. Já o processo automático, suportado por uma ferramenta de mapeamento, torna esse processo mais simples, mas nem sempre o torna transparente.

A linguagem Java fornece o mecanismo de serialização, que pode ser utilizado para o armazenamento de objetos. Dada uma raiz de persistência (um objeto), todos os objetos referenciáveis a partir da raiz serão armazenados em um arquivo do sistema operacional. A serialização é fácil de ser usada, porém apresenta alguns pontos indesejáveis a um mecanismo de persistência. Um deles é a falta de escalabilidade, pois quando se trabalha com grande quantidade de dados, todos os dados são armazenados em um mesmo arquivo. Outro ponto é a falta de robustez, pois quando se trabalha com versões de aplicações diferentes, o arquivo produzido por uma versão não poderá ser lido pela outra. Um terceiro ponto a ser levantado é a falta de transparência em relação à estrutura interna utilizada. Não se conhece, a priori, como os objetos são armazenados, o que torna complexa uma modificação direta na base. Em último lugar, este mecanismo permite apenas que se armazene e recupere dados de uma só vez, o que pode ser bastante custoso, ou mesmo impraticável, para determinados tipos de aplicações.

Existe também uma grande variedade de ferramentas comerciais que se propõem a resolver esse problema (CETUS LINKS, 2001), entre elas, as mais conhecidas são: BoldSoft (BOLDSOFT, 2001), TopLink(THE OBJECT PEOPLE, 2001), CocoBase (THOUGHT, 2001), Java Blend (SUN MICROSYSTEMS, 2001), etc. Contudo, devido ao custo elevado, essas soluções se tornam inviáveis para sistemas de pequeno porte.

3 Contexto da Abordagem Proposta

O MOR foi desenvolvido para dar mais uma alternativa de persistência à Infra-estrutura Odyssey (WERNER et al., 2000). A Infra-estrutura Odyssey fornece suporte à reutilização de software, através da engenharia de domínio (BRAGA et al., 1999) e da engenharia de aplicações (MILER, 2000), através de uma extensão da UML.

A infra-estrutura Odyssey disponibiliza três formas de persistência das suas informações: serialização, GOA++ (MATTOSO et al., 2000) e MOR. A Figura 1 exibe o relacionamento entre a infra-estrutura e suas formas de armazenamento.

Desta forma, a Infra-estrutura Odyssey passa a ter mais uma opção para o armazenamento de objetos, além da serialização e do GOA++, que permite o armazenamento em bases relacionais, provendo um mapeamento entre o modelo de objetos e o modelo relacional através do mecanismo de reflexão de Java (um *framework* desenvolvido pela SUN, que permite a introspecção e instanciação de objetos em tempo de execução).

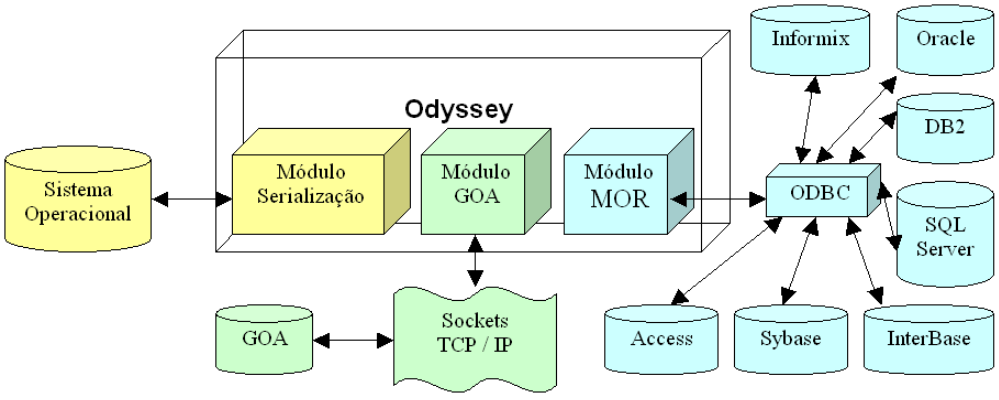

Figura 1: Formas de armazenamento do Odyssey.

O MOR realiza o acesso ao banco de dados relacional através da ponte JDBC (Java DataBase Connectivity), uma API da linguagem Java que permite que o acesso seja independente da plataforma. Qualquer banco que permita acesso via ODBC pode ser utilizado, pois a máquina virtual JAVA fornece o driver para conexão ODBC. A Tabela 1 exibe uma comparação entre as formas de armazenamento da Infra-estrutura Odyssey.

Forma de Armazenamento	Nível de Complexidade de Utilização	Nível de Tecnologias Relacionadas	Nível de Confiabilidade	Propósito Principal
Serialização	baixo	baixo	Baixo	Testes e pequenos modelos. É indicado para o uso em ambientes que não tem acesso a um SGBD via ODBC.
GOA	alto	alto	Médio	Pesquisa e desenvolvimento de tecnologias para armazenamento de objetos. Deve ser utilizado com cautela em produção pois ainda está em fase de desenvolvimento.
MOR	médio	médio	alto	Utilização em produção com grandes modelos. Deve ser utilizado como forma padrão de armazenamento do Odyssey.

Tabela 1: Formas de armazenamento de objetos da Infra-estrutura Odyssey.

4 Descrição da Arquitetura

O MOR provê um mapeamento dos objetos gerenciados em memória pela aplicação a qual ele esteja conectado, em tabelas a serem manipuladas por um banco relacional. Cada classe a ser armazenada é tratada no banco como uma tabela. Cada objeto de uma determinada classe é mapeado em uma tupla da tabela que representa sua classe. Os atributos da classe são as colunas da tabela correspondente. Além dos atributos já existentes, outros dois são adicionados a cada tabela para que o MOR possa hierarquizar a persistência. Esses atributos são:

- **this:** armazena o ID do objeto (gerado automaticamente pelo MOR);
- **super:** armazena o ID do objeto que representa a classe pai na estrutura de herança (ou NULL caso não exista herança). A tupla apontada pelo atributo super reside em uma tabela que contém informações sobre a classe herdada (Figura 2).

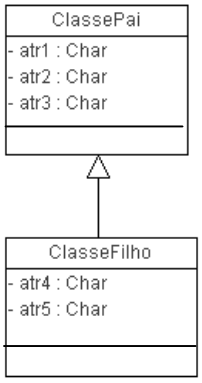

Tabela "pacote_ClassePai"

this	super	atr1	atr2	art3
10	NULL	a	b	c

Tabela "pacote_ClasseFilho"

this	super	atr4	atr5
35	10	d	e

Figura 2: Exemplo de mapeamento de herança.

As tabelas que representam estas classes possuem o nome na forma pacote_subpacote_ …_NomeDaClasse, ou seja, o nome da tabela é representado pelo nome completo da classe, com o divisor '.' substituido por underscore ("_").

Considere que um objeto da classe ClasseFilho tenha sido instanciado com os seguintes valores de atributos: atr1 = ‘a’, atr2 = ‘b’, atr3 = ‘c’, atr4 = ‘d’, atr5 = ‘e’. A Figura 2 apresenta as tabelas criadas pelo MOR. Uma tupla na tabela pacote_ClassePai, que representa a classe pai, será criada com os valores dos atributos atr1, atr2 e atr3. O campo **this** conterá um número identificador gerado pelo MOR. O campo **super** será preenchido com o valor NULL, já que esta classe herda da classe padrão Object. Na tabela pacote_ClasseFilho, que representa a classe filha, será criada uma tupla com os valores dos atributos atr4 e atr5. Ou seja, estes dois atributos são atributos não herdados. O campo **super** será então preenchido com o ID (**this**) da tupla que representa o objeto da classe pai.

O MOR realiza quatro operações básicas:

- **Remoção de tabelas:** o MOR faz uma consulta ao esquema do banco removendo todas as tabelas do usuário;
- **Criação das tabelas:** são criadas as tabelas necessárias para o mapeamento das classes, obtendo seus atributos primitivos e referências através de reflexão;
- **Armazenamento dos objetos:** é utilizada a reflexão para a obtenção dos atributos primitivos e referências. O MOR varre a raiz de persistência e, recursivamente, armazena os objetos. Tipos primitivos são armazenados diretamente na tabela como atributos primitivos do banco, enquanto que referências são tratadas como chaves estrangeiras de outras tabelas. Caso a classe tenha uma superclasse, esta também é armazenada.
- **Recuperação dos objetos:** é utilizada a reflexão para a atribuição dos atributos primitivos e referências. Os objetos relacionados são recuperados recursivamente do banco de dados. Caso os valores dos atributos não possam ser recuperados, são utilizados os valores pré-definidos no construtor sem parâmetro da classe correspondente. Desta forma, o MOR pode ignorar eventuais erros na obtenção de algum atributo, recuperando a parte do modelo não comprometida, ao contrário do armazenamento por serialização, que ao encontrar um erro não recupera o modelo.

Todas as operações, com exceção da primeira, fazem uso do mecanismo de reflexão implementado na biblioteca “java.lang.reflect” da linguagem Java. A reflexão é uma técnica de algumas linguagens de programação que permite o acesso a dados internos dos objetos e seus relacionamentos em tempo de execução.

Nem todas as classes do sistema podem ser tratadas de uma forma padrão pelo MOR. Alguns casos especiais são levados em conta e, para que se faça o armazenamento completo, são necessárias algumas considerações:

- **Método construtor sem parâmetro:** As classes que serão armazenadas pelo MOR devem possuir um construtor sem parâmetros, ou seja, além dos construtores originais da classe a ser armazenada, um construtor adicional dever ser feito. O construtor sem parâmetro é necessário para que o mecanismo de reflexão possa instanciar o objeto no momento da recuperação. Ele também é útil para definir valores *default* para os atributos (caso esses não possam ser recuperados).
- **MVC Swing:** A biblioteca gráfica swing é baseada em no padrão MVC (Model View Controler) (GAMMA et al., 1995), onde os componentes *model* guardam referências fracas aos componentes *view* (referências necessárias para atualização do view no caso de mudança do *model*). Entretanto, não é desejável que essas referências sejam armazenadas, pois os objetos de interface com o usuário também seriam armazenados. Para contornar esse problema foram criados tratadores para os *models* do swing.
- **Atributos transientes:** Os atributos transientes não serão armazenados no MOR. Este tipo de atributo indica os dados que não deverão ser considerados no

armazenamento. Contudo, alguns componentes fornecidos pela SUN (Hashtable, Vector, etc.) colocam o modificador *transient* em atributos importantes, e que deveriam ser armazenados. Isso ocorre pois esses componentes sobrescrevem o método writeObject e readObject, armazenando os atributos de uma forma própria no caso da serialização. Para contornar esse problema foram criados tratadores para esses componentes.

- **Atributos finais:** A versão atual do MOR não armazena atributos finais, pois os consideramos como constantes. Mas pode acontecer do atributo final não ser inicializado no momento da declaração e nem no construtor sem parâmetro (o que acarretaria em uma falha de armazenamento). Logo, sempre que o MOR for utilizado, é necessário inicializar os atributos finais ou na declaração ou no construtor sem parâmetro.
- **Classes primitivas:** Outro problema relacionado aos atributos finais é o caso das classes primitivas, que guardam os seus valores neste tipo de atributo (String, Integer, Float etc.). Essas classes têm que ser construídas já com os valores corretos (não podem ser atualizados *a posteriori*). Para contornar esse problema foram criados tratadores para essas classes.

Devido a essas considerações, foi criada uma estrutura extensível de tratadores. Para toda classe que não pode ser armazenada automaticamente deve ser construído um tratador, que herda da classe base "Tratador". Um tratador deve informar ao MOR qual será o seu comportamento quando a classe do tipo que o tratador trata estiver sendo armazenada, criada ou recuperada. A Figura 3 exibe o diagrama de classes de alto nível do MOR.

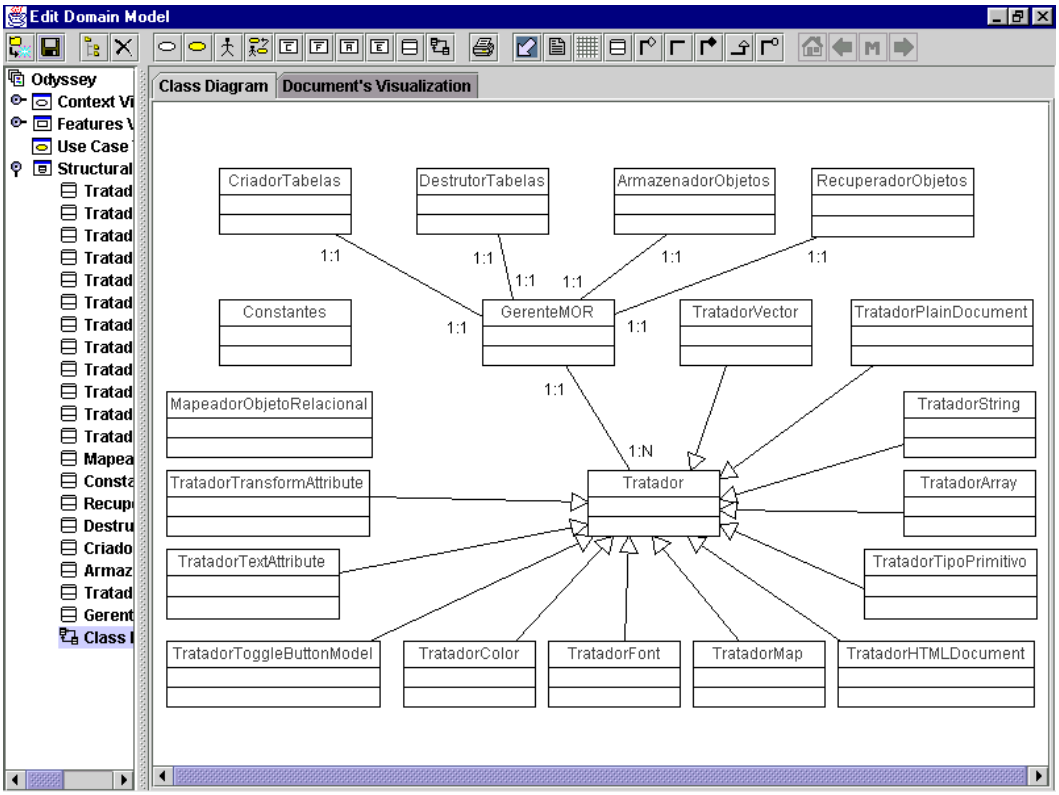

Figura 3: Modelo de classes do MOR dentro do Odyssey.

Ao utilizar a solução disponibilizada pelo MOR não é necessário codificar a parte do mecanismo de armazenamento, bastando uma linha de código para armazenar todos os

objetos que estão em memória e outra linha de código para recuperá-los do banco, como está exibido na Tabela 2.

Armazenamento	GerenteMOR.getInstancia("odyssey").armazena(dados);
Recuperação	dados = (Hashtable)GerenteMOR.getInstancia("odyssey").recupera();

Tabela 2: Código de armazenamento e recuperação de objetos no MOR.

5 Conclusão

Este artigo apresentou uma abordagem para o armazenamento de objetos Java em bancos de dados relacionais. A principal contribuição deste trabalho é prover uma solução transparente e acessível para o problema de mapeamento objeto-relacional.

A cada execução o MOR executa uma carga completa do banco para a memória e, posteriormente, uma reconstrução completa da base. Entretanto, esse procedimento pode se tornar extremamente oneroso para grandes bases de dados.

O MOR utiliza bancos de dados relacionais para armazenar informações, o que impossibilita a execução de OQLs (Object Query Language) sobre a base. As OQLs se diferenciam das SQL por, entre outras coisas, permitirem a chamada de métodos e a utilização de expressões de caminho.

Um trabalho futuro seria a comparação entre as abordagens de serialização, GOA++ e MOR para possibilitar a integração da estrutura do GOA++ ao mecanismo de reflexão do MOR, permitindo com que não seja necessária a reconstrução da base a cada armazenamento de objetos e a carga completa da base para a memória no momento da recuperação dos objetos.

Referências Bibliográficas

BOLDSOFT, 2001, site na Internet em http://www.boldsoft.com/, acessado em 11/05/2001.

BRAGA, R. M. M., WERNER, C. M. L., 1999, "Odyssey-DE: Um Processo para Desenvolvimento de Componentes Reutilizáveis", *X CITS*.

CETUS LINKS, 2001, site na Internet em http://www.cetus-links.org/oo_db_systems_3.html, acessado em 11/05/2001.

GAMMA, E., HELM, R., JOHNSON, R., et al., 1995, "Design Patterns", *Addison Wesley*.

MATTOSO, M., WERNER, C. M. L., BRAGA, R. M. M., et al., 2000, "Persistência de Componentes num Ambiente de Reuso", *XIV Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas*, João Pessoa.

MILER, N., 2000, "A Engenharia de Aplicações no Contexto da Reutilização baseada em Modelos de Domínio", tese de mestrado, *COPPE/UFRJ*.

SUN MICROSYSTEMS, 2001, "Java Blend", site na Internet em http://www.sun.com/software/javablend, acessado em 11/05/2001.

THE OBJECT PEOPLE, 2001, site na Internet em http://www.objectpeople.com/toplink/, acessado em 11/05/2001.

THOUGHT, 2001, site na Internet em http://www.thoughtinc.com/cber_index.html, acessado em 11/05/2001.

WERNER, C. M. L., BRAGA, R. M. M., MATTOSO, M., et al., 2000, "Infra-estrutura Odyssey: estágio atual", *XIV Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas*, João Pessoa, outubro.