Uma arquitetura para implementar aplicações padrão ODMG sobre banco de dados relacional

VITOR VALERIO DE SOUZA CAMPOS[1]
CARLOS ALBERTO HEUSER[2]

[1]UEL – Universidade Estadual de Londrina
Campus Universitário Cx. Postal 6001 CEP 86051-990, Londrina - PR, Brasil
valerio@uel.br

[2]UFRGS – Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves 9500 Instituto de Informática, Porto Alegre, RS Brazil
heuser@inf.ufrgs.br

Resumo. Este trabalho concentra-se na estruturação de uma arquitetura que permita a conexão entre a aplicação desenvolvida em linguagem de programação orientada a objetos (LPOO) segundo o padrão ODMG e o SGBDR. Apresenta, também, um conjunto de regras para, a partir de um esquema relacional existente, obter o modelo OO correspondente.

A característica marcante da arquitetura é que a base de dados relacional subjacente é projetada de acordo com as técnicas clássicas de projeto de base de dados relacional. Diferentemente de outras propostas, este trabalho trata os conceitos de chave primária e chave estrangeira ao invés dos conceitos de projeto de base de dados orientada a objetos (OO), identificadores de objetos e referências entre objetos.

Palavras chaves: Orientação a objetos, Banco de Dados Orientado a Objetos, Persistência, Mapeamento objeto-relacional, ODMG.

1 Introdução

Sistemas gerenciadores de banco de dados relacional (SGBDR) têm uma tecnologia bem conhecida e consolidada no mercado. Muitas aplicações foram desenvolvidas sobre estes sistemas e grandes quantidades de dados encontram-se armazenadas neste paradigma.

Um banco de dados relacional pode ser acessado de múltiplas formas. O acesso pode ser feito através de aplicações desenvolvidas em uma linguagem de programação usando-se SQL embutido. Usuários finais podem acessar a base de dados diretamente para consultas SQL ou utilizar interfaces gráficas que as geram. A base de dados pode ser acessada através da *WEB*, com auxílio de ferramentas para integração *WEB*-banco de dados.

Contudo, o paradigma Orientado a Objeto (OO) está se estabelecendo como modelo de programação de aplicações. Os benefícios deste paradigma são a manutenção, a extensão e a modificação mais simples de aplicações.

Foi definido um padrão de interface de programação OO sobre bases de dados, o padrão definido pelo *Object Data Management Group* (ODMG) [CAT94][CAT97][COO97]. Este padrão é usado por vários produtos comerciais, tais como o POET da POET *Software* [POE96], o O_2 da O_2 *Tecnology* [O2T] e o OpenDM da *Siemens Nixdorf Informations System* [SIE96]. O ODMG é um grupo que atua na padronização de Sistemas Gerenciadores de Banco de Dados Orientados a Objeto (SGBDOO), permitindo aos clientes de SGBDOO escreverem aplicações portáveis, isto é, que possam ser executadas em mais de um produto SGBDOO.

A linguagem de programação orientada a objeto possui características que permitem o desenvolvimento de aplicações complexas, tais como tipos abstratos de dados, complexas restrições de integridade, agregação e herança, que tornam o software facilmente extensível,

reusável e manutenível. Estas características apresentam-se como importantes vantagens para o programador, proporcionando sua rápida penetração no mercado.

Apesar do avanço do paradigma OO, uma faixa extensa do mercado ainda é dominada por SGBDR. Parte significativa dos dados encontra-se armazenada em bancos de dados relacionais, tornando-se importante a construção de ferramentas de conexão que permitam ao programador OO acessar bases de dados relacionais [BAN96].

A característica marcante da arquitetura aqui descrita, que difere de outras propostas existentes na literatura [RAM94][ROU89][TSM94], é que a base de dados relacional subjacente é projetada de acordo com as técnicas clássicas de projeto de base de dados relacional [ELM89]. No projeto clássico de bases de dados relacionais são usados os conceitos de chave primária e chave estrangeira. Já no projeto de bases de dados OO são usados os conceitos de identificadores de objetos e referências entre objetos.

A técnica de mapeamento a ser utilizada obtém as classes do C++ ODL (a linguagem de definição de esquema do padrão ODMG) diretamente do esquema relacional. Estas regras oferecem várias alternativas de mapeamento, sendo requerida interação humana na definição do esquema orientado a objetos.

O trabalho concentra-se na estruturação de uma arquitetura que permita a conexão entre a aplicação desenvolvida em linguagem de programação orientada a objetos (LPOO) e o SGBDR. Assim, a arquitetura possibilita a coexistência de dois diferentes paradigmas: relacional e OO.

A arquitetura a ser construída poderá ser utilizada sobre bases de dados legadas, já existentes em uma organização. Além disso, permitirá que a base de dados seja acessada por ferramentas para SGBD relacional, como WEB browsers ou ferramentas de consultas.

Esta arquitetura não é um produto de software que provê todas as funcionalidades do padrão ODMG como os produtos OpenDM [SIE96] e POET [POE96]. Contudo, permite a usuários que não têm acesso a produtos comerciais, desenvolver uma aplicação Orientada a Objetos e armazenar em um Banco de Dados Relacional existente.

O padrão ODMG pode ser usado com diferentes LPOO, como Java, C++ e *SmallTalk*. Neste trabalho, foi adotada a linguagem C++ .

2 Arquitetura do modelo ORDBMS

Nesta seção, vai-se descrever a visão geral da arquitetura, o detalhamento e o mapeamento adotado para, a partir de um banco de dados relacional, gerar classes em ODL.

2.1 Visão geral da arquitetura

A arquitetura proposta aqui, foi concebida com o objetivo de permitir que um usuário programe utilizando o modelo de objetos do ODMG e armazene seus dados em um banco de dados relacional.

Esta arquitetura separa o tratamento de persistência, que é feito pelo componente mapeamento, do domínio do problema. A separação tem por objetivo permitir uma menor dependência das classes do domínio do problema em relação a mudanças na forma de armazenamento e está organizada em quatro(4) componentes interrelacionados, conforme mostra o esquema na figura 2.1, onde:

No lado esquerdo superior, está representado o componente domínio do problema (DP) – genérico que contêm as classes com capacidade de persistência, transação e banco de dados. Na parte intermediária superior está o componente Mapeamento – genérico que contêm as classes genéricas de mapeamento para o armazenamento de objetos no banco de dados. Estes componentes são divididos em três subcompontes a saber:

- Entidade, Banco de Dados e Transação.
- Relacionamento e Referência.
- Extent e Iterator.

No lado esquerdo inferior, está representado o componente DP – específico que contêm as classes específicas utilizadas na aplicação. Na parte intermediária inferior está representado o componente Mapeamento - específico que contêm as classes específicas de mapeamento para o armazenamento de objetos no banco de dados relacional. Estes componentes são divididos em três subcompontes a saber:

- Entidade.
- Relacionamento e Referência.
- Extent e Iterator.

No lado direito está representada o componente Classes da MFC que contêm as classes da MFC.

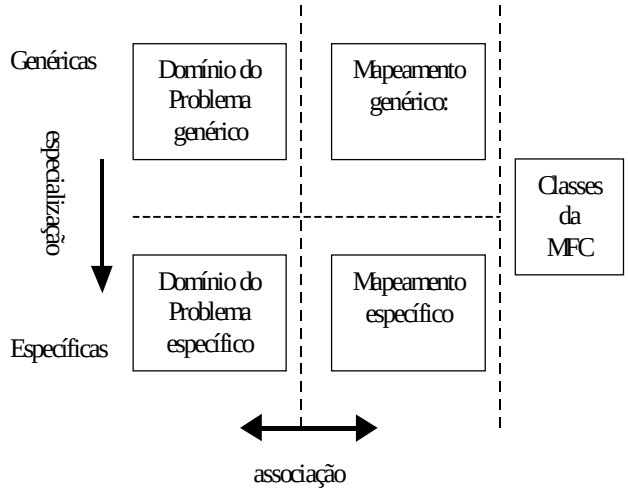

FIGURA 2.1 - Esquema da arquitetura da aplicação

Esta estrutura de trabalho fornece ao programador da aplicação, no componente domínio do problema, um conjunto de classes padrão ODMG, do qual permite derivar classes especializadas. O componente Mapeamento fornece um conjunto de classes que implementam o mapeamento OO-relacional. Estas classes não são utilizadas diretamente, nem referenciadas pelo programador da aplicação.

O componente Domínio do Problema contem classes abstratas de onde derivam todas as classes persistentes. Através da interface fornecida por estas classes, os objetos persistentes podem ser armazenados, recuperados, inseridos e excluídos do banco de dados, bem como são tratados seus relacionamentos. Os métodos destas classes são métodos “template”, que contém o código necessário para a comunicação com as classes associadas no componente Mapeamento.

O componente DP específico é formado pelas classes que modelam o domínio de problema da aplicação. Estas classes são especializações das classes do componente DP genérico.

O componente Mapeamento é formado por classes abstratas genéricas, as quais contém métodos abstratos, servindo apenas para definir a interface das classes Mapeamento específicas da aplicação. Os métodos destas classes são ativados pelos métodos “template” existentes no componente Mapeamento genérico.

No componente Mapeamento-específico, cada classe corresponde a uma construção (tabela, tupla, chave estrangeira) da base de dados relacional. Os métodos destas classes realizam o mapeamento entre os modelos e contém código SQL específico para cada operação a ser realizada sobre a base de dados. As classes e seu métodos podem ser gerados automaticamente a partir do esquema da base de dados a ser usada.

2.2 Classes da Componente domínio do Problema

Esta especificação de projeto trabalha com um subconjunto da interface de programação estabelecida pelo ODMG 2.0. Este conjunto de classes constitui parte do domínio de problema a ser tratado. O projeto através das classes ODMG possibilita a abstração de todo o mecanismo de persistência no qual se utiliza um banco de dados relacional para armazenamento. As classes são: d_Database, d_Object, d_Ref, d_Transaction, d_Rel_Ref, d_Rel_Set, d_Extent, d_Iterator [JOR97][CAT94a].

3 Linguagem de manipulação de objetos (OML)

A Linguagem de Manipulação de Objetos (OML) é usada para criar, remover, identificar, referenciar, incluir valor, excluir valor e invocar operações sobre objetos persistentes. Um dos princípios guias da OML, usados no projeto do padrão ODMG, é que a sintaxe usada para realizar operações sobre um objeto persistente não seja tanto quanto possível diferente daquelas usadas para objetos não persistente [POE 96].

3.1 Criação de objetos

Uma aplicação desenvolvida através da OML pode criar objetos de qualquer tipo definido no esquema ODL ou em adição à definição de tipos C++. Tipos que não são definidos no esquema ODL são transientes, isto é, os objetos são alocados na memória e não no banco de dados [SIE96]. Tipos do esquema ODL podem ser usados tanto para objetos transientes como para objetos persistentes.

Objetos persistentes são criados de acordo com exemplo abaixo. Este especifica que o novo objeto criado será colocado no banco de dados específico.

d_Ref<Order> Ord = new(database, “Order”) Order;

3.2 Remoção de objetos

Objetos, uma vez criados, podem ser removidos usando-se o método d_Ref::delete_object da OML. A remoção de um objeto é permanente e sujeita ao término da transação. O objeto é removido da memória e, sendo ele um objeto persistente, também será removido do banco de dados. A instância d_Ref continuará existindo na memória, mas ela referencia um valor indefinido. Uma tentativa de acessar este objeto de qualquer instância d_Ref falhará e sinalizará um erro [SIE96].

```
d_Ref<Product> Prod1;
...
Prod1.delete_object();
```

3.3 Modificação de objetos

O estado de um objeto é modificado atualizando suas variáveis membro ou invocando operações sobre ele. Atualizações de objetos persistentes são feitas visíveis para outros usuários do banco de dados quando a transação termina.

O fato de que um objeto ter sido ou irá ser modificado durante a transação deverá ser comunicado ao sistema de banco de dados usando-se o método d_Object::mark_modified; este é usado como segue, assumindo-se que obj_ref refere-se para o objeto:

```
obj_ref->mark_modified( );
```

3.4 Acessando e modificando atributos

Atributos podem ser acessados como variáveis membro de C++. Por isso o operador ‘->’ é sobrecarregado. Se ele é usado em expressões comuns, isto é, no lado direito de uma atribuição ou como parâmetro de método, o valor do atributo é lido do banco de dados. Se ele é usado no lado esquerdo de uma atribuição, o valor é armazenado na memória principal e o subseqüente termino da transação escreverá a modificação para o banco de dados [SIE96].

Conforme mostra o exemplo abaixo, o código lê as variáveis membro ‘cod’ de Product1 e atualiza a variável membro number.

```
d_Ref <Product> Product1;
// atributo de leitura
cout <<“Código do Produto” << Product1->cod;
// atualiza atributos
Product1->number++;
```

3.5 Manutenção de relacionamento

Ambos os relacionamentos um-para-um e um-para-muitos são suportados na OML. A integridade do relacionamento é mantida pelo SGBD [CAT97]. Nas seções seguintes será apresentado como estabelecer, modificar e remover relacionamentos para relacionamentos um-para-muitos através de cenário.

3.5.1 Cenário de relacionamento um-para-muitos

O cenário usa o relacionamento entre os objetos definidos na classe Customer e Order. Devido a um lado do relacionamento ser uma coleção de referências, em vez de uma única referência, inserções e remoções de elementos da coleção são realizadas explicitamente pela aplicação ou implicitamente pelo software de banco de dados [JOR97].

Assuma os objetos e os relacionamentos mostrados na figura 3.1. As seguintes variáveis serão inicializadas para referenciar objetos no banco de dados.

```
d_Ref<Custumer>  CustA, CustB
d_Ref<Order> OrdA, OrdB, OrdC, OrdD, OrdE, OrdF
```

Pode-se observar que na figura é apresentado somente os objetos e que na declaração acima são apresentadas somente as referências aos objetos.

3.5.2 Estabelecendo um relacionamento

Suponha que seja estabelecido um relacionamento entre os objetos, OrdF e CustB. A seguinte linha estabelece o relacionamento

```
OrdF->is_req_by = CustB
```

Depois que esta linha é executada a referência is_req_by de OrdF aponta para CustB e também, de forma implícita, o OrdF é adicionado ao conjunto de pedidos no objeto Customer. Produzir-se-ia o mesmo efeito se fosse executado

a declaração abaixo, que adiciona OrdF diretamente ao cliente.

CustB->req.insert_element(OrdF);

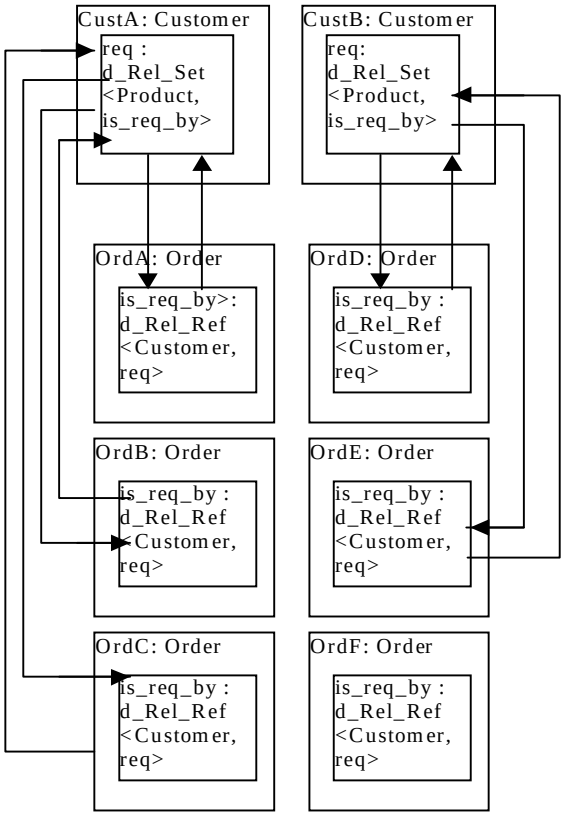

FIGURA. 3.1 - Conjunto inicial de relacionamentos um-para-muitos

3.5.3 Modificando um relacionamento

Um relacionamento pode ser modificado. Suponha que se tenha uma instância da classe Order, OrdC e que este passe a relacionar-se com o CustB. Pode-se supor que a aplicação execute o seguinte comando:

OrdC->is_req_by = CustB

Este código resulta na associação de OrdC para CustB não mais para CustA. As duas operações seguintes são realizadas implicitamente pelo software de banco de dados.

CustA->req.remove_element(OrdC);

CustB->req.insert_element(OrdC);

3.5.4 Removendo um relacionamento

Suponha que o OrdE seja removido do relacionamento com o CustB Executando-se a seguinte declaração atribui-se para as referências is_req_by e req o valor NULL respectivamente em OrdE e CustB. Isto resulta na remoção do relacionamento.

CustB->req.remove_element(OrdE);

Quando o comando é executado, a referência is_req_by de OrdE é removida implicitamente.

3.6 Mapeamento de um esquema relacional para classes C++

Na literatura há varias propostas de mapeamento entre os paradigmas relacional e OO [RAM97, FAH95, KEL95, SAE94, TSM94, MAR90, INT96]. Entretanto, a maioria delas não preenche os requisitos do presente trabalho que são: gerar classes C++ de acordo com a ODL a partir do banco de dados relacional existente, identificando as classes e os relacionamentos de associação, e incluir um método nas classes para identificar as chaves primárias. Neste sentido, será descrita nesta seção um mapeamento que permita que tabelas definidas no banco de dados sejam mapeadas para as classes em C++ ODL através de um conjunto de regras.

Em um banco de dados relacional os relacionamentos estão representados pelas chaves estrangeiras. Desta forma, será utilizado o conceito de chaves estrangeiras para definir as regras de mapeamento para classes no modelo OO.

De forma básica uma tabela sem relacionamento será mapeada para uma classe. Tabelas que estejam relacionadas através de chave estrangeira estão envolvidas em um relacionamento um-para-um ou um-para-muitos e serão mapeadas como um atributo d_Rel_Ref ou d_Rel_Set nas classes do esquema Orientado a Objetos. Tabelas que estejam relacionadas através de uma terceira tabela estão envolvidas em um relacionamento muitos-para-muitos e serão mapeadas como um atributo d_Rel_Set nas classes do esquema Orientado a Objetos. Nas seções abaixo será detalhada a forma como será realizado o mapeamento do relacional para o objeto.

3.7 Exemplo de utilização da arquitetura

Para exemplificar o esquema da arquitetura da aplicação será utilizada a classe base d_Object que é genérica e especializações de d_Object: Customer e Order, que são específicas, conforme mostra a figura 3.2. A notação usada para descrever a arquitetura é a linguagem de representação do UML [FOW 97].

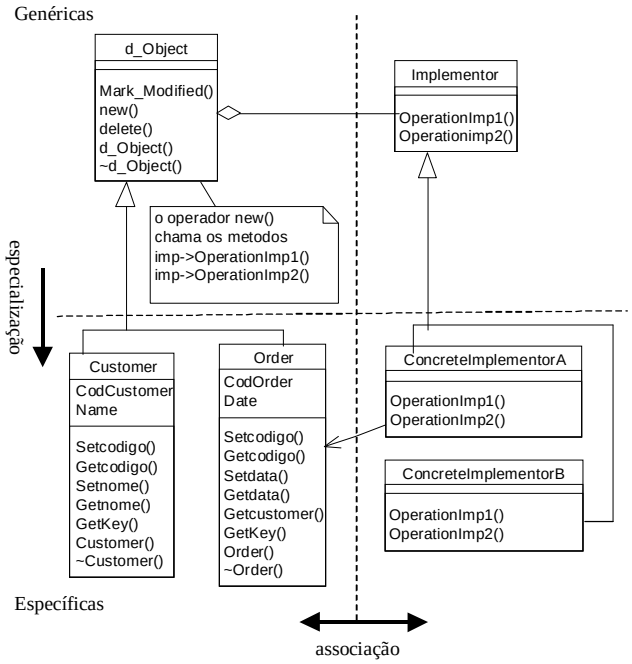

FIGURA 3.2 - Exemplo de esquema da arquitetura da aplicação.

A figura 3.2 apresenta o padrão de projeto (*design pattern*) Bridge[GAM94]. O padrão de projeto Bridge está sendo utilizado para o desenvolvimento desta arquitetura, porque o mesmo método de classes específicas terá implementações diferentes, não sendo, desta forma, possível implementar um único método na classe genérica. Por exemplo, o método delete() deverá ter uma implementação diferente para cada classe específica. Por isso não é possível implementá-lo em d_Object. Desta forma, o uso do padrão de projeto Bridge propõe que a classe de abstração e a classe de implementação sejam implementadas em uma hierarquia de classes separadas. Então, haverá uma hierarquia de classes para as classes de abstração (d_Object, Customer e Order) e uma hierarquia separada para as classes de implementação, sendo Implementor sua raiz. Por exemplo, a subclasse ConcreteImplementorA prove uma implementação baseada em Order.

Todas as operações das subclasses Customer e Order são implementadas em termos das operações abstratas da interface Implementor. Isto separa d_Object das várias implementações específicas.

Na figura 3.2 d_Object define uma interface de abstração e mantém uma referência para um objeto do tipo Implementor. Customer e Order estendem a interface definida por d_Object. A classe Implementor define uma interface para a classe de implementação. Esta interface não tem que corresponder exatamente à interface da abstração, podendo ser diferente. O que foi adicionado ao padrão de projeto Bridge é um ponteiro para a classe específica do domínio de problema, com a finalidade de acesso as variáveis membro.

Serão adotados como exemplo da aplicação, as entidades Customer e Order do esquema de banco de dados definido na figura 3.2.

4 Operações da OML sobre a arquitetura

A presente seção tem por objetivo mostrar como as várias operações da OML (*object manipulation language*) são implementadas na arquitetura de mapeamento. As operações da OML aparecem na interface ODMG na forma de métodos das classes ODMG.

4.1 Operação de criação de um objeto de banco de dados

Quando o usuário deseja criar um novo objeto, o mesmo deve chamar o construtor da classe que deseja inicializar. Este chama o método d_Object::new() da classe d_Object. O método new() de d_Object chama o método CObject::new() da classe CObject, que aloca espaço para o objeto na memória. A seguir, ele chama o construtor e o método created() da classe m_Descriptor. O objeto m_Descriptor é apontado para o objeto criado pelo operador new. O objeto m_Descriptor também é marcado como criado para indicar que o objeto inicializado pelo operador new foi criado. O método created() chama o método Addhead() de m_DescriptorSet, que é uma especialização de CobList. Este método adiciona um ponteiro para o objeto

m_Descriptor que por sua vez tem um ponteiro para o objeto inicializado. O diagrama de seqüência de criação de um novo objeto é mostrado na figura 4.1.

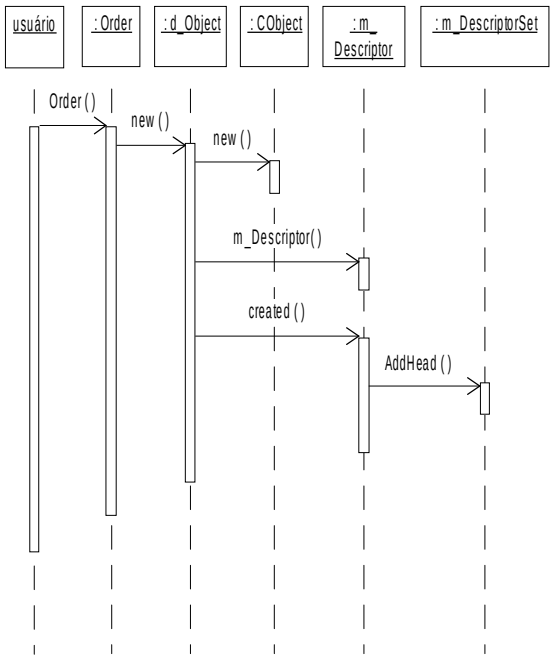

FIGURA 4.1 - Diagrama de seqüência de criação de um novo objeto

4.2 Operação de remoção de um objeto do banco de dados

Quando o usuário deseja remover um objeto, ele chama um método específico da classe a ser removida. Este método remove o objeto da memória, mas a sua remoção do disco só ocorrerá no fim da transação.

O método Order::delete() chama por herança o método d_Object::delete() da classe d_Object. O método delete() de d_Object chama o método m_Descriptor::delete() de m_Descriptor, que chama o método Setcod() pertencente à própria classe. O Setcod() é um método que tem a função de atribuir para a variável m_cod o valor da chave primária de Order. Após a execução do método Setcod() o objeto Order é removido da memória através da execução do método destrutor de Order. O diagrama de seqüência de remoção de um objeto é mostrado na figura 4.2.

4.3 Operação de seleção de um objeto do banco de dados

Quando o usuário deseja selecionar um objeto, ele chama o método d_Extent::select_element() da classe d_Extent. Por exemplo, quando o usuário deseja selecionar um objeto da classe Order, ele chama método select_element() de m_Extent<Order> que chama método concreto d_Extent::select_element(). Este método seleciona o objeto desejado no banco de dados, permitindo ao usuário manipular o objeto, e ao final da transação ele é gravado de forma persistente.

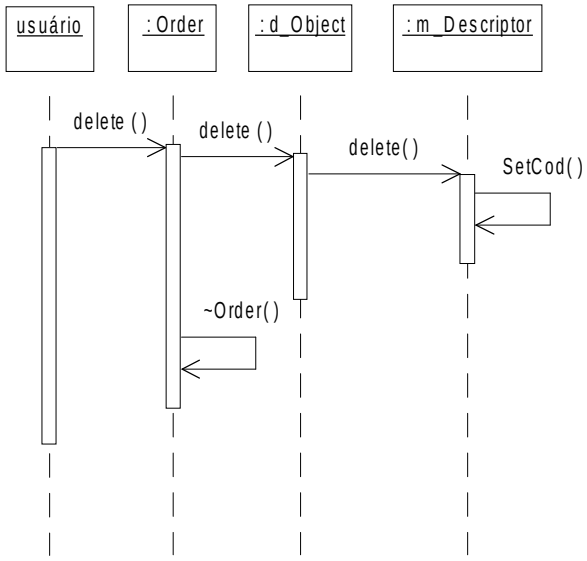

FIGURA 4.2 - Diagrama de seqüência de remoção de um objeto

O método m_Extent::select_element() da classe m_Extent chama os seguintes métodos da classe CRecordset: Open() e Move(). O método Open() é responsável pela seleção da tupla de acordo com o critério estabelecido. O método Move() posiciona o cursor na tupla selecionada pelo método Open(). Uma vez selecionado e posicionado sobre o registro pode-se construir o objeto na memória através do método Order(). O objeto criado na mémoria é marcado como carregado pelo método m_Descriptor::load(). O diagrama de seqüência para seleção de um objeto é mostrado na figura 4.3.

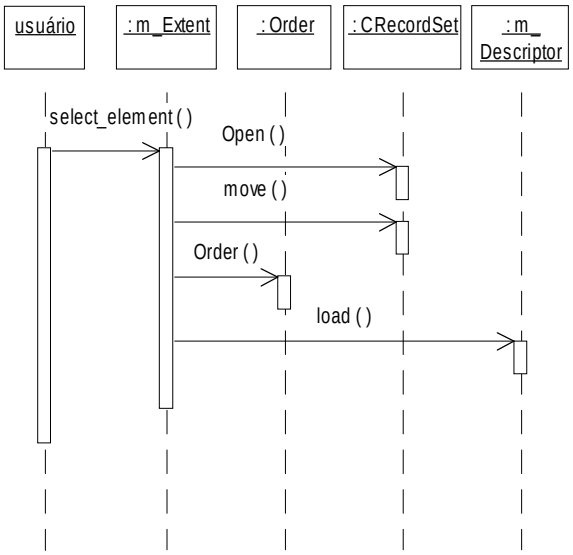

FIGURA 4.3 - Diagrama de seqüência do select_element da classe m_Extent

4.4 Estabelecendo um relacionamento um-para-muitos

Quando o usuário deseja estabelecer um relacionamento entre objetos, pode usar tanto o método d_Rel_Ref::operator= da classe d_Rel_Ref() quanto o método d_Rel_Set::insert_element() da classe d_Rel_Set. Estes métodos são responsáveis pelo relacionamento bidirecional entre os objetos. A classe d_Rel_Set e a classe d_Rel_Ref possuem respectivamente as classes agregadas, chamadas m_RelSetDescriptor e m_RelRefDescriptor que armazenam as chaves estrangeiras. É através destas classes de mapeamento que se tem os atributos para fazer persistente o relacionamento do objeto no banco de dados relacional. O diagrama de seqüência do operador de atribuição de relacionamento é mostrado na figura 4.4 e o diagrama de seqüência do método de atribuição de relacionamento é mostrado na figura 4.5.

O método d_Rel_Ref::operator= chama o método assign() da própria classe e o método insert_refelement() da classe d_Rel_Set que fazem a atribuição do relacionamento entre os objetos. O método assign() realiza as seguintes atividades: insere o valor da chave estrangeira em m_RelRefDescriptor, se houver, e faz com que o ponteiro aponte para um objeto do domínio do problema. A forma de inserir o valor da chave estrangeira do objeto na instância da classe m_RelRefDescriptor é através do método Getkey(). O método Getkey() chama o método Getkey() do objeto do domínio do problema que devolve o valor da chave estrangeira. Atribui-se o valor da chave estrangeira para m_RelRefDescriptor através do método Setkey(). Por último executa-se o método SetPointer() que aponta para o objeto da classe do domínio do problema. O diagrama de seqüência do método assign() de m_RelRefDescriptor é mostrado na figura 4.6.

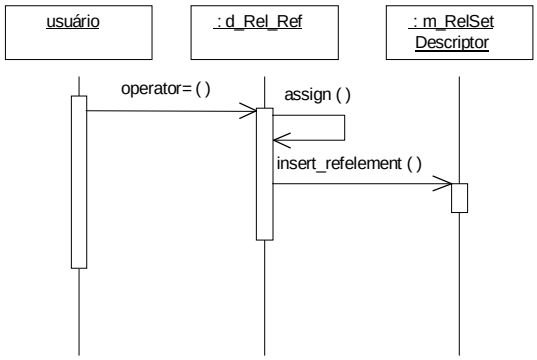

FIGURA 4.4 - Diagrama de seqüência do operador de atribuição de relacionamento

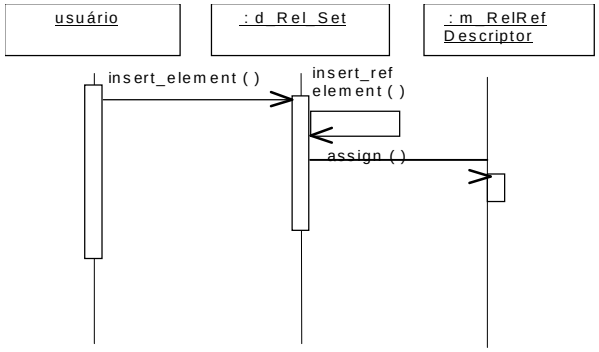

FIGURA 4.5 - Diagrama de seqüência do método de atribuição de relacionamento

O método Insert_refelement() realiza as seguintes atividades: insere na lista um ponteiro para o objeto através do método AddHead() e insere o valor da chave estrangeira no objeto da classe m_RelSetDescriptor. A forma de inserir o valor da chave estrangeira do objeto do domínio de problema no objeto da classe m_RelSetDescriptor é através do método Getkey(). O método Getkey() chama o método

Getkey() do objeto do domínio do problema que devolve o valor da chave estrangeira, se existir. Atribui-se o valor da chave estrangeira a m_RelSetDescriptor através do método Setkey(). Por último executa-se o método SetPointer() que aponta para o objeto do domínio do problema especificado. O diagrama de seqüência do método insert_refelement é mostrado na figura 4.7.

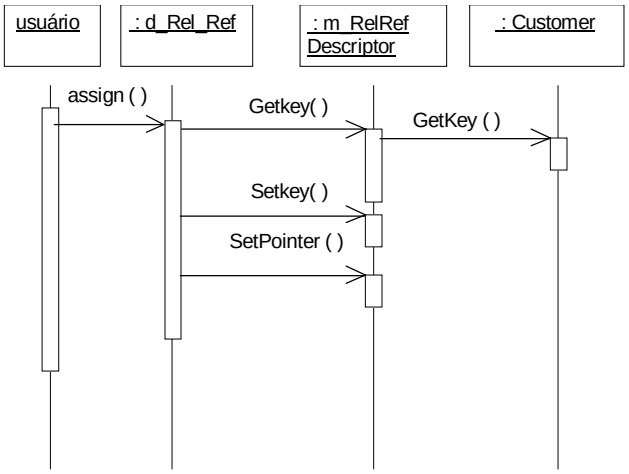

FIGURA 4.6 - Diagrama de seqüência do método atribui de RelRefCliente

4.9 Modificando um relacionamento de um-para-muitos

O operador d_Rel_Set::operator-> da classe d_Rel_Set, é o operador de modificação de um relacionamento. O método chama o método remove_element() e insert_element() da própria classe. O método remove_element() remove o relacionamento entre os objetos. O diagrama de seqüência do operador de modificação de relacionamento é mostrado na figura 4.8.

De forma similar o operador d_Rel_Ref::operator-> da classe d_Rel_Ref realiza a operação de modificação de um relacionamento. O operador chama o método clear() e operator=. O método clear() remove o relacionamento entre os objetos. O diagrama de seqüência do operador de modificação de relacionamento é mostrado na figura 4.9.

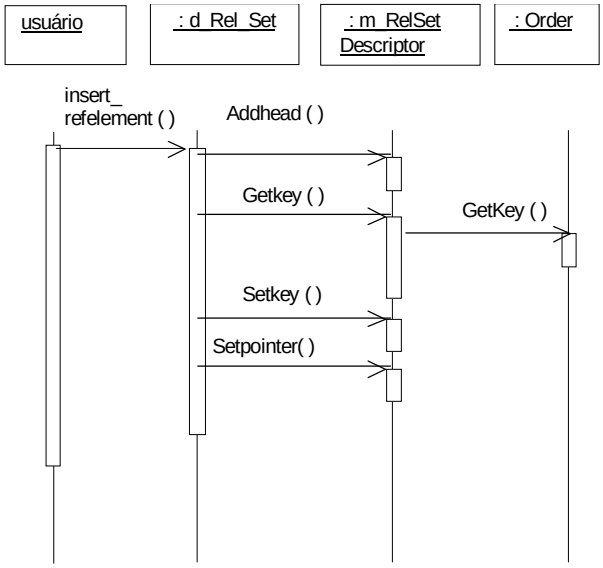

FIGURA 4.7 - Diagrama de seqüência do método insert_refelement

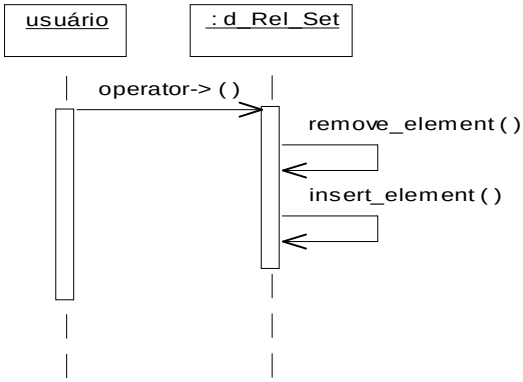

FIGURA 4.8 - Diagrama de seqüência do método operator-> de d_Rel_Set

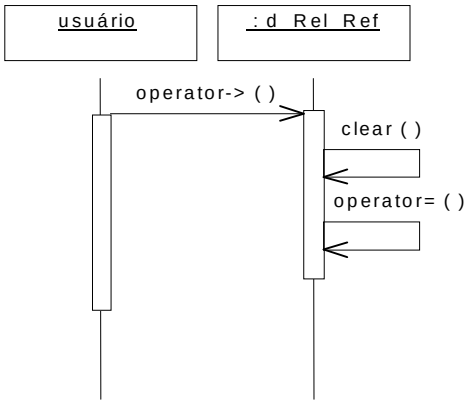

FIGURA 4.9 - Diagrama de seqüência do método operator-> de d_Rel_Ref

4.10 Removendo um relacionamento de um-para-muitos

Quando o usuário deseja remover um relacionamento um-para-muitos entre objetos ele usa o método d_Rel_Set::remove_element() da classe d_Rel_Set. A classe d_Rel_Set possui a classe agregada chamada m_RelSetDescriptor que mantém a chave estrangeira e um ponteiro para outro objeto participante do relacionamento. Uma vez executado este método, tanto a chave primária quanto o ponteiro são atribuídos a NULL. Como um objeto aponta para o outro, aquele que possui um ponteiro inverso ao objeto que teve o relacionamento removido também é removido. É função da arquitetura manter a integridade desses ponteiros, para evitar que exista alguma referência a um objeto inexistente.

O método d_Rel_Set::remove_element() da classe d_Rel_Set chama os seguintes métodos: RemoveAt() que remove um ponteiro para o objeto do domínio do problema, SetKeyNULL() que atribui a chave estrangeira o valor NULL e o método removepointer() da classe d_Rel_Ref que chama o método SetpointertoNULL(), que atribui para o ponteiro o valor NULL. O diagrama de seqüência do método de remoção de um relacionamento é mostrado na figura 4.10.

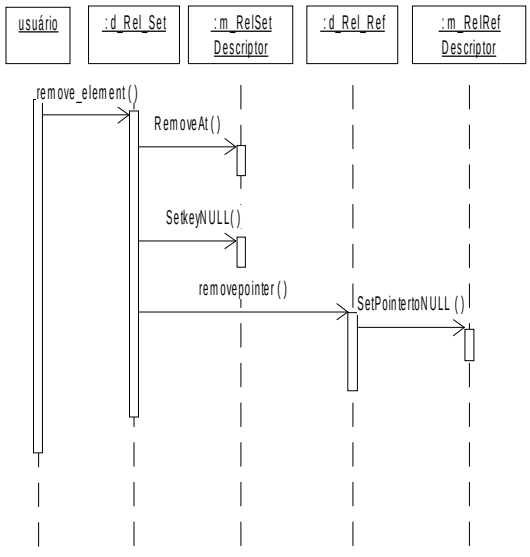

FIGURA 4.10 - Diagrama de seqüência do método remoção de relacionamento de d_Rel_Set

4 Conclusão

Neste trabalho, é apresentada uma arquitetura para implementar aplicações padrão ODMG sobre um Banco de Dados Relacional.

A abordagem diferencia-se de outras existentes na literatura. Na presente abordagem a base de dados é projetada usando os conceitos do modelo relacional de chave primária e chave estrangeira e não os conceitos de projeto de base de dados OO, identificadores de objetos e referências entre eles. Com isso, o modelo desenvolvido poderá ser usado sobre bases de dados legadas, já existentes em uma organização. Além disso, permitirá que as bases de dados sejam acessadas por ferramentas para SGBD relacional, como WEB browsers ou ferramentas de consultas.

Produtos de *software* comerciais, tais como o OpenDM e POET provêm todas as funcionalidades do padrão ODMG. A arquitetura aqui apresentada foi especificada para ser um produto de *software* para usuários que não têm condições de acesso a produtos comerciais desenvolver uma aplicação Orientada a Objetos e armazenar tuplas em um Banco de Dados Relacional existente.

Como contribuições da abordagem proposta destacam-se: a) a especificação de uma arquitetura que mostra um modo de realizar o mapeamento do paradigma Orientado a Objetos para o paradigma Relacional a partir de um RDBMS existente; b) apresentação de regras de mapeamento do relacional para o OO; c) apresentação da forma como ocorre a manipulação dos objetos no modelo de objetos do ODMG.

A partir deste trabalho novos tópicos de pesquisa poderão ser abordados.

O primeiro é um *framework*. O *framework* pode ser construído através da análise dos componentes genéricos do domínio de problema e mapeamento. A análise é feita através dos estudos dos seus subcomponentes.

O segundo é um assistente gerador de código. Este assistente é uma ferramenta de *software* que faz a análise do banco de dados relacional e deriva as classes do domínio de problema específico, bem como as classes específicas do componente Mapeamento. O Assistente gera somente as assinaturas dos métodos para classes do domínio de problema específico. Para o

componente especifico - Mapeamento, o Assistente gera as classes com os métodos.

Outros aspectos a serem estudados são a construção de uma *interface* ODMG estendida que dê ênfase a bases de dados temporais e a especificação de uma *cache* de objetos para otimização da ferramentas OO para RDBMS.

References

[BAN96] BANCILHON, Francois. **Object, Relational, Object-Relational & Relational-Object.** Abr. 1996. Disponível por WWW em: http://www.sigs.com/publications/docs/oc/9604.c.bancilhon.html (05/04/97).

[CAT94] CATTELL, R. G. G. **The Object database standard, ODMG-93**: Release 1.1. San Francisco: Morgan Kaufmann Publishers, 1994. 171p.

[CAT94a] CATTELL, R. G. G.. **Object data management:** object-oriented and extended relacional database system**.** Reading: Addison Wesley Publishing Company, 1994. 381p.

[CAT97] CATTELL, R. G. G, Barry, Douglas K. **The Object database standard, ODMG-2.0.** San Francisco: Morgan Kaufmann Publishers, 1997. 270p.

[COO97] COOPER, R. **Object Database**: An ODMG Aproach. Boston: International Thomson Computer Press, 1997. 499p.

[ELM89] ELMASRI, R.; NAVATHE, S. B. **Fundamentals of database systems**. Redwood City: Benjamin/Cummings, 1989. 802p.

[FAH95] FAHRNER, C.; VOSSEN, G. Tranforming Relational Database Schemas into Objecty-Oriented Schemas according to ODMG-93. In: INTERNATIONAL CONFERENCE ON DEDUCTIVE AND OBJECT-ORIENTED DATABASE, 4., 1995, Singapura. **Proceedings…** Singapura: Springer, 1995

[FOW97] FOWLER, M.; SCOTT, K. **UML Distilled:** Applying the Standard Object Modeling Language. Reading: Addison-Wesley Publishing Company, 1997. 179p.

[GAM94] GAMMA, E. **Design Patterns:** elements of reusable object-oriented software. Reading: Addison-Wesley Publishing Company. 1994

[INT96] INTEC MFC GROUP. **Using relational databases with MFC. An Object Oriented approach**. Igualada, Espanha: Intec – INT Informàtica i Noves Tecnologies, 1996. Disponível por WWW em: http://www.intec.es/mfcgroup/oodb.htm

[JOR97] JORDAN, D. **C++ Object Database:** Programming with the ODMG Standard. Reading: Addison-Wesley Publishing Company, 1997. 455p.

[KEL95] KELLER. A. M. et al. Architecting Object Applications for High Performance with Relational Databases. In: OOPSLA WORK-SHOP ON OBJECT DATA-BASE BEHAVIOR, BENCHMARKS, AND PERFORMANCE, 1995, Austin. Proceedings… Austin: [s.n.], 1995. Disponível por WWW em: http://www.db.stanford.edu/pub/keller

[MAR90] MARKOWITZ, VICTOR M.; MAKOWSKY, JOHANN A. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. **IEEE Transactions on Software Engeneering**, New York, v. 16, n. 10, p. 777-790, Aug. 1990

[02T97] O_2 TECNOLOGY. **O_2 version Mangement:** Reference manual: Release 4.6. Versailles: O_2 Tecnology, 1997.

[POE96] POET SOFTWARE. ODMG – Programer's Guide. Poet Software: [S. l.]: POET Software, 1996. Disponível por WWW em: http://www.poet/odmg/index

[RAM94] RAMANATHAN, S. Providing object-oriented access to a relational database. In: ANNUAL ACM SOUTHEAST CONFERENCE HELD IN TUSCALOOSA, 1994,

Tuscaloosa. **Proceedings…** Tuscaloosa: ACM Press, 1994.

[ROU89] ROUSSOPOULOS, N. A Relational Object Oriented System. In: INTERNATIONAL CONFERENCE FUNDATION OF DATA ORGANIZATION AND ALGORITHMS, 3., 1989, Paris. **Proceedings…**, Paris: Springer-Verlag, 1989.

[SAE95] SACRAMENTO, E. R.; LAENDER, A. H. F. Uma abordagem Orientada a Objetos para Projeto Lógico de Banco de Dados Relacionais. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 1994, São Carlos. **Anais…** São Carlos: [s.n.], 1994.

[SIE96] SIEMENS NIXDORF INFORMATIONS SYSTEM. OpenDM – ODMG – 93 1.0 User Guide. C-LAB. Cooperative Computing & Communication Laboratory: Paderborn, Germany, Nov. 1996.

Disponível por WWW em: http://www.c-lab.de/~opendm/odmg93/

[TSM94] TEIXEIRA, P.; SÁ, R.; MIRANDA, V. Implementação do standard ODMG - 93, sobre SGBDs relacionais.

Disponível por WWW em: http://albertina.inesc.pt/leic/1994

/34161.html (07/04/97)