Persistência de Objetos no Framework MIOLO

Ely Edison da Silva Matos

Universidade Federal de Juiz de Fora
ely.matos@ufjf.edu.br

1. Introdução

A Orientação a Objetos (OO) tem sido o paradigma de desenvolvimento de software mais utilizado nos últimos anos. Desde a sua popularização, em meados dos anos 80, uma questão sempre esteve presente: como recuperar os valores dos objetos utilizados durante a execução de um aplicativo depois que este é finalizado.

Os objetos utilizados por uma aplicação são armazenados em memória primária e, devido a sua volatilidade, seus valores são perdidos sempre que a aplicação termina; esses objetos são ditos transientes. Se os objetos são armazenados em memória secundária, eles podem ser recuperados mesmo após o término da aplicação que os manipulou. Um objeto que não perde seus valores ao término da aplicação é dito persistente.

Várias soluções foram propostas durante os últimos anos para a persistência de objetos. Dentre elas podemos citar o desenvolvimento de linguagens de programação persistentes; a extensão do modelo de dados dos bancos de dados relacionais existentes; o desenvolvimento de bancos de dados orientados a objetos e mais recentemente o desenvolvimento de camadas de persistência para permitir o mapeamento entre os objetos da aplicação e bancos de dados relacionais. Algumas dessas soluções foram bem aceitas e outras não, mas todas elas propiciaram novos conhecimentos e técnicas importantes para persistência de objetos.

A Universidade Federal de Juiz de Fora (UFJF) adotou, em 2003, o Framework MIOLO como base de desenvolvimento de seu sistema corporativo, o SIGA (Sistema Integrado de Gestão Acadêmica). A fim de utilizar de forma extensiva a Orientação a Objetos, foi desenvolvida uma solução para a persistência de objetos em bancos de dados relacionais a ser usada no framework. Este texto apresenta uma visão geral desta solução, com exemplos de sua utilização.

2. Descasamento de Impedância Objeto-Relacional

A Orientação a Objetos (OO) permite a construção de aplicações formadas por objetos que possuem dados e comportamento. Os Sistemas Gerenciadores de Bancos de Dados Relacionais (SGBDR) trabalham com armazenamento de dados em tabelas e a manipulação desses dados por linguagens de consulta e manipulação como a SQL.

É comum desenvolvedores utilizarem linguagens orientadas a objetos para implementar os aspectos lógicos de um sistema e um SGBDR para persistência dos dados. Muitos projetos com essas características já fracassaram; isso demonstrou que as duas tecnologias não se encaixam perfeitamente, ou seja, não podem trabalhar juntas sem implicar na perda de parte do potencial de ambas. O termo **descasamento de impedância objeto/relacional** (*object-relational impedance mismatch*) foi adotado na década de 90, sendo utilizado para indicar esse fato. O termo original vem da área da Engenharia Elétrica e é utilizado quando componentes heterogêneos perdem sua capacidade de funcionamento máxima quando utilizados em conjunto, isso devido a diferenças de algumas características (como resistência ou potência).

O descasamento ocorre porque as tecnologias possuem bases diferentes: A tecnologia orientada a objetos está baseada em princípios de engenharia de software bem fundamentados, enquanto a tecnologia relacional se apóia em princípios matemáticos (AMBLER, 2002). Cabe aos projetistas o trabalho de minimizar o descasamento durante o desenvolvimento de um projeto de software. A falta de planejamento pode ampliar as diferenças entre os paradigmas, aumentando demasiadamente o esforço necessário para manter os dois esquemas (relacional e de objetos) consistentes, podendo implicar em um projeto mal sucedido.

Vários fatores podem evidenciar o descasamento entre as tecnologias. Um deles é a navegação entre entidades do sistema (objetos na OO ou tabelas em SGBDR). Com a OO, isto é feito através de referências que representam relacionamentos entre classes; nos SGBDR, o mesmo objetivo é alcançado com a junção de linhas de uma ou mais tabelas. Outro fator diz respeito ao código escrito durante a implementação de um sistema: um programador muitas vezes tem de conhecer uma linguagem para manipulação de dados (como a SQL) além da linguagem natural na qual o sistema está sendo desenvolvido. Quanto maior o descasamento, mais linhas de código terão que ser escritas para solucionar o problema e o projeto resultante pode ser difícil de se manter e testar. Além disso, o desempenho do sistema pode cair devido à complexidade necessária para coordenar os esquemas heterogêneos.

2. Framework MIOLO

O framework MIOLO é um software brasileiro para desenvolvimento de aplicações acessíveis via web, utilizando Orientação a Objetos através da linguagem PHP5. Sua construção teve início em 2001, na universidade UNIVATES[1] (Lajeado/RS) e atualmente está sob responsabilidade da cooperativa SOLIS[2]. Atualmente a UFJF utiliza a versão 2.0, sendo que a versão 2.5 está em fase de testes.

O framework MIOLO apresenta as seguintes características principais:

- Arquitetura em camadas, separando o código referente à apresentação, regras de negócio, recursos externos e integração.
- Possibilidade de uso do padrão MVC (*Model-View-Controller*), buscando melhorar a estrutura do código.
- Implementação do padrão *Front Controller*.
- Rico conjunto de componentes de interface com o usuário, programado em PHP5 e completamente extensíveis, possibilitando um modelo de programação *event-driven* e encapsulando componentes escritos em Javascript.
- Gerenciamento de sessão e estado.
- Geração do código HTML *Tableless* pelo próprio framework, permitindo a criação de aplicações *cross-browser*.
- Implementação de suporte para AJAX.
- Mecanismos de segurança, com autenticação (via banco de dados ou LDAP), controle de permissões e manutenção de logs.
- Camada DAO (*Data Access Objects*) para abstração de acesso a banco de dados e camada para persistência de objetos.
- Customização da interface através de temas existentes ou criados pelo usuário, com uso de CSS e templates.
- Geração de relatórios através de arquivos PDF, utilizando as bibliotecas ezPDF e JasperReports.

O uso de PHP5 permite estender facilmente as funcionalidades do framework para questões mais específicas, através das inúmeras bibliotecas de classes disponíveis em software livre.

3. Camada de Persistência

No framework MIOLO, o desenvolvedor encapsula as regras de negócio em objetos que estendem a classe MBusiness. Até a versão 2.0, esta classe definia os mecanismos de acesso a bancos de dados, através da camada DAO, sendo necessário que o desenvolvedor usasse comandos SQL. A partir da versão 2.0, a classe MBusiness estende a classe MPersistentObject, que define uma interface simples para armazenamento e recuperação dos objetos.

A implementação da camada de persistência no framework MIOLO está baseada nos trabalhos de Ambler (2005) e Rudoy (2002). A figura 1 apresenta as principais classes da camada.

---

[1] http://www.univates.br
[2] http://www.solis.coop.br/

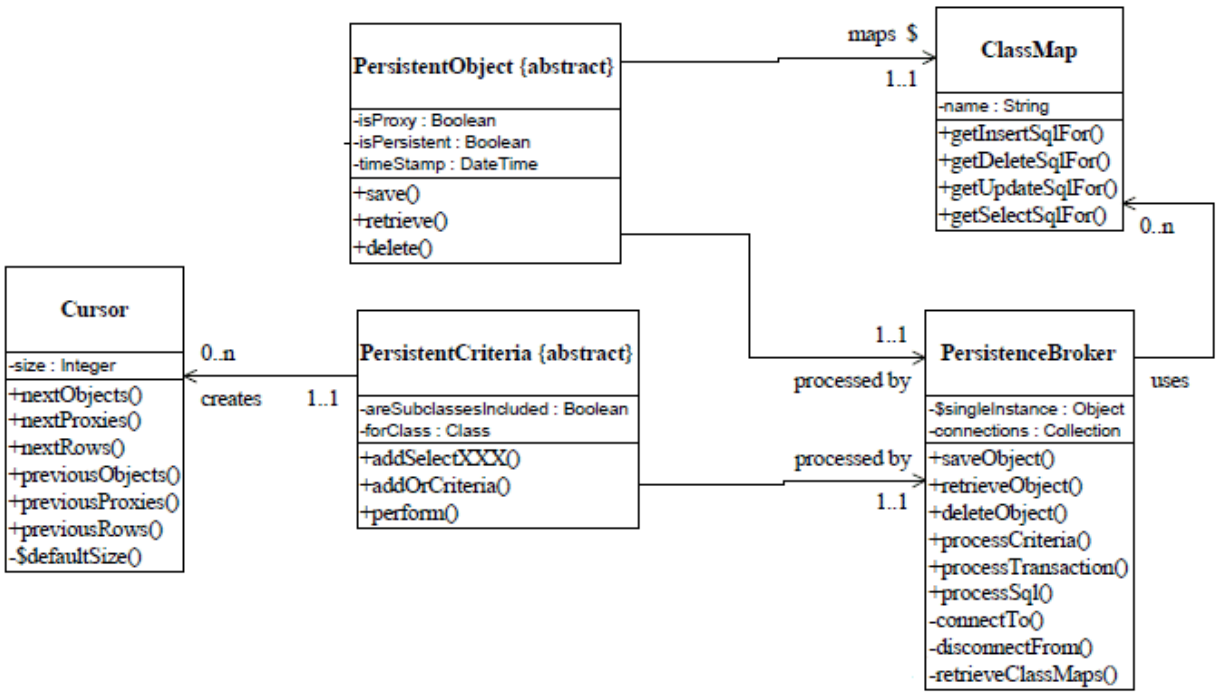

Figura 1 – Classes principais da camada de persistência.

A implementação atual estendeu a proposta de Ambler em vários pontos e possui as seguintes características principais:

- O mecanismo de persistência está encapsulado. A classe MBusiness passa a herdar da classe de persistência, tornando os objetos de negócio virtualmente persistentes. Estão disponíveis métodos tais como **save**, **delete** e **retrieve** que tratam automaticamente o acesso ao banco de dados.
- Ações sobre múltiplos objetos. Mecanismos para recuperação e remoção de múltiplos objetos. Os mecanismos de recuperação permitem retornar objetos MQuery (com acesso ao resultado da consulta SQL via camada DAO do MIOLO) ou Cursores (um cursor está implementado como um array de objetos).
- Suporte a "lazy read" através do uso de proxies. Um objeto proxy permite recuperar apenas alguns atributos do objeto, evitando o *overhead* de recuperar todos os atributos. Isto é útil em situações tais como a exibição de listas de seleção, por exemplo.
- Suporte a associações. Quando um objeto é recuperado, removido ou atualizado, a mesma ação pode ser realizada nos objetos associados, se desejado. As associações do tipo ManyToMany podem ser tratadas automaticamente pela camada de persistência.
- Suporte a herança, tornando possível mapear uma árvore de herança para um esquema no banco de dados.
- Suporte a transações, geração automática de identificadores (OID), geração automática do comando SQL e acesso a diferentes bancos de dados, que são características implementadas pela camada DAO do MIOLO.
- Suporte a conversão de valores de atributos, através de classes de conversão.
- Suporte a indexação de campos (vários campos da tabela retornados como um array em PHP5).
- Suporte a operações de conjunto (INTERSECT, MINUS, UNION).
- Suporte a campos tipo BLOB.

Para exemplificar o processo de mapeamento objeto-relacional e a execução de consultas, são mostrados o diagrama de classes (figura 2) e o esquema do banco de dados correspondente (figura 3).

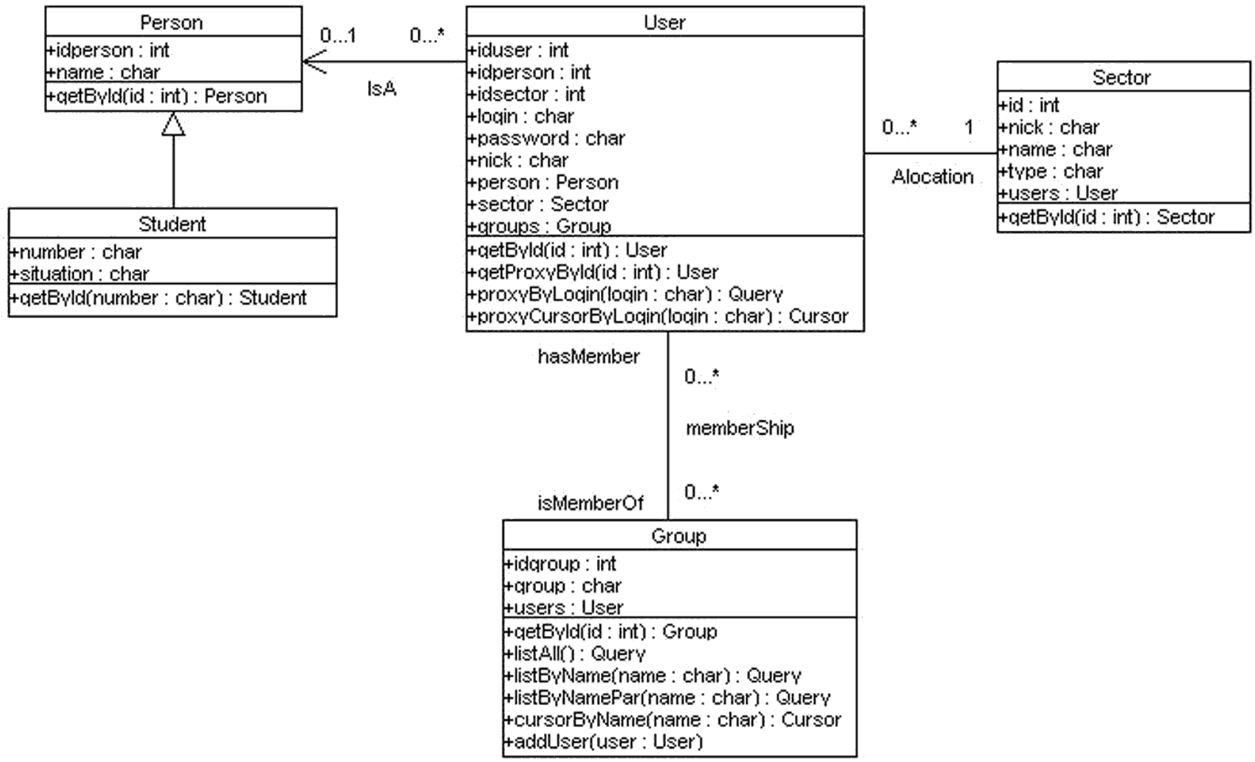

Figura 2 – Modelo de Classes

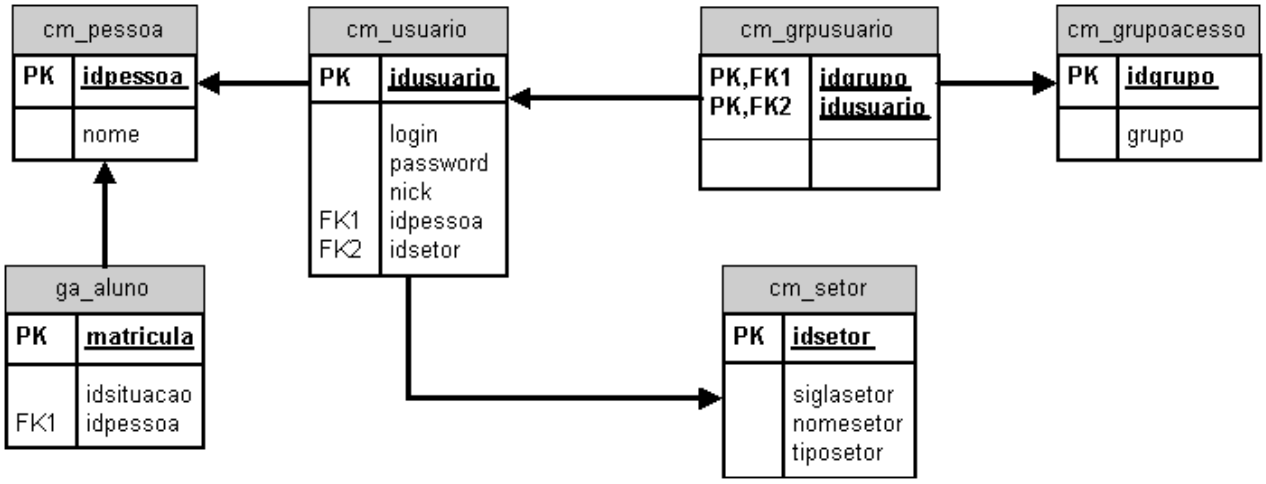

Figura 3 – Esquema do banco de dados relacional

3.1. Mapeamento

No contexto da camada de persistência, o mapeamento diz respeito a como são representados os objetos, seus atributos/propriedades e suas associações, em termos do modelo relacional.

Para representar o mapeamento das classes persistentes e das classes associativas (classes que são usadas dentro da camada de persistência para representar as associações muitos-para-muitos – ManyToMany) são usados arquivos XML. São criados mapas para as classes, para os atributos, para as associações, para as colunas e para as tabelas, permitindo uma grande flexibilidade no processo de mapeamento. A figura 4 mostra um exemplo de mapeamento (classe User).

```
<?xml version="1.0" standalone="yes"?>
<map>
   <moduleName>persistence</moduleName>
   <className>user</className>
   <tableName>cm_usuario</tableName>
   <databaseName>common</databaseName>
   <attribute>
      <attributeName>iduser</attributeName>
         <columnName>idusuario</columnName>
         <key>primary</key>
         <idgenerator>seq_cm_usuario</idgenerator>
   </attribute>
   <attribute>
      <attributeName>login</attributeName>
         <columnName>login</columnName>
         <proxy>true</proxy>
   </attribute>
   <attribute>
      <attributeName>password</attributeName>
         <columnName>password</columnName>
   </attribute>
   <attribute>
      <attributeName>nick</attributeName>
         <columnName>nick</columnName>
   </attribute>
   <attribute>
      <attributeName>idperson</attributeName>
         <columnName>idpessoa</columnName>
         <proxy>true</proxy>
   </attribute>
   <attribute>
      <attributeName>idsector</attributeName>
         <columnName>idsetor</columnName>
         <proxy>true</proxy>
   </attribute>
   <attribute>
      <attributeName>person</attributeName>
   </attribute>
   <attribute>
      <attributeName>groups</attributeName>
   </attribute>
   <attribute>
      <attributeName>sector</attributeName>
   </attribute>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>person</toClassName>
      <cardinality>oneToOne</cardinality>
      <target>person</target>
         <retrieveAutomatic>true</retrieveAutomatic>
         <saveAutomatic>true</saveAutomatic>
         <entry>
            <fromAttribute>idperson</fromAttribute>
            <toAttribute>idperson</toAttribute>
      </entry>
   </association>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>group</toClassName>
      <associativeClassModule>persistence</associativeClassModule>
      <associativeClassName>groupuser</associativeClassName>
      <cardinality>manyToMany</cardinality>
      <target>groups</target>
         <retrieveAutomatic>false</retrieveAutomatic>
         <saveAutomatic>false</saveAutomatic>
         <direction>
            <fromAttribute>users</fromAttribute>
            <toAttribute>groups</toAttribute>
```

```
      </direction>
   </association>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>sector</toClassName>
      <cardinality>oneToOne</cardinality>
      <target>sector</target>
         <retrieveAutomatic>true</retrieveAutomatic>
         <saveAutomatic>false</saveAutomatic>
         <entry>
            <fromAttribute>idsector</fromAttribute>
            <toAttribute>id</toAttribute>
      </entry>
   </association>
</map>
```

Figura 4 – Mapeamento da classe User

3.2. Objetos persistentes

Uma vez que a classe MBusiness passa a herdar da classe PersistentObject, todos os objetos de negócio são automaticamente persistentes, ou seja, podem usar os métodos de persistência. O acesso direto à camada DAO continua válido, permitindo construir comandos SQL de forma independente da camada de persistência.

Os principais métodos de um objeto persistente são:

- *function retrieve():* Preenche os atributos do objeto, acessando o banco de dados e recuperando o(s) registro(s) necessário(s) com base no atributo chave do objeto. O atributo chave do objeto deve ser preenchido antes da execução do método. As associações podem ser automaticamente recuperadas, dependendo da configuração feita no mapeamento.
- *function save():* Armazena (persiste) um objeto no banco de dados. O armazenamento pode envolver uma ou mais inclusões ou atualizações, em uma ou mais tabelas do banco de dados. As associações podem ser automaticamente armazenadas, dependendo da configuração feita no mapeamento.
- *function delete():* Remove um objeto no banco de dados. A remoção pode envolver uma ou mais exclusões ou atualizações, em uma ou mais tabelas do banco de dados. As associações podem ser automaticamente removidas, dependendo da configuração feita no mapeamento.
- *function retrieveAssociation($target):* Recupera os dados referentes a uma associação e preenche o atributo correspondente com o objeto (ou com um array de objetos). Geralmente usado quando a recuperação da associação não é automática.
- *function getCriteria():* Retorna um objeto usado para executar consultas customizadas no banco de dados (seção 3.4).

3.3. Cursor

Um cursor é um array de objetos. Pode ser usado como resultado de uma consulta ao banco de dados.

3.4. Queries

As consultas ao banco de dados são realizadas através da MOQL (*Miolo Object Query Language*), que usa um mecanismo chamado "*query by criteria*". Através deste mecanismo, definimos um "criteria" (um critério para a consulta), instanciando um objeto do tipo RetrieveCriteria. Um objeto RetrieveCriteria está associado a uma classe persistente que serve de base para a consulta (as consultas sempre são feitas sob a perspectiva de uma determinada classe persistente), sendo obtido através do método getCriteria().

A figura 5 mostra exemplos de consultas. Para efeito dos exemplos, $this->group, $this->user, $this->sector, $this->person, $this->student são objetos das respectivas classes.

Consulta simples

```
       $criteria = $this->group->getCriteria();
       $query = $criteria->retrieveAsQuery();
SQL gerado:
       SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso
```

Consulta com filtro composto

```
       $criteria = $this->group->getCriteria();
       $cc = new CriteriaCondition;
       $cc->addCriteria($criteria->getCriteria('group','LIKE',"'%A%'"));
       $cc->addOrCriteria($criteria->getCriteria('group','LIKE',"'%E%'"));
       $criteria->addCriteria('group','LIKE',"'C%'");
       $criteria->addCriteria($cc);
       $query = $criteria->retrieveAsQuery();
SQL gerado:
       SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso WHERE
((cm_grupoacesso.grupo LIKE 'C%') AND (((cm_grupoacesso.grupo LIKE '%A%') OR
(cm_grupoacesso.grupo LIKE '%E%')) ))
```

Consulta com parâmetro

```
       $criteria = $this->group->getCriteria();
       $criteria->addCriteria('group','LIKE',"?");
       $criteria->addOrCriteria('group','LIKE',"'C%'");
       $query = $criteria->retrieveAsQuery("A%");
SQL gerado:
       SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso WHERE
((cm_grupoacesso.grupo LIKE 'A%') OR (cm_grupoacesso.grupo LIKE 'C%'))
```

Consulta com join, via associação oneToOne, com definição de colunas

```
       $criteria = $this->user->getCriteria();
       $criteria->addCriteria('sector.nick','LIKE',"'PROR%'");
       $criteria->addColumnAttribute('login');
       $criteria->addColumnAttribute('sector.nick');
       $query = $criteria->retrieveAsProxyQuery();
SQL gerado:
       SELECT cm_usuario.login,cm_setor.siglasetor FROM cm_usuario,cm_setor WHERE
(cm_setor.siglasetor LIKE 'PROR%') and (cm_usuario.idsetor=cm_setor.idsetor)
```

Consulta com join, via associação manyToMany, com definição de colunas e agrupamento

```
       $criteria = $this->group->getCriteria();
       $criteria->addColumnAttribute('group');
       $criteria->addColumnAttribute('count(users.iduser)');
       $criteria->addGroupAttribute('group');
       $query = $criteria->retrieveAsQuery();
SQL gerado:
       SELECT cm_grupoacesso.grupo,count(cm_usuario.idusuario) FROM
cm_grpusuario,cm_grupoacesso,cm_usuario WHERE
(cm_grpusuario.idgrupo=cm_grupoacesso.idgrupo) and
(cm_grpusuario.idusuario=cm_usuario.idusuario) GROUP BY cm_grupoacesso.grupo
```

Consulta com auto-relacionamento

```
       $criteria = $this->sector->getCriteria();
       $criteria->setAutoAssociationAlias('S1','S2');
       $criteria->addColumnAttribute('S1.nick');
       $criteria->addColumnAttribute('S2.nick');
       $criteria->addCriteria('S1.nick', '=', 'S2.parent');
       $criteria->addCriteria('S1.parent','=', "'PROGRAD'");
       $query = $criteria->retrieveAsQuery();
SQL gerado:
       SELECT S1.siglasetor,S2.siglasetor FROM cm_setor S1,cm_setor S2 WHERE
((S1.siglasetor = S2.paisetor) AND (S1.paisetor = 'PROGRAD'))
```

Consulta com subquery referenciando a query externa

```
       $subCriteria = $this->user->getCriteria();
       $subCriteria->setReferenceAlias('S');
       $subCriteria->addColumnAttribute('count(iduser)');
```

```
       $subCriteria->addCriteria('idsector','=',"S.idsector");
       $criteria = $this->sector->getCriteria();
       $criteria->setAlias('S');
       $criteria->addCriteria($subCriteria, '>', '150');
       $query = $criteria->retrieveAsQuery();
SQL gerado:
       SELECT S.idsetor,S.siglasetor,S.nomesetor,S.paisetor,S.tiposetor FROM cm_setor S
WHERE ((SELECT count(cm_usuario.idusuario) FROM cm_usuario WHERE (cm_usuario.idsetor =
S.idsetor)) > 150)

Consulta com herança entre classes
       $criteria = $this->student->getCriteria();
       $criteria->addColumnAttribute('number');
       $criteria->addColumnAttribute('name');
       $criteria->addCriteria('name','LIKE',"'FELIPE A%'");
       $query = $criteria->retrieveAsQuery();
SQL gerado:
       SELECT ga_aluno.matricula,cm_pessoa.nome FROM ga_aluno,cm_pessoa WHERE
(cm_pessoa.nome LIKE 'FELIPE A%') and (ga_aluno.idpessoa=cm_pessoa.idpessoa) and
(ga_aluno.idpessoa=cm_pessoa.idpessoa)
```

Figura 5 – Exemplos de consultas em MOQL

4. Conclusão

Este texto apresentou, de forma superficial, as principais características da camada de persistência implementada no framework MIOLO.

A implantação desta camada, além de ser uma contribuição da UFJF para a comunidade deste software livre, permitiu melhorar a qualidade do código do SIGA. O desenvolvimento de ferramentas também foi simplificado. Como exemplo, está sendo implementado no SIGA um gerador de relatórios customizados, que permite ao usuário definir colunas, filtros e agrupamentos dos relatórios. Este gerador utiliza a camada de persistência para ocultar os detalhes de implementação do banco de dados (tais como as junções de tabelas).

Como trabalhos futuros, existem propostas de automatizar o processo de geração dos mapas XML a partir dos diagramas de classes, bem como criar ferramentas gráficas que facilitem a geração das consultas.

Referências

AMBLER, S.J., 2005, *The Design of a Robust Persistent Layer For Relational Databases.* Disponível em http://www..ambysoft.com/persistenceLayer.pdf.

AMBLER, S.J., 2002, *The Object-Relational Impendance Mismatch.* Disponível em http://www.agiledata.org.

RUDOY, Artyom, 2002, *Persistence Layer*. Disponível em http://artyomr.narod.ru.