Um Arcabouço Orientado por Aspectos para Implementação Automatizada de Persistência

César Francisco de M. Couto[1], **Marco Túlio O. Valente**[2], **Roberto da S. Bigonha**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

[2]Departamento de Ciência da Computação – Pontifícia Universidade Católica de Minas Gerais (PUCMinas)

{cesarfmc,bigonha}@dcc.ufmg.br, mtov@pucminas.br

Resumo. *Este artigo apresenta um arcabouço orientado por aspectos para a implementação automatizada de persistência. A implementação de persistência em bancos de dados relacionais pode ser dividida nos seguintes sub-interesses: controle de conexões, controle de transações e controle de sincronização de objetos. O arcabouço proposto inclui uma ferramenta de geração capaz de gerar aspectos para implementar cada um desses sub-interesses.*

Abstract. *This paper presents an aspect oriented framework to support the automated implementation of data persistence. The implementation of persistence in relational databases can be classified in the following concerns: connection control, transaction control and object synchronization control. The proposed framework includes a tool that generates aspects for each of these concerns.*

1. Introdução

Analisando o emprego de programação orientada por objetos em sistemas de computação, percebe-se que existem dificuldades em implementar de forma modularizada certos interesses, os quais são chamados de transversais. A implementação desses interesses em linguagens orientadas por objetos leva a dois problemas conhecidos como entrelaçamento e dispersão de código. Programação orientada por aspectos é uma nova técnica de programação cujo objetivo é permitir a implementação de forma modularizada de interesses transversais, melhorando, ou até mesmo eliminando, a dispersão e o entrelaçamento de código [3, 6].

Dentre aqueles interesses que claramente possuem um comportamento transversal, destaca-se o suporte à persistência. Assim, não surpreende que a implementação de persistência seja um exemplo clássico de emprego de orientação por aspectos [1, 4, 5]. No entanto, os arcabouços (*frameworks*) já propostos para implementação de persistência por meio de aspectos demandam um esforço considerável de programação, exigindo a definição de diversos aspectos e interfaces. A implementação de tais aspectos é normalmente tediosa e, portanto, sujeita a erros. Além disso, existe uma forte dependência entre os aspectos implementados e as classes da aplicação base. Logo, modificações nessas classes freqüentemente implicam em redefinição dos aspectos de persistência. Na prática, tais dificuldades podem dificultar a adoção desses arcabouços por parte de desenvolvedores de sistemas de grande porte. Daí a importância de ferramentas para geração de código

que auxiliem usuários a criar os aspectos de persistência específicos de uma determinada aplicação [5].

Neste artigo, apresenta-se um arcabouço de persistência orientado por aspectos. Diferentemente de outras soluções, o arcabouço proposto inclui um gerador de aspectos, o qual é responsável por criar de forma automática todos os aspectos responsáveis por persistir objetos de uma aplicação alvo em um meio de armazenamento não-volátil. Para gerar tais aspectos, exige-se apenas que o usuário do arcabouço informe parâmetros como os seguintes:pontos do sistema onde deve-se abrir e fechar conexões, métodos que atualizam o estado de objetos persistentes e o mapeamento de tais objetos para tabelas de um banco de dados relacional. Esses parâmetros são todos informados por meio de uma interface gráfica, dispensando assim o usuário de dominar uma linguagem orientada por aspectos. Os aspectos gerados pelo arcabouço são implementados em AspectJ [2]. Na versão atual do sistema, assume-se que o meio de armazenamento persistente é um banco de dados relacional. No entanto, nada impede que o arcabouço seja utilizado com outros meios de armazenamento. Por fim, o sistema não requer que a aplicação alvo obedeça a uma arquitetura particular (por exemplo, que seja um sistema *Web*, um sistema três camadas etc). O arcabouço é genérico o suficiente para permitir que persistência seja incorporada a quaisquer aplicações orientadas por objetos desenvolvidas em Java.

Este artigo é estruturado da seguinte forma. A Seção 2 descreve as principais funcionalidades, a ferramenta de geração de aspectos e a interface de programação do arcabouço de persistência proposto. Na Seção 3, apresenta-se um estudo de caso consistindo no emprego do arcabouço para adicionar persistência em um sistema *Web* de comércio eletrônico. A Seção 4 compara e analisa o arcabouço com soluções similares, e a Seção 5 conclui o artigo.

2. Descrição do Arcabouço

A implementação de persistência em bancos de dados relacionais pode ser dividida nos seguintes sub-interesses: controle de conexões, controle de transações e controle de sincronização de objetos. O arcabouço proposto inclui um gerador capaz de gerar aspectos para implementar cada um desses sub-interesses, conforme descrito nas subseções seguintes.

2.1. Aspectos para Controle de Conexões

Estes aspectos devem ser capazes de interferir na execução de um sistema orientado por objetos de forma a definir pontos de sua execução a partir dos quais deseja-se fazer uso do serviço de persistência e pontos onde deseja-se finalizar o uso de tal serviço. Além de estabelecer tais pontos, deve-se também informar alguns parâmetros que são necessários para ativar a conexão com o banco de dados, tais como: URL do banco, *drivers* a serem usados, *login* e senha para conexão, etc. De forma análoga a outros arcabouços de persistência, o seguinte aspecto abstrato é usado para capturar os pontos de junção essenciais ao estabelecimento de conexões:

```
abstract aspect ConnectionControlAspect {
  abstract pointcut openConnection();
  abstract pointcut closeConnection();
  before(): openConnection() { openConnectionBD(); }
  after(): closeConnection() { closeConnectionBD(); }
}
```

O primeiro conjunto de junção abstrato (`openConnection`) captura os pontos de junção antes (comportamento transversal `before`) dos quais deve-se abrir uma conexão com o banco dados (método `openConnectionBD`). Esta conexão deve ser desativada após (comportamento transversal `after`) os pontos de junção capturados pelo conjunto de junção `closeConnection`. A desativação propriamente dita é realizada pelo método concreto `closeConnectionBD`. Por questões de espaço, omite-se o código dos métodos `openConnectionBD` e `closeConnectionBD`.

A partir de configurações feitas pelo usuário do arcabouço em uma interface gráfica, a ferramenta de geração cria um aspecto derivado do aspecto mostrado e que implementa seus conjuntos de junção abstratos. Detalhes dessa ferramenta, bem como o aspecto gerado pela mesma, são descritos na Seção 3.

2.2. Aspectos para Controle de Transações

Estes aspectos são responsáveis por definir os métodos da aplicação alvo que são transacionais, isto é, métodos cuja execução deve preservar as propriedades ACID (atomicidade, consistência, independência e durabilidade) ao atualizar o estado de objetos persistentes. Basicamente, antes de iniciar a atualização do estado, tais métodos devem se comunicar com o banco de dados para informar o início da transação. Caso a atualização seja efetuada com sucesso, eles devem comunicar tal fato ao gerenciador de banco de dados (*commit*). Caso contrário, deve-se comunicar o insucesso da atualização (*rollback*).O seguinte aspecto abstrato é usado para capturar conjuntos de junção essenciais ao controle de transações:

```
abstract aspect TransactionControlAspect {
  abstract pointcut transactionalMethods();
  around(): transactionalMethods() {
    try {
      beginTransaction();
      proceed();
      commitTransaction();
    } catch(Exception e) {rollbackTransaction();}
  }
}
```

O conjunto de junção abstrato `transactionalMethods` captura os pontos de junção relativos à execução de métodos transacionais. Tal execução deve ser precedida por chamadas a `beginTransaction` e `commitTransaction`. Em caso de falhas, deve-se providenciar a realização de um *rollback*.

Na Seção 3, mostra-se um aspecto concreto derivado do aspecto mostrado nesta seção, o qual é gerado automaticamente pelo gerador de aspectos que acompanha o arcabouço proposto.

2.3. Aspectos para Sincronização de Objetos

A geração de aspectos para sincronização entre objetos na memória e sua representação no banco de dados consiste em definir primeiramente quais as classes da aplicação são classes de persistência. Por exemplo, para um sistema de comércio eletrônico possivelmente as classes de persistência são Cliente, Pedido, ItemPedido, Produto etc. Todas as classes persistentes implementam uma interface chamada `Persistent`. O gerador de aspectos disponibilizará uma interface gráfica para que seja definido as classes persistentes

da aplicação, além de fazer um mapeamento da classe persistente com a tabela correspondente no modelo de dados. O seguinte aspecto abstrato é usado para capturar conjuntos de junção essenciais à sincronização de objetos:

```
public abstract aspect SynchronizationControlAspect {
    private Set dirtyEntities = new HashSet();
    abstract pointcut insert(Persistent p);
    abstract pointcut update(Persistent p);
    pointcut executeUpdate(): TransactionControlAspect.aspecOf().
                                                transactionalMethods;
    after(): insert(Persistent p) {p.insertEntity(); }
    after(): update(Persistent p) {dirtyEntitys.add(p); }
    after(): executeUpdate() {
      Iterator iterator = dirtyEntities.iterator();
      while(iterator.hasNext()) {
        Persistent p = (Persistent) iterator.next();
        try {p.updateEntity();}finally{iterator.remove();}
      }
    }
}
```

Os conjuntos de junção abstratos `insert` e `update` definem os pontos da aplicação alvo que devem ser instrumentados com código para, respectivamente, inserir uma nova linha em uma determinada tabela do banco de dados e para atualização de uma linha de uma tabela. No primeiro caso, tais pontos normalmente correspondem a instanciações, via operador `new`, de objetos de classes persistentes. O segundo caso corresponde a chamadas de métodos de nome `set*` de classes persistentes.

Ao conjunto de junção `insert` associa-se um comportamento transversal do tipo `after` a qual simplesmente invoca um método de nome `insertEntity` tendo como alvo o objeto que foi instanciado. A implementação deste método é gerada automaticamente pelo sistema e é adicionada à classe do respectivo objeto persistente (usando-se os recursos de declaração inter-tipos de AspectJ). Esta implementação contém o código responsável por emitir um comando SQL para inserção de linhas em uma tabela de um banco de dados relacional.

Ao conjunto de junção `update` associa-se um comportamento transversal do tipo `after` a qual simplesmente adiciona o objeto a um conjunto de objetos que ainda não foram sincronizados com o banco de dados. Este conjunto é chamado de `dirtyEntities`. Após a execução de métodos transacionais (comportamento transversal `executeUpdate`), este conjunto é percorrido invocando-se o método `updateEntity` de cada um de seus elementos. A implementação deste método é gerada automaticamente pelo sistema e é adicionada à classe do respectivo objeto persistente. Esta implementação contém o código responsável por emitir um comando SQL para atualização de uma linha em uma tabela de um banco de dados relacional.

Para cada classe persistente da aplicação, o gerador de aspectos gera um aspecto que estende `SynchronizationControlAspect` e que define seus conjuntos de junção abstratos. Além disso, os aspectos gerados implementam os métodos `insertEntity` e `updateEntity`. Caso o arcabouço não incluísse um gerador de aspectos, estes subaspectos deveriam ser implementados de forma manual, em um processo tedioso, que consumiria um tempo considerável e sujeito a erros.

3. Estudo de Caso: Sistema de Comércio Eletrônico

Esta Seção descreve um estudo de caso baseado em um sistema de comércio eletrônico que emprega o arcabouço. Esse sistema contém funcionalidades como cadastramento e controle de produtos, cadastramento de clientes e geração de pedidos. Os requisitos funcionais identificados foram implementados usando orientação por objetos pura (Java), e os transversais como controle de conexão, controle de transação e sincronização dos objetos foram implementados usando o arcabouço. Analisando esse tipo de sistema, verifica-se que as entidades básicas são Cliente, Produto, Pedido e Item de Pedido. A Figura 1 apresenta o modelo de dados para tais entidades.

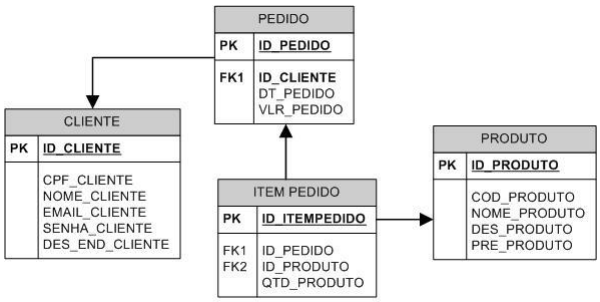

Figura 1. Modelo de Dados do Sistema de Comércio Eletrônico

A partir do modelo acima foi criado um banco de dados no SGBD MySQL. Como mencionado anteriormente o uso do arcabouço requer algumas configurações iniciais para que os aspectos que controlam a persistência possam ser gerados. A primeira configuração a ser feita consiste em informar dados como URL do banco de dados, nome do banco, usuário, senha e drive, para que extração do modelo do banco possa ser feita. A Figura 2 mostra a interface gráfica da extração do modelo de dados.

Figura 2. Geração do Modelo

Concluída a primeira configuração o usuário solicita a extração do modelo, a ferramenta gera e o armazena em arquivo modelo.xml. Para a próxima etapa de configuração é necessário definir os pontos do sistema que necessitam de controle de conexão, bem como os métodos que precisam de controle de transação, ou seja, métodos transacionais. Para o sistema de comércio eletrônico estes pontos são definidos como mostra a Figura 3.

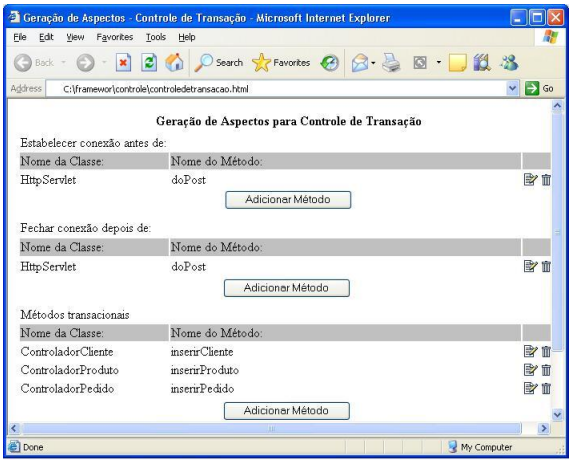

Figura 3. Controle de Conexão e Transação

Após essa configuração já é possível a geração dos aspectos para controlar a conexão e transação. Para a entidade Cliente os seguintes conjuntos de junções foram especializados.

```
pointcut openConnection():execution(* ClienteServlet.doPost(..));
pointcut closeConnection():execution(* ClienteServlet.doPost(..));
pointcut transactionalMethods():execution(*
                          ControladorCliente.inserirCliente(..));
```

Por último é necessário definir os pontos do sistema que precisam de sincronização dos objetos. Para esse sistema foram definidos que as classes `Cliente`, `Produto`, `Pedido` e `ItemPedido` são classes tidas como persistentes, ou seja, classes que implementam uma interface `Persistent` do arcabouço. Portanto é feito o mapeamento entre as classes persistentes e as entidades do banco de dados. Além disso é feito também uma mapeamento entre os atributos das classes e os campos das tabela no banco de dados. A Figura 4 ilustra esse processo.

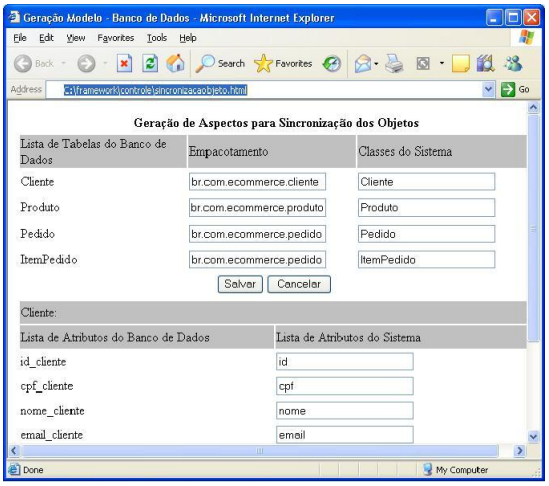

Figura 4. Sincronização de Objetos

Após essas configurações os aspectos para a sincronização dos objetos são gerados. Para a entidade Cliente o aspecto abstrato `PersistenceControlAspect` foi

especializado pelo aspecto `ClienteAspect` definido abaixo.

```
public aspect ClienteAspect extends PersistenceControlAspect {
    declare parents: Cliente implements Persistent;
    pointcut insert(Persistent p) (execution(Cliente.new(..))
                                   && target(p));
    pointcut update(Persistent p): (execution(Cliente.set*(..))
                                   && target(p));
    public void Cliente.insertEntity() throws Exception {
        Persistente obj = null;
        IClienteDAO dao = (IClienteDAO) DAOFactory.getDAO(
                            Constantes.CLASS_NAME_CLIENTE_DAO);
        dao.insert(this);
    }
    public void Cliente.updateEntity() throws Exception {
        IClienteDAO dao = (IClienteDAO) DAOFactory.getDAO(
                             Constantes.CLASS_NAME_CLIENTE_DAO);
        dao.update(this);
    }
}
```

Como descrito na Seção 2 quando um objeto que implementa a interface `Persistent` é criado dentro de um método transacional este objeto é inserido no banco automaticamente através do método `insertEntity` e quando um objeto que implementa essa mesma interface invoca um método set dentro de um método transacional, esse objeto é adicionado a lista de objetos alterados que posteriormente serão atualizados no banco através do método `updateEntity`.

4. Trabalhos Relacionados

Soares, Borba e Laureano [5] propõem um arcabouço para implementação de distribuição e persistência usando AspectJ. Descrevem ainda como este arcabouço foi utilizado para reestruturar uma aplicação *Web* denominada Health Watcher. O arcabouço proposto, no entanto, é voltado para aplicações que possuam a mesma arquitetura do sistema Health Watcher. Além disso, a geração dos aspectos de persistência é feita de forma manual, sem suporte de uma ferramenta de geração código. Os próprios autores reconhecem que seria bastante útil possuir uma ferramenta deste tipo.

Rashid e Chitchyan [4] propõe um projeto inicial de arcabouço de persistência baseado em aspectos, com o objetivo de introduzir o requisito de persistência em um sistema. Os autores criaram as partes da persistência que são independentes do sistema por meio da criação de pontos de junção abstratos que devem ser concretizados por aspectos especializados. A principal desvantagem deste artigo em relação ao descrito aqui é que o mesmo não possui uma ferramenta de geração de aspectos para especialização dos aspectos abstratos. Além disso, os pontos de inserção e atualização são os métodos `new` e `set` de uma classe persistente, como também é feito nesse arcabouço, porém a construção dos métodos de inserção e atualização fica a cargo do programador, diferentemente do que é feito no arcabouço onde estes métodos são gerados automaticamente pelo gerador a partir do modelo de dados.

Por último Camargo e Masiero [1] descrevem um padrão de projeto baseado em aspectos para a camada de persistência. O padrão proposto utiliza como base o padrão orientado por objetos de camada de persistência descrito por Yoder em [7]. Quando aplicado de forma manual, este padrão é útil apenas em pequenos sistemas, mas para sistema maiores torna-se tediosa a criação dos aspectos para cada classe tida como sendo persistente.

5. Conclusões

A implementação de requisitos como conexão, controle de transação e controle de persistência em linguagens orientadas por objetos leva a dois problemas conhecidos como entrelaçamento e dispersão do código. Este artigo apresentou um arcabouço para resolução desses problemas. O arcabouço proposto apresenta as seguintes vantagens:

- Geração automática dos aspectos para controle de conexão e transação e sincronização de objetos.
- Instanciação automática dos aspectos abstratos que compõem o arcabouço.
- Geração automática dos métodos para inserção e atualização de registros no banco de dados.

O estudo de caso realizado mostrou que o arcabouço é de fácil utilização pois são necessárias somente algumas configurações iniciais, além disso o sistema atende a diversas classes e arquitetura de sistemas, já que permite a configuração de qualquer classe e método que necessite fazer uso do serviço de persistência (isto é, que precise abrir e fechar conexões, controlar transações e sincronizar objetos).

A principal desvantagem do arcabouço é o fato de não disponibilizar recursos para a recuperação automática de dados do banco. O arcabouço não constrói nenhuma consulta SQL para que a recuperação de dados possa ser feita. Assim, cabe ao programador implementar tais consultas. Além disso o arcabouço não utiliza o padrão de projeto camada de persistência. Pretende-se eliminar tais desvantagens em trabalhos futuros.

Referências

[1] Valter V. de Camargo, Ricardo A. Ramos, Rosângela Penteado, and Paulo C. Masiero. Projeto baseada em aspectos do padrão camada de persistência. In *XVII Simpósio Brasileiro de Engenharia de Software*, 2003.

[2] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In *Proceedings of the 15th European Conference on Object-Oriented Programming*, page 220ff. Springer-Verlag, 2001.

[3] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming*, page 220ff. Springer-Verlag, 1997.

[4] Awais Rashid and Ruzanna Chitchyan. Persistence as an aspect. In *Proceedings of the 2nd Aspect-Oriented Software Development*, pages 120–129. Springer-Verlag, 2003.

[5] Sérgio Soares, Eduardo Laureano, and Paulo Borba. Implementing distribution and persistence aspects with aspectj. In *Proceedings of the 17th ACM SIGPLAN conference on Object-Oriented Programming, systems, languages and applications*, pages 174–190. ACM Press, 2002.

[6] Fabio Tirelo, Mariza A. da S. Bigonha Roberto da S. Bigonha, and Marco T. de O. Valente. Desenvolvimento de software orientado por aspectos. In *XIII Jornada de Atualização em Informática*. Sociedade Brasileira de Computação, 2004.

[7] Yoder J. W., Johnson R. E., and Wilson Q. D. Connecting business objects to relational databases. In *Conference on the Pattern Languages of Programs*, 1998.