Extensão Orientada a Aspectos para Interações Multi-participantes Confiáveis*

Luciano Azevedo Cassol[1], **Avelino Francisco Zorzo**[2], **André Zampieri**[1]

[1]Departamento de Informática – Universidade de Caxias do Sul (UCS)
Rua Francisco Getúlio Vargas, 1130 – 95001-970 Caxias do Sul - RS

[2]Faculdade de Informática – Pontifícia Universidade Católica do RS (PUCRS)
Av. Ipiranga, 6681 – 90619-900 Porto Alegre - RS

lacassol@ucs.br, zorzo@inf.pucrs.br, azampier@ucs.br

Abstract. *The use of mechanisms of fault tolerance in the development of systems is more and more frequent. A problem that can happen in these mechanism is the tangling among the fault tolerance mechanism and the system in development, as well as the fault tolerance code scattering in several points of the system. The use of aspect oriented programming can aid in the task of separating multi-dimensional interests in the software development. This work presents an extension of the framework for dependable multiparty interactions through the use of aspect oriented programming with the objective of obtaining transparency in its use in system development.*

1. Introdução

No ciclo de desenvolvimento de *software*, o tratamento de exceções deve ser pensando em toda a sua estrutura. É necessário saber o que o sistema deve e o que o sistema não deve fazer. Normalmente essa análise requer um esforço muito grande por parte dos projetistas de *software*. Identificar o que é normal e o que é excepcional é um processo manual e passível de erros que podem gerar situações catastróficas.

O tratamento de exceções na maioria das linguagens de programação requer uma disciplina rígida por parte dos programadores e a realidade é que os *software* que são desenvolvidos hoje usam linguagens de programação que preveêm pouco suporte ao tratamento de exceções.

Há um efeito colateral ao programar tratamento de exceções na aplicação, o entrelaçamento entre o código do comportamento normal do sistema e o código para descoberta e tratamento de exceções [Lippert and Lopes 2000]. O entrelaçamento é uma conseqüência tanto da linguagem de programação quanto decisões de projeto.

O uso da programação orientada a aspectos vem para auxiliar o projeto de verificação de situações anormais, uma vez que o modelo de programação orientada a aspectos preocupa-se com aspectos transversais da aplicação [Kiczales et al. 1997].

O objetivo principal do desenvolvimento de *software* orientado a aspectos é auxiliar na tarefa de separar interesses multi-dimensionais, usando mecanismos de abstração e composição permitindo especificar e combinar os diferentes módulos de uma aplicação [Kiczales et al. 1997].

*Trabalho parcialmente financiado pelo CNPQ - Chamada CT-INFO: CNPq 31/2004 - PDPG-TI.

O trabalho de [Lippert and Lopes 2000] apresenta um estudo sobre o uso de programação orientada a aspectos para diminuir o entrelaçamento do controle e tratamento de exceções em Java. Outros trabalhos como [Simons and Stafford 2004] e [Silveira and Weber 2005] também utilizam a programação orientada a aspectos como uma forma de não espalhar o código na aplicação cliente para diferentes domínios.

O presente trabalho apresenta uma extensão orientada a aspectos do *framework* de interações multi-participantes confiáveis (DMI) que possibilite uma transparência no uso desse *framework* por aplicações clientes. Este trabalho está organizado da seguinte forma: a Seção 2 apresenta as interações multi-participantes confiáveis; a Seção 3 apresenta a extensão orientada a aspectos proposta; e, a Seção 4 apresenta as conclusões sobre esse trabalho.

2. Interações Multi-participantes Confiáveis Orientadas a Aspectos

Interações Multi-participantes Confiáveis (DMI) é uma abstração de controle geral que é usada para uma interação entre diversos participantes e fornece instrumentos para o tratamento de exceções concorrentes. Tem como função auxiliar na construção de atividades concorrentes complexas e auxiliar na recuperação de erros em sistemas onde existe cooperação entre as atividades concorrentes [Zorzo and Stroud 1999].

Uma DMI é um tipo de interação multi-participante que fornece instrumentos para tratamento de exceções concorrentes: quando uma exceção ocorre em um dos participantes da interação, mas não é tratada por ele, a exceção deve ser propagada para todos os participantes da interação. Uma DMI também assegura a consistência na saída: um participante somente deixa sua interação quando todos tiverem terminado suas funções e os dados que são acessados competitivamente por diversas interações estiverem em um estado consistente.

O desenvolvimento de sistemas que utilizem o *framework* de DMI necessita uma arquitetura de projeto de *software* estruturada para utilizar o *framework*. Esse tipo de característica faz com que o desenvolvimento do sistema tenha que ser baseado no uso do *framework*, tornando essa decisão uma decisão da fase de projeto do *software*.

Essa forma de construção de aplicações que utilizem o *framework* de DMI ocasiona um conjunto de restrições que muitas vezes podem ocasionar problemas para utilizar o *framework* de DMI. Entre essas restrições é importante ressaltar o entrelaçamento de código no momento da criação da DMI.

O código da Figura 1 mostra como a aplicação cliente deve criar uma DMI. Na linha cinco da classe `Role1` é possível visualizar que a classe `Role1` tem que estender a classe `RoleImpl` do framework de DMI. Na linha dois da classe `ExampleNR` é possível observar a criação da referência à classe `Role` pertencente ao *framework*. Entre as linhas quatro e dez ocorre a criação dos objetos relativos as classes do *framework*. Esse entrelaçamento, que é uma forma de dependência, causaria transtornos em uma eventual troca de *framework* para tratamento de exceções.

Um dos principais pontos de entrelaçamento entre aplicação cliente e o *framework*, é a obrigatoriedade que a aplicação cliente tem de extender determinadas classes do *framework* e re-implementar o comportamento de alguns métodos. Essa obrigatoriedade acarreta uma rigidez na arquitetura do *software*. Essa rigidez dificulta a utilização

de *frameworks* que não possibilitem uma flexibilidade no modelo de implementação.

```
1. import java.rmi.RemoteException;
2. import drip.Manager;
3. import drip.RoleImpl;
4.
5. public class Role1 extends RoleImpl {
6.     public Role1(String n, Manager mgr, Manager leader)
         throws RemoteException{
7.      super (mgr, leader, n);
8.   }
9.
10.   public void body(Object list[])
        throws Exception, RemoteException {
11.      try{
12.          System.out.println("Role 1 ...");
13.      } catch (Exception e) {
14.          throw e;
15.      }
16.   }
17.}
```

```
1. public class ExampleNR implements Runnable {
2.   Role role1, role2, role3;
3.   public ExampleNR() throws Exception {
4.     Manager leader = new ManagerImpl("leader","DMI");
5.     Manager mgr2   = new ManagerImpl("mgr2","DMI");
6.     Manager mgr3   = new ManagerImpl("mgr3","DMI");
7.
8.     role1 = new Role1 ("Role1", leader, leader);
9.     role2 = new Role2 ("Role2", mgr2, leader);
10.    role3 = new Role3 ("Role3", mgr3, leader);
11.  }
12.  public void run() {
14.  }
12.}
```

Figura 1. Exemplo de código com o framework DMI

Uma solução que transformasse o *framework* em um componente transparente, ao ponto de poder ser ligado e desligado quando necessário, resolve esse problema e facilita a aderência do uso de DMI em aplicações legadas, bem como o desenvolvimento de novas aplicações deixando a arquitetura do *software* da aplicação cliente independente do uso ou não do *framework* de DMI. O desenvolvimento de *software* orientado a aspectos (DSOA) visa resolver os problemas de entrelaçamento e espalhamento de código [Kiczales et al. 1997].

3. Extensão orientada a aspectos do *framework* de DMI

A nova estrutura orientada a aspectos para o *framework* de DMI é baseada numa estrutura onde um conjunto de aspectos estão separados do código funcional dos objetos. Uma arquitetura reflexiva é usada para introduzir dois níveis diferentes de execução:

- Funcional: as funcionalidades do modelo orientado a objetos estão definidas nesse nível. Foram acrescentados um conjunto de *annotations* (Java 5) na estrutura original do *framework*. Esse componente tem como função ”anotar”o código de modo que um conjunto de aspectos possam interceptar esse trecho de código e interagir com a aplicação cliente.
- Aspectos: os aspectos são definidos nesse nível. Há um conjunto de aspectos que são responsáveis por retirarem o entrelaçamento de código entre o *framework* e aplicação cliente mas também são responsáveis por proverem as funcionalidades do *framework* para a aplicação cliente.

A Figura 2 apresenta a arquitetura proposta para o novo *framework* onde é possível visualizar uma separação clara ente os componentes da aplicação cliente e os componentes da estrutura de DMI. Um conjunto de aspectos faz o relacionamento entre essas duas estruturas. Esses aspectos não estão inseridos no código da aplicação cliente, evitando assim um acoplamento entre essas estruturas.

O código da Figura 3 mostra como uma determinada classe pode utilizar a nova estrutura do framework de DMI para redução do entrelaçamento de código entre os componentes. A linha 2 apresenta uma *annotation* com informações para os aspectos capturarem a execução dessa classe. Essa *annotation* é o único ponto de entrelaçamento entre aplicação cliente e o *framework* de DMI.

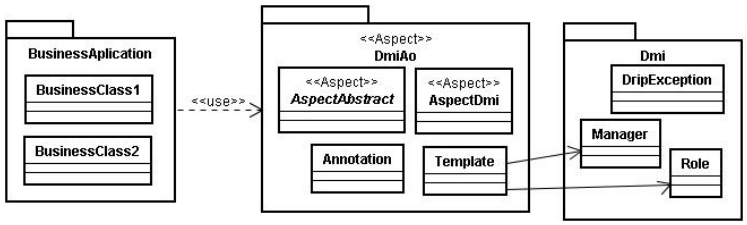

Figura 2. Modelo da Arquitetura do framework DMI

```
1. public class BusinessClass{
2.   @Dmi(name="DMI", number=2,exception="BusinessException", handlerException="BusinessHandler",
3.       role="BusinessClass,BusinessRole", parameter="null, null", init = true)
4.   public BusinessClass() {
5.       System.out.println("Criação da estrutura
6.       de relacionamentos entre managers e roles"); }
7.   @Role(0)
8.   public void methodExecution(){
9.       //codigo da role }
10.}
```

Figura 3. Exemplo de código com o framework DMI orientado a Aspectos

4. Conclusão

O presente trabalho apresentou uma extensão orientada a aspectos para um *framework* de tratamento de exceções concorrentes. O objetivo desse trabalho foi de extender o *framework* existente de modo que fosse possível retirar o acoplamento do mesmo com aplicações clientes e ao mesmo tempo manter a compatibilidade com versões anteriores do mesmo.

Esse trabalho está na fase de testes e validação da nova arquitetura do *framework* de DMI. Essa validação está sendo feita através da refatoração de controladores de células de produção [Zorzo and Stroud 1999] para a nova arquitetura. Após a finalização da fase de validação será possível fazer uma análise dos resultados obtidos com a arquitetura de *software* orientada a aspectos para *framework* de tratamento de exceções.

Referências

[Kiczales et al. 1997] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. (1997). Aspect-oriented programming. In Akşit, M. and Matsuoka, S., editors, *11th Europeen Conf. Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer Verlag.

[Lippert and Lopes 2000] Lippert, M. and Lopes, C. V. (2000). A study on exception detecton and handling using aspect-oriented programming. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 418–427. ACM Press.

[Silveira and Weber 2005] Silveira, K. K. and Weber, T. S. (2005). Um injetor de falhas de comunicação construído usando programação orientada a aspectos. In *VI Workshop de Testes e Tolerância a Falhas*, Fortaleza, CE, Brasil.

[Simons and Stafford 2004] Simons, K. and Stafford, J. A. (2004). Cmeh: Container managed exception handling for increased assembly robustness. In *CBSE*, pages 122–129.

[Zorzo and Stroud 1999] Zorzo, A. F. and Stroud, R. J. (1999). A distributed object-oriented framework for dependable multiparty interactions. In *OOPSLA Conference Proceedings*, volume 34, pages 435–446, Denver, Colorado - USA. ACM, Inc.