Universidade de São Paulo
Instituto de Matemática e Estatística

MAC 5701 Tópicos em Ciências da Computação

Plano de Estudos:
Programação Orientada a Aspectos Dinâmica

Flávia Rainone
fla@ime.usp.br

Orientador: Francisco Reverbel
reverbel@ime.usp.br

São Paulo, 24 de abril de 2005

1.Introdução

Esse documento descreve o plano de estudos a ser realizado no primeiro semestre de 2005, como parte da disciplina MAC5701 – Tópicos em Ciências da Computação. O tópico a ser estudado é Programação Orientada a Aspectos Dinâmica. Essa pesquisa será utilizada como embasamento para a minha dissertação de mestrado.

2.Motivação

A programação orientada a aspectos ou POA permite o encapsulamento de funcionalidades ortogonais a um sistema.

Funcionalidade ortogonal é aquela que deve ser aplicada a vários pontos de uma mesma aplicação. Para que essa funcionalidade seja implementada corretamente, é necessário que uma certa ação seja executada em diversos momentos da execução de tal sistema. Esse tipo de funcionalidade não é facilmente implementada utilizando-se mecanismos convencionais. Compare esse tipo de funcionalidade com as chamadas funcionalidades locais [2]. Essas funcionalidades são as que devem ser aplicadas a um ponto específico do sistema. Basta inseri-las no ponto correto do código que elas estarão implementadas corretamente. Com funcionalidades ortogonais, isso não é o suficiente, pois elas têm que ser aplicadas a mais de um ponto. Suponha que estamos utilizando uma linguagem orientada a objetos. Então, podemos encapsular uma funcionalidade ortogonal em uma classe *A*. Ainda assim, teremos que invocar métodos da classe *A* em diversas classes do sistema, para que a funcionalidade que *A* contém seja aplicada aos pontos adequados. Desse modo, uma classe *B*, que encapsula uma funcionalidade local, terá que invocar métodos de *A*, o que nada tem a ver com o seu papel no sistema. Isso prejudica o encapsulamento das funcionalidades. Além desse problema, temos um acoplamento indesejável entre *A* e as outras classes. Se a funcionalidade ortogonal contida em *A* tiver que ser alterada, podemos ter que alterar outras classes, dependentes de *A*. Além disso, se os pontos aos quais tal funcionalidade deve ser aplicada forem alterados, teremos que alterar o código de várias classes, para que a invocação dos métodos de *A* satisfaçam os novos requisitos. Alguns exemplos de funcionalidade ortogonal são segurança, transações e *log*.

Ao utilizar POA, o programador implementa uma funcionalidade ortogonal em um componente separado (uma classe, uma biblioteca, ou outra unidade de encapsulamento; isso depende da linguagem utilizada) e especifica em quais pontos do sistema essa funcionalidade deverá ser aplicada. Desse modo, quando o sistema entrar em execução, a funcionalidade em questão será executada em todos os pontos especificados.

A programação orientada a aspectos nos permite encapsular completamente uma funcionalidade ortogonal (note que classes como *B* não terão mais que invocar métodos de *A*, no exemplo dado). Esse encapsulamento elimina os problemas citados e facilita a

atualização do código. Para mudar os pontos do sistema nos quais a funcionalidade deverá ser aplicada, basta fazer uma reconfiguração de tal funcionalidade, o que em muitas ferramentas de POA não implica em alteração de código. Além disso, para atualizar a implementação de uma funcionalidade ortogonal, basta alterar o componente que a encapsula, sem necessidade de alterar código em vários pontos do sistema.

A programação orientada a aspectos dinâmica permite aplicar funcionalidades ortogonais a pontos de um sistema em execução. Uma possível utilização de tal facilidade ocorre quando um grande sistema em produção apresenta um *bug* em seu funcionamento. Se esse sistema não puder sair do ar e for necessário que a causa do erro seja descoberta, podemos querer ativar o *log* em um ou mais pontos durante a execução, para que a causa do erro seja detectada. A ativação do *log* pode ser feita através de programação orientada a aspectos dinâmica, adicionando-se a funcionalidade ortogonal de *log* aos pontos adequados. Com POA dinâmica, esse procedimento pode ser aplicado a um sistema em tempo de execução, sem que ele tenha que ser compilado novamente.

3.Plano de Trabalho

Existem algumas implementações de programação orientada a aspectos dinâmica em Java. O objetivo do estudo será analisá-las, avaliando as vantagens e desvantagens de cada uma. Seguem os tópicos que serão estudados:

- as possibilidades existentes de implementação de programação orientada a aspectos dinâmica em Java;
- estudo de caso em algumas ferramentas de POA, como JBoss AOP, AspectWerkz e PROSE.

Além desses tópicos, o trabalho incluirá a implementação de POA dinâmica no JBoss AOP utilizando agentes de JVMTI (para saber mais sobre essa interface de programação, consulte [10]).

4.Referências

[1] Robert E. Filman, Daniel P. Friedman, *Aspect-Oriented Programming is Quantification and Obliviousness*

[2] Robert E. Filman, *What is Aspect Oriented Programming, Revisited*, maio de 2001.

[3] Robert E. Filman, Michael Haupt, Katharina Mehner, Mira Mezini *Proceedings of the 2004 Dynamic Aspect Workshop (DAW04)*

[4] Andrei Popovici, Thomas Gross, e Gustavo Alonso, *Dynamic Weaving for Aspect-Oriented Programming*

[5] M. Pinto, L. Fuentes, M. E. Fayad, e J. M Troya, *Separation of Coordination in a Dynamic Aspect Oriented Framework*

[6]Jonas Bonér, *AspectWerkz – Dynamic AOP for Java*
[7]Ruzanna Chitchyan, *Comparing Dynamic AO Systems*
[8]JBoss AOP, http://www.jboss.org/products/aop
[9]AspectWerkz, http://aspectwerkz.codehaus.org/
[10]PROSE http://prose.ethz.ch
[11]JVM Tool Interface, http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/jvmti.html