1 Introdução

Com o crescimento, nas últimas décadas, da complexidade do código das aplicações e da dificuldade de gerenciá-lo, fez-se necessário o desenvolvimento de novas técnicas para estruturação do código e, dentre essas, destaca-se a técnica de orientação a aspectos.

A Programação Orientada a Aspectos (POA) (KICZALES, 1996; KICZALES et alii.1997) é um forma bem difundida de composição de interesses transversais e tem por objetivo modularizar interesses que atravessem os elementos básicos de um sistema de software (i.e. classes e métodos) através de uma nova abstração chamada aspecto. O aspecto utiliza construções nomeadas advices para incluir comportamento adicional em pontos específicos do código, que são chamados de join points que podem ser exemplificados como sendo a chamada ou execução de um método e a inicialização de objeto, por exemplo. Pointcuts, por sua vez, são construções definidas dentro dos aspectos que ajudam a identificar esses join points no código – essas construções são representadas por expressões que, sintaticamente, correspondem a um conjunto específico de join points. Recentemente, surgiram várias técnicas e frameworks orientados a aspectos, tais como AspectJ (KICZALES, 1996), CaesarJ (MEZINI e OSTERMANN, 2003), Spring AOP (SPRING, 2009) e JBoss AOP (JBOSSAOP, 2009). Este último pode ser descrito como sendo uma implementação do paradigma de orientação a aspectos, desenvolvida pela JBoss, e é utilizado principalmente em seu servidor de aplicação denominado JBoss AS, onde são acessados alguns de seus serviços como cache, comunicação assíncrona, transações, segurança.

Já os servidores de aplicação podem ser definidos como sendo plataformas de middleware para desenvolvimento e implantação de componentes de software (FLEURY e REVERBEL, 2003). Esses componentes podem ser tanto uma aplicação quanto serviços a serem utilizados por uma aplicação, sendo que a disponibilidade desses serviços no servidor pode variar dependendo do fabricante.

O JBoss AS (JBOSSAS, 2009), por exemplo, é um servidor de aplicação fornecido pela JBoss e provê os seguintes serviços: segurança, distribuição, cache, transação, persistência, mensagens, serviços web, administração, processamento de requisições web, clusterização e ainda é flexível o suficiente para permitir a adição de novos serviços desenvolvidos pelo usuário, sendo, por essas características, um dos servidores opensource mais utilizados para desenvolvimento de aplicações corporativas.

O tratamento de exceções é um mecanismo que visa melhorar a robustez da aplicação e há várias formas de realizá-lo, como, por exemplo, por sub-rotinas, que são rótulos para uso com GOTO e variáveis de estado, mas que ao mesmo tempo, os usos dessas formas de tratamento de exceção possuem severas consequências negativas (GOODENOUGH, 1975b). Desta forma, visando melhorar o gerenciamento de erros em uma aplicação, foi introduzido o conceito de exceção (GOODENOUGH, 1975a) que é utilizado para generalizar operações de sinalização e tratamento do fluxo excepcional da aplicação, permitindo lidar com falhas nas operações desta tais como erros de domínio e erros de limites. Outra função da exceção é, ainda, a de indicar a validade de um resultado ou as circunstâncias em que ele foi obtido. Contudo, características como polimorfismo e herança em conjunto com o tratamento de exceção ainda constituem fontes de problemas (MILLER e TRIPHATHI, 1997).

1.1.O Problema

Com o advento da orientação a aspectos que permite que códigos com interesses diferentes sejam fundidos de forma a adicionar novas funcionalidades a uma aplicação, aumentou a complexidade com relação ao tratamento de exceção, pois as novas funcionalidades que são adicionadas pelos aspectos tanto podem incluir novas exceções à aplicação, quanto podem capturar as que ocorram nos fluxos interceptados por aqueles. Desta forma, adicionar aspectos a aplicações pode levar a cenários inconsistentes (COELHO et alii, 2008) que podem, por sua vez, aflorar mais facilmente quando os aspectos que interceptarem a aplicação fizerem parte de um framework, pois nem o framework conhece a aplicação interceptada por ele e nem a aplicação o conhece.

Além disso, somente a ferramenta SAFE (COELHO, 2008) aborda o problema de analisar programas orientados a aspectos, apesar da ferramenta não levar em consideração aspectos desenvolvidos com o JBoss AOP. Este, por sua vez, é utilizado principalmente no servidor de aplicações JBoss AS para prover algumas das funcionalidades do framework EJB¹ (Enterprise Java Beans), plataforma de desenvolvimento Java mais utilizada para o desenvolvimento de aplicações corporativas. Desta forma, a maioria dos desenvolvedores utiliza indiretamente aspectos no seu dia-a-dia sem conhecer quais as consequências principalmente relativas ao tratamento de exceção.

1.2. Objetivos

Dado que os efeitos da composição de aspectos JBoss AOP com a aplicação não são muito compreendidos com relação ao controle do fluxo de exceção, temos por objetivo analisar os cenários em que estas ocorrem e, assim, obter dados de forma a ser possível detectar padrões que indiquem defeitos. A análise desses cenários realizada de forma manual não é razoável (ROBILLARD e MURPHY, 1999), principalmente para sistemas de médio e grande porte, sendo obrigatório, portanto, o uso de ferramentas que reduzam a quantidade de cenários a se analisar.

Desta forma, estendemos a ferramenta SAFE para JBoss AOP para que a mesma calcule automaticamente os caminhos de exceção (isto é, o caminho em um grafo de chamadas de uma aplicação que se inicia no sinalizador e termina no manipulador da exceção sinalizada) nesse último sistema. Assim, a ferramenta deve fornecer uma melhor compreensão do fluxo de exceções em aplicativos JBoss AOP e, também, identificar possíveis falhas no código de tratamento de exceções como exceções não-capturadas (MILLER E TRIPATHI, 1997; JO et alii, 2004) e exceções capturadas por subsunção (ROBILLARD e MURPHY, 1999). O uso da implementação do paradigma de orientação a aspectos JBoss AOP, é comum no servidor de aplicação JBoss AS, mais especificamente ele é utilizado para implementar a separação de interesses do container EJB.

Portanto, dando continuidade ao estudo realizado por Coelho (2008), o objetivo desta dissertação é:

¹ EJB é uma especificação desenvolvida pela Sun microsystems (ver detalhes na Seção 2.3.1)

(i) Realizar um estudo exploratório, a fim de investigar os efeitos provocados por aspectos do fluxo de exceção de programas incluindo os fluxos que ocorrem nas bibliotecas utilizadas pelos programas, mais especificamente em aplicações que utilizem o framework EJB e o servidor de aplicações o JBoss AS;

(ii) Construir uma ferramenta de análise estática para fornecer dados sobre o fluxo de exceções em programas AOP, mais especificamente para JBoss AOP, de forma a permitir que os desenvolvedores e pesquisadores analisem suas aplicações, e obtenham uma fotografia do estado atual do tratamento de exceção da aplicação analisada.

1.3.Estrutura do Documento

A presente dissertação divide-se em seis capítulos, nos quais serão abordados tanto aspectos teóricos quanto comparações entre trabalhos que abordem o mesmo tema. No Capítulo 2 será apresentada uma revisão de conceitos básicos relativos (i) à orientação a aspectos, mais especificamente em relação ao JBoss AOP; (ii) aos servidores de aplicação ; (iii) ao framework EJB; e (iv) à ferramenta SAFE na qual este trabalho foi baseado.

No Capítulo 3 serão apresentadas a arquitetura e heurísticas da ferramenta, bem como as decisões de design que delinearão a construção da mesma. Já no Capítulo 4, buscar-se-á descrever o estudo empírico realizado neste trabalho, que tem por objetivo avaliar o impacto dos aspectos sobre os fluxos de exceção dos programas que utilizem o JBoss AOP.

No Capítulo 5 apresentar-se-ão trabalhos relacionados a este, de forma a fazer uma comparação entre as abordagens adotadas e suas vantagens e desvantagens em relação ao presente estudo. Por fim, as conclusões e as propostas de trabalhos futuros serão apresentadas no Capítulo 6.