2
Estado da Arte e Trabalhos Relacionados

Neste capítulo, serão apresentados alguns conceitos relativos a frameworks orientado a objetos, e o paradigma da programação orientada a aspectos. Além disso, serão apresentados alguns trabalhos desenvolvidos que abordam a utilização da tecnologia de aspectos em conjunto com frameworks para o desenvolvimento de software.

2.1.
Frameworks Orientados a Objetos

A engenharia de software é uma disciplina que visa auxiliar o desenvolvimento de software, através de técnicas, métodos e ferramentas, para promover a diminuição no esforço do desenvolvimento de software e aumentar a sua qualidade. O reuso de software é um dos pilares da engenharia de software, pois, através dele é possível reaproveitar artefatos de software de qualidade anteriormente desenvolvidos, testados e validados com sucesso e posteriormente aplicá-los em novos de projetos de software. Além disso, a reutilização de software permite que o desenvolvimento seja realizado com menos esforço por parte dos desenvolvedores, assim, diminuindo o tempo de entrega do software [1].

O projeto de software não é uma tarefa trivial, projetar software para ser reutilizado é uma tarefa menos trivial ainda [36]. Diversas abordagens para o desenvolvimento de software que buscam a reutilização são conhecidas, dentre as quais podemos citar: padrões de projetos [33], biblioteca de classes [34] e componentes de software [34].

Outra abordagem que visa a reutilização de software é obtida pelo uso da tecnologia de framework [31] que será a abordagem utilizada para estudo nesta dissertação. Várias definições para framework são encontradas na literatura, dentre as quais podemos citar:

> "Um framework é um conjunto de classes que constitui um design abstrato para soluções de uma família de problemas." [35]

> "Uma arquitetura desenvolvida com o objetivo de se obter a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com grande potencial de especialização [36]"

> "Um software parcialmente completo projetado para ser instanciado. O framework define uma arquitetura para uma família de subsistemas e oferece os construtores básicos para criá-los. Os pontos de extensão (*hot-spots*) são explicitados, nos quais adaptações de código para funcionamento especifíco de certos módulos devem ser feitas." [14]

Um framework não é um software executável. Para gerar uma aplicação completa e executável deve-se instanciar[4] o framework definindo o código especifico da aplicação para cada *hot-spot*. Como apresentado em uma das definições de framework, os *hot-spots* representam os pontos flexíveis de um framework, sendo representados por classes ou métodos abstratos [14]. Além disso, os frameworks apresentam partes que não variam (fixas) em sua estrutura que são os chamados *frozen-spots* que constituem o núcleo de um framework e que sempre estão presentes em cada instância (aplicação) do framework [31].

A estrutura de um framework é representada na Figura 1, onde são explicitados o núcleo (*frozen-spots*) e os pontos de extensão (*hot-spots*) que constituem um framework, além das implementações (instanciações) dos *hot-spots*.

[4] Processo de criação de uma aplicação baseada em frameworks que cria classes que estendem os pontos de flexibilidade de um framework

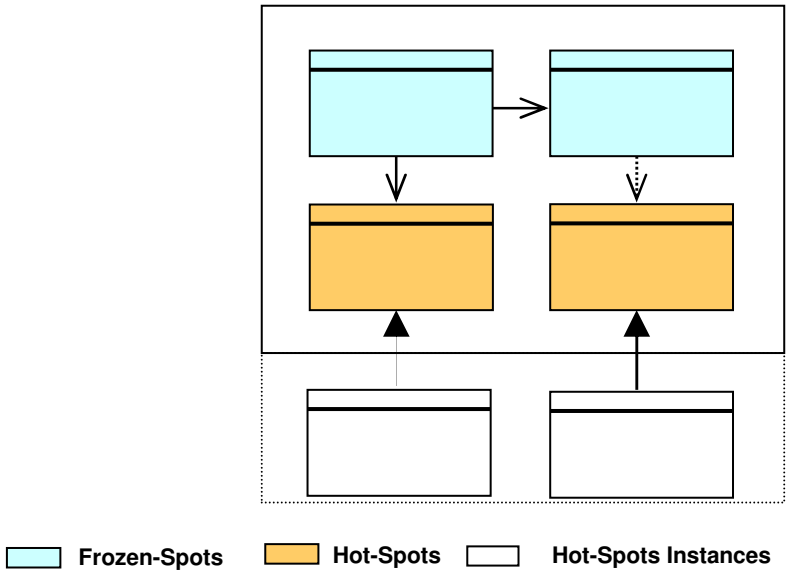

Figura 1. Estrutura de um Framework.

Frameworks Orientados a Objetos (OO), também chamados de frameworks de aplicação, geram famílias de aplicações orientadas a objetos. Seus pontos de extensão são definidos como classes abstratas ou interfaces, que são estendidas ou implementadas por cada instância da família de aplicações [15]. Frameworks OO, normalmente, são implementados utilizando mecanismos e linguagens da programação orientada a objetos. Frameworks de aplicação são classificados quanto ao seu escopo em **frameworks de infra-estrutura de sistemas**, **frameworks de integração de middleware** e **frameworks de aplicações corporativas** [15]. As três classificações de escopo de um framework de aplicação serão apresentadas a seguir:

(i) **frameworks de infra-estrutura de sistemas** simplificam o desenvolvimento de sistemas de infra-estrutura portáveis e eficientes, como sistemas operacionais, frameworks de comunicação. Sua utilização, em geral, acontece no ambiente interno da organização não sendo vendidos para clientes diretamente.

(ii) **frameworks de aplicações corporativas** são voltados para um domínio de aplicação específico como, por exemplo, os domínios da aviação, telecomunicações e financeiro. São primordiais para atividades de negócio.

(iii) **frameworks de integração de middleware** são freqüentemente utilizados para integrar componentes distribuídos e tratar aspectos de

infraestrutura, o desenvolvedor da aplicação utiliza tal framework para abstrair os detalhes em baixo nível e concentra-se no objetivo principal da aplicação.

Além da classificação quanto ao escopo, Fayad et al [15] também classificam os frameworks OO quanto às técnicas de extensão, podendo ser:

- ***Whitebox*** está fortemente ligado as características das linguagens OO como herança e classes abstratas. O reuso e a extensão das funcionalidades existentes são feitos através da criação de novas classes utilizando herança e implementação de métodos. Além disso, o usuário do framework deve entendê-lo muito bem para que possa criar uma instância;
- ***Blackbox*** são instanciados a partir de *scripts* ou de algum tipo de configuração (arquivo XML, conjunto de *wizards* gráficas). Para produzir uma instância o usuário do framework não precisa entender detalhes internos. Por fim, promovem menos flexibilidade que os frameworks do tipo *whitebox*;
- ***Graybox*** apresentam tanto as características dos frameworks *whitebox* quanto as do *blackbox.* Foram criados para evitar as desvantagens presentes nos frameworks *whitebox* e *blackbox.* Apresentam bastante flexibilidade e extensibilidade, além da habilidade de esconder informações não necessárias para os desenvolvedores da aplicação

Segundo Matsson [36], o processo de desenvolvimento e uso de frameworks compreendem a três etapas que são: análise do domínio do problema, *design* e implementação do framework e instanciação do framework. A análise do domínio do problema consiste em identificar os requisitos de um domínio e possíveis requisitos futuros. Durante a análise do domínio, os *hot-spots* e *frozen-spots* são parcialmente descobertos [31]. A etapa de *design* e implementação do framework define as abstrações do framework, onde os *hot-spots* e *frozen-spots* são modelados. Por fim, na etapa de instanciação os *hot-spots* do framework são implementados, gerando uma ou várias aplicações (instâncias).

Como apresentado anteriormente, cada aplicação gerada apresenta em seu código os *frozen-spots* do framework. A Figura 2, adaptada de [31], compara o processo de desenvolvimento tradicional de software orientado a objetos com o processo de desenvolvimento de frameworks. O desenvolvimento tradicional

orientado a objetos difere do desenvolvimento de frameworks, pois o primeiro gera uma única aplicação executável, equanto que um framework pode gerar inúmeras aplicações pertencentes a um mesmo domínio.

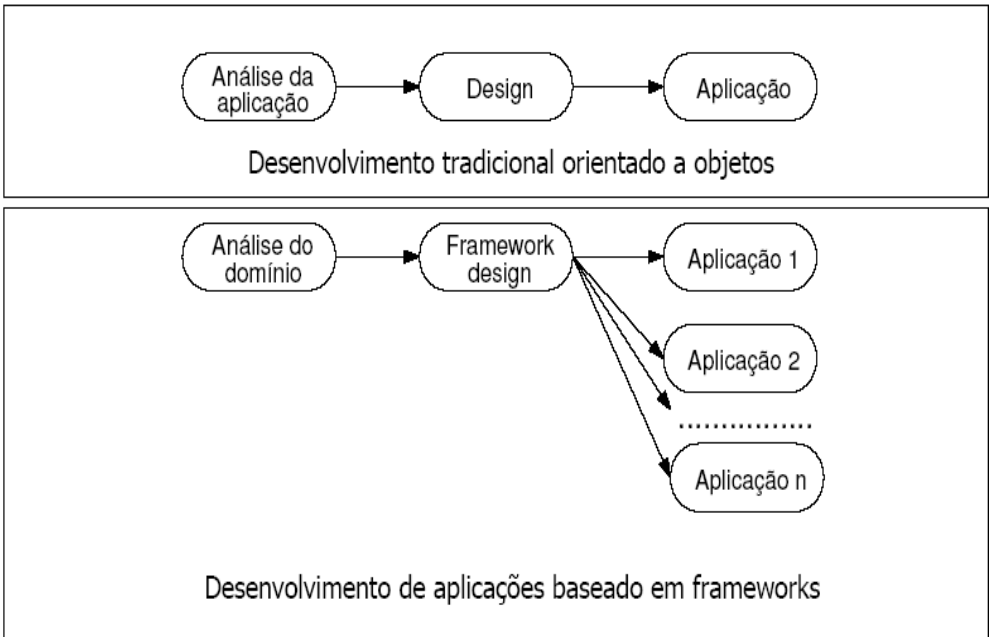

Figura 2. Processo de desenvolvimento de aplicações baseadas em frameworks comparado ao desenvolvimento OO tradicional.

A utilização da tecnologia de framework permite aumentar a produtividade e qualidade no desenvolvimento de aplicações. Frameworks oferecem reutilização de código e design [36]. Matsson [36], compara a utilização de frameworks com outras abordagens que promovem a reutilização de software, as quais serão apresentadas a seguir:

- Frameworks e *Design Patterns – design patterns* são mais abstratos do que um framework. Frameworks sempre estão relacionados a um domínio de aplicação, enquanto *design patterns* são mais genéricos e podem ser aplicados a vários domínios de aplicação. Além disso, *design patterns* possuem uma arquitetura menor do que a de um framework. Por fim, um framework pode conter vários *design patterns*, no entanto o oposto não se aplica.
- Frameworks e Biblioteca de Classes – biblioteca de classes são um conjunto de classes relacionadas que apresentam um propósito geral. Suas classes não são relacionadas a um domínio de aplicação especifica, como no caso das classes de um framework. Uma classe de uma biblioteca é

reutilizada sozinha, enquanto que uma classe de um framework é reutilizada em conjunto com outras classes. Outra questão que diferencia biblioteca de classes de frameworks é o impacto na arquitetura da aplicação, enquanto uma aplicação usa uma biblioteca de classes, ou seja, chamadas são feitas apenas da aplicação para a biblioteca de classes; já um framework pode assumir o controle da execução de uma aplicação, através da inversão de controle [37]

- Frameworks e Aplicações Orientadas a Objetos – uma aplicação OO descreve um programa executável completo que atende a todos os requisitos de uma especificação; já um framework captura as funcionalidades de diversas aplicações de um domínio, mas não é executável, já que não cobre o comportamento de uma aplicação específica.

O desenvolvimento de software baseado em frameworks é uma abordagem que colabora para a reutilização tanto de *design* quanto de código para projetos de software [36]. A tecnologia de frameworks representa o estado da arte na engenharia de software, pois agrega produtividade e qualidade no desenvolvimento de aplicações. A produtividade é alcançada pela diminuição das linhas de código necessárias para codificar uma aplicação, através do aproveitamento de códigos já existentes. A qualidade é obtida por ser uma tecnologia que se apóia em boas praticas da engenharia de software, sendo o reuso sua principal característica.

2.2.Programação Orientada a Aspectos

Ao longo da evolução da computação diversas técnicas de programação foram desenvolvidas para auxiliar a codificação de programas de computador. Alguns paradigmas e linguagens foram criados, a fim de facilitar o desenvolvimento e manutenção de programas. Metodologias e técnicas surgiram para acompanhar a complexidade dos sistemas de software que atualmente estão em praticamente todos os locais, desde um aparelho celular até supercomputadores.

Os primeiros computadores digitais surgiram na década de 40 para auxiliar cálculos científicos de diferentes espécies. Aplicações científicas possuem

estrutura de dados simples, porém necessitam de um grande número de computações aritiméticas. Fortran foi a primeira linguagem para aplicações científicas. Na década de 50 as primeiras aplicações comerciais surgiram, a primeira linguagem bem-sucedida para aplicações comerciais foi o COBOL [38].

Com o aumento do poder computacional do hardware e a redução no seu custo, no final da década de 60 e inicio da década de 70, iniciou-se uma mudança na forma de se desenvolver software. A mudança foi motivada pelo aumento da complexidade dos requisitos dos sistemas, onde novas aplicações, além das científicas, estavam sendo pensadas para uso computacional. Devido a complexidade dos novos sistemas, pesquisas apontaram algumas deficiências nas linguagens, metodologias e técnicas de programação, como por exemplo, o uso excessivo de instruções *GOTO*[5] que reduzem a legibilidade de um programa, tornando-os pouco confiáveis e difíceis de serem mantidos [38].

A partir da década de 70, novas metodologias de desenvolvimento de software foram criadas, como a programação estruturada. O conceito de procedimento é amplamente utilizado na programação estruturada, onde dados são enviados para subprogramas para computações [38]. Na década de 80 surgiu a programação orientada a objetos (POO), um paradigma que tem como idéia principal o conceito de que os objetos do mundo real que interagem com outros objetos, podem ser simulados computacionalmente.

A programação orientada a objetos trouxe benefícios que os paradigmas predecessores não contemplavam, uma vez que sua base está focada em conceitos e abstrações para modelar objetos do mundo real para serem implementados na forma de software, como por exemplo, o conceito de classe que representa a estrutura geral de um objeto, juntamente com suas características (atributos) e seu comportamento (métodos). A idéia de estruturar objetos do mundo real em termos de software possibilitou um grande avanço no desenvolvimento de sistemas, devido a possilidade de qualquer objeto do mundo real, concreto ou abstrato, poder ser representado em um sistema de computador [32].

Atualmente o paradigma orientado a objetos vem sendo utilizando amplamente no desenvolvimento de aplicações pertencentes aos mais diferentes

[5] Desvio incondicional que transfere o controle da execução para um lugar especificado no programa

domínios, desde aplicações cientificas até as aplicações comerciais. Por sua facilidade em modularizar regras de negócio em classes, a POO ajuda os desenvolvedores que seguem boas práticas de engenharia de software, possibilitando que o desenvolvimento das aplicações se torne cada vez mais modulares, desta forma, facilitando a manutenção e a futura evolução do sistema.

Apesar dos grandes benefícios da programação orientada a objetos, pesquisas apontaram que alguns interesses de um sistema não são tratados com clareza pela POO [18]. Dijkstra afirma que um sistema é composto de diversos interesses e que para tratar da melhor forma um sistema, é importante separar os seus interesses **(*separation of concerns)***, assim, é possível que o desenvolvedor se concentre com mais facilidade no desenvolvimento de um interesse por vez, de forma que os outros interesses serão tratados posteriormente [18].

Um *concern* (interesse) é um requisito, uma propriedade, uma funcionalidade de um sistema [11]. Segundo [21], os interesses podem ser classificados como: **interesses núcleo (*core concerns*) –** são interesses que fazem parte da lógica de negócios de um módulo, ou seja, o que o módulo se propõem à fazer; e **interesses transversais (*crosscutting concerns*) –** são restrições globais e interesses periféricos que permeiam diversos módulos de um sistema e que não estão diretamente relacionados com a lógica de negócios de um determinado módulo, geralmente são requisitos não funcionais, como por exemplo, segurança, persistência e tratamento de erros.

A programação orientada a objetos é capaz de tratar de forma clara os requisitos funcionais (interesses) de um sistema, por meio da abstração de classe que permite separar quais classes fazem parte do sistema, assim, é possível tratar cada classe como uma unidade independente das outras classes. Porém, alguns interesses ou requisitos, tipicamente os não funcionais, não são tratados de maneira satisfatória em sistemas complexos pela programação orientada a objetos.

As limitações da POO em tratar a modularização de determinados interesses que entrecortam outros módulos de um sistema, também chamados de **interesses transversais (*crosscutting concerns)***, levaram à observação dos seguintes problemas: **i) código espalhado (*code scattering*) –** indica que um interesse é implementado em vários módulos do sistema; **ii) código entrelaçado (*code tangling*)** – ocorre quando um único módulo manipula vários interesses simultaneamente.

Como exemplo de interesses transversais, podemos citar: persistência, segurança, tratamento de erros, *logging* e distribuição. A não modularização de alguns interesses transversais em sistemas complexos, pode trazer algumas conseqüências, tais como, a dificuldade de compreensão, manutenção e reutilização dos sistemas [21]. A Figura 3 tirada de [21], ilustra o interesse transversal *logging* que é usado no servidor web Tomcat, as colunas representam as classes e a linhas vermelhas representam o interesse *logging* que entrecorta várias classes do sistema.

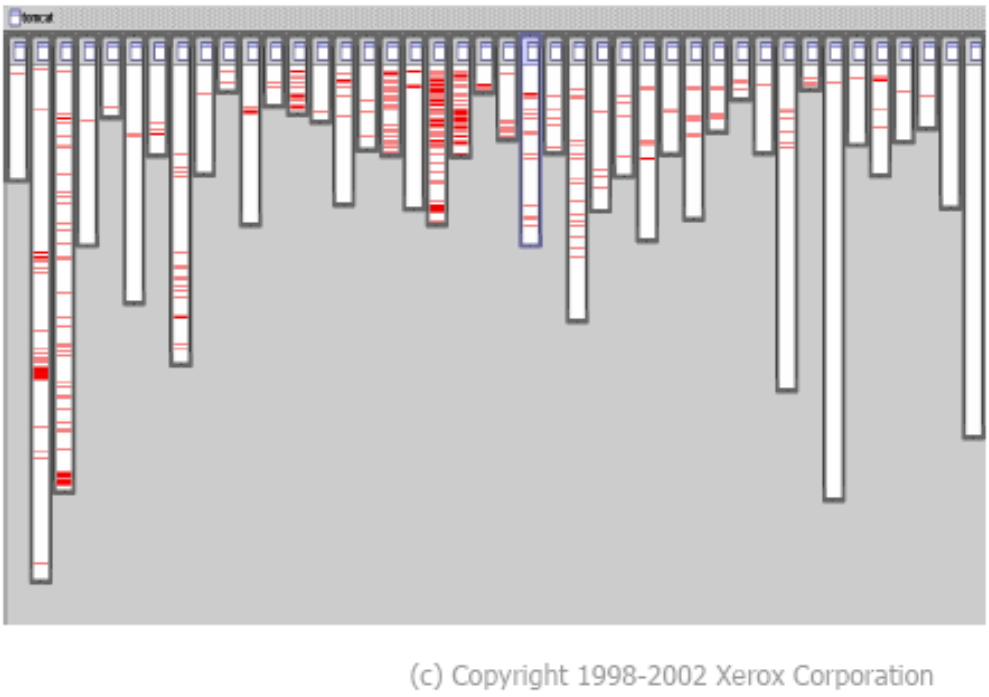

Figura 3. Interesse Transversal *logging* representado no Tomcat.

Com o objetivo de prover técnicas para modularizar os interesses transversais que ficam espalhados e entrelaçados em um sistema, diversas metodologias de programação surgiram após o surgimento da POO, dentre elas podemos destacar: programação generativa [39], meta programação [54], programação orientada a sujeitos [55], programação adaptativa [56], e filtros de composição [57]. Em 1997, surgiu a programação orientada a aspectos (POA), que tem por objetivo ser uma abordagem complementar ao paradigma de programação orientado a objetos, através da separação avançada de interesses [6]. A programação orientada a aspectos vem sendo a abordagem mais utilizada atualmente para modularizar interesses transversais, ela criou novas abstrações para tratar os códigos que não são claramente modularizados pela POO, através de

mecanismos que separam os interesses transversais e os compõem aos interesses núcleo de um sistema.

Algumas linguagens fornecem mecanismos para a implementação da programação orientada a aspectos. A linguagem mais utilizada atualmente para implementar POA é a AspectJ [12], que é uma extensão da linguagem Java que fornece os elementos para o desenvolvimento de programas orientados a aspectos. Além de AspectJ, outras implementações da POA existem para algumas linguagens de programação, tais como, AspectC [40] para a linguagem C; Pythius [41] para a linguagem Python, AspectC# [42] para a linguagem C#, entre outras.

Um **aspecto** é uma abstração da linguagem AspectJ para definir uma unidade de modularização que é capaz de modularizar um interesse transversal que entrecorta diversas partes de um sistema. A Figura 4 apresenta o conceito de aspecto que modulariza um interesse transversal que se espalha por várias classes de um sistema. As classes representadas por um retângulo azul, inicialmente apresentam um interesse transversal que entrecorta todas as classes. Este interesse é representado pela linha vermelha, com a utilização da abstração de aspectos, o interesse que antes estava adicionado aos códigos de todas as classes passou a ser modularizado de forma que permitiu a retirada do seu código das classes, assim, permitindo a modularização do interesse transversal em um aspecto que está representado pelo losango vermelho.

Os elementos que fazem parte de um aspecto na linguagem AspecJ seguem as mesmas idéias da abstração de classe para a linguagem Java. Assim como as classes, um aspecto pode conter métodos, atributos e sub-aspectos. Além disso, outros elementos foram introduzidos pela linguagem AspectJ para permitir a modularização dos interesses transversais (aspectos) e a sua posterior combinação para afetar/interceptar os interesses núcleo de um módulo, que são: ***join point, pointcut, advice e inter-type declaration***.

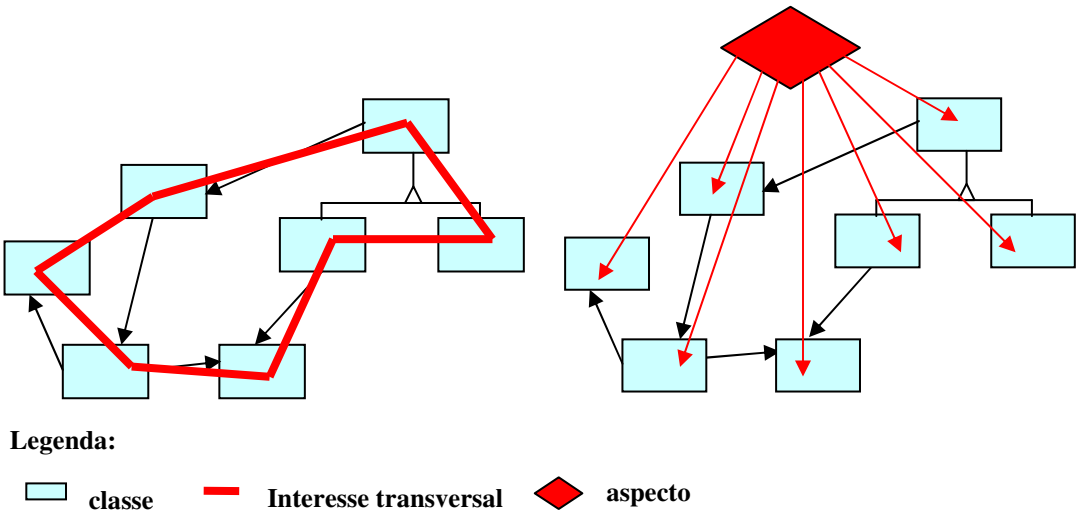

Figura 4. Modularização do interesse transversal que atravessa diversas classes através de um aspecto

Um ***join point*** é definido como um ponto de execução de um programa, onde os aspectos afetam o comportamento dos interesses núcleo de um sistema. Como exemplo de *join points* podemos citar: execuções de métodos, chamadas de métodos, acesso a um atributo, a inicialização de um objeto, entre outros. Um ***pointcut*** é um conjunto de *join point* que expõem dados do contexto de execução desses pontos de combinação. Além disso, *pointcuts* podem ser combinados para criar novos *pointcuts*. A combinação de *pointcuts* é feita através de operadores lógicos. Um ***advice*** é o código executado antes (*before)*, no lugar *(around)* ou depois (*after*) de um *pointcut* ser afetado. Um *advice* é similar a um método em Java. A ***inter-type declaration*** permite alterar de maneira estática a estrutura de uma aplicação com a introdução de novos atributos e métodos nas classes básicas de um sistema [21].

Um exemplo de definição de um aspecto em AspectJ tirado de [22] é apresentado na Figura 5. O aspecto *FaultHandler* consiste de uma *inter-type declaration* que introduz um atributo na classe *Server* (linha 03), dois métodos (linhas 05-07 e 09-11), a definição de um *pointcut* (linha 13) e duas *advices* (linhas 15-17 e 19-22). O *pointcut*, chamado de *services*, define como *join points* os pontos da execução do programa onde objetos da classe *Server* têm qualquer um de seus métodos públicos chamados.

```
01 aspect FaultHandler {
02
03   private boolean Server.disabled = false;
04
05   private void reportFault() {
06     System.out.println("Failure! Please fix it!.");
07   }
08
09   public static void fixServer(Server s) {
10     s.disabled = false;
11   }
12
13   pointcut services(Server s): target(s) && call(public * * (..));
14
15   before(Server s): services(s) {
16     if (s.disabled) throw new DisabledException();
17   }
18
19   after(Server s) throwing (FaultException e): services(s) {
20     s.disabled = true;
21     reportFault();
22   }
23 }
```

Figura 5. Exemplo da definição de um aspecto em AspectJ.

2.3. Frameworks e Aspectos

Com o surgimento da orientação a aspectos [6] e da linguagem de programação AspectJ [6, 22] , alguns autores iniciaram pesquisas sobre como o desenvolvimento de frameworks poderia ser endereçado com o uso da tecnologia de aspectos, através da reutilização de interesses transversais que os frameworks apresentam. Estruturalmente um framework orientado a aspectos é formado por classes.

Muitos pesquisadores utilizam o termo Framework Orientado a Aspectos (FOA), para definir um framework orientado a objetos que usa estruturas de aspectos em sua implementação [7]. Para Hanenberg, um FOA é um conjunto de aspectos concretos e abstratos [9]. Vários trabalhos recentes abordam o uso de frameworks orientados a aspectos. A seguir alguns destes trabalhos são discutidos.

Hanenberg et al [25] propõem um framework OA que oferecer alternativas de algoritmos para percurso em grafos. Os autores desenvolveram alguns requisitos do framework como: algoritmos de caminhamento, tratamento de nós adjacentes, políticas de ciclos e de comportamento. Cada requisito foi implementado de forma separada. Para os autores, a falta de adequação da forma

de composição da orientação a objetos, para combinar os requisitos do framework, através da utilização de herança, gera código redundante.

Garcia et al [26, 27] propõem um framework OA para a implementação de sistemas multi-agentes. A idéia dessa abordagem é modularizar as propriedades (interesses) de uma arquitetura de software baseada em agentes, tais como, mobilidade, aprendizado e autonomia. Com o aumento da complexidade dos agentes, suas propriedades entrecortam as funcionalidades básicas de outros módulos da arquitetura de um sistema multi-agente. A abordagem apresenta um método orientado a aspectos para permitir a separararação de interesses, assim permitindo a modularização das propriedades dos agentes.

Constantinides et al [29] propõem um framework para concorrência, onde esse interesse transversal é inserido nas aplicações. O framework proposto nesse trabalho, foi um dos primeiros que abordou a utilização da tecnologia de aspectos, apesar de não ter utilizado uma linguagem de programação orientada a aspectos para desenvolver tal framework.

Vanhaute et al [30] utilizam um FOA para o interesse transversal de segurança. Para os autores, o sub-interesse de controle de acesso, que é utilizado pelo interesse de segurança, apresenta o mesmo requisito para várias aplicações, mas apresenta algumas políticas específicas de uma aplicação, por exemplo, o usuário que acessou os recursos disponíveis e o sistema. Um problema identificado pelos autores foi a definição dos conjuntos de junção, pela natureza estática das definições, elas só podem ser referenciadas e estendidas estaticamente. Devido a essa restrição, existe a impossibilidade de isolar uma definição de conjunto de junção abstrata em um contrutor separado e confiar na delegação para selecionar a versão concreta.

Soares et al [43] desenvolveram frameworks com aspectos para: (i) persistência; (ii) distribuição e (iii) concorrência. Os frameworks atingem os seguintes interesses: controle de transações, carregamento de objetos, conexão com o banco de dados, sincronização do estado de objetos com entidades do banco de dados.

Camargo et al [44] propõem uma arquitetura de referência para o projeto e implementação de frameworks orientados a aspectos. A arquitetura separa a parte funcional da parte de composição, assim, contribuindo para a melhor compreensão de sua estrutura e facilitando a sua evolução. A vantagem dessa

abordagem é a facilidade para integrar vários frameworks em um repositório para apoiar o reuso, dessa forma, facilitando o processo de reuso.

Pinto et al [49] abordam a utilização de um framework de aplicação orientado a aspectos, para auxiliar o desenvolvimento de ambientes virtuais de colaboração. A utilização da tecnologia de aspectos facilitou a configuração e evolução desses ambientes. Em conseqüência da diversidade e quantidade de comportamentos em um ambiente virtual de colaboração, os aspectos foram classificados em tipos, entre os quais podemos citar : orientado ao usuário, orientado ao ambiente, orientando ao componente, entre outros.

Kulesza et al [28] apresentam uma abordagem para o desenvolvimento de frameworks usando programação orientada a aspectos. O trabalho propõe uma abordagem sistemática para o desenvolvimento de frameworks usando técnicas orientadas a aspectos (OA). O objetivo central da abordagem é melhorar a capacidade de extensão e configuração de frameworks orientados a objetos (OO) para diferentes cenários de reutilização, através de uma melhor gerência de suas características. A abordagem é composta por: (i) um conjunto de diretrizes para o projeto e implementação de frameworks usando programação orientada a aspectos; e (ii) um modelo generativo usado para a instanciação automática do framework e suas variabilidades OO e OA. As diretrizes propõem a definição de um conjunto de pontos de junção de extensão (EJPs – *extension join points*) no código do framework, os quais podem ser usados para estender a funcionalidade básica do framework através da implementação de aspectos de extensão. Tais aspectos são responsáveis pela implementação de características transversais opcionais, alternativas ou de integração demandadas por usuários do framework.

Esse trabalho de dissertação explora o uso combinado de técnicas de OO e OA para o projeto e implementação de um framework de monitoramento e análise de processos de negócio. A técnica de OA é explorada, sobretudo, para permitir uma composição transversal não-invasiva entre o framework proposto e o sistema web a ser monitorado.