UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

UM PROCESSO PARA EXTRAÇÃO DE META-DADOS EM CÓDIGO ORIENTADO A ASPECTOS

TRABALHO DE GRADUAÇÃO

Felipe José Hanauer

Santa Maria, RS, Brasil
2012

UM PROCESSO PARA EXTRAÇÃO DE META-DADOS EM CÓDIGO ORIENTADO A ASPECTOS

por

Felipe José Hanauer

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Dr. Eduardo Kessler Piveta

Trabalho de Graduação N. 340

Santa Maria, RS, Brasil
2012

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

UM PROCESSO PARA EXTRAÇÃO DE META-DADOS EM CÓDIGO ORIENTADO A ASPECTOS

elaborado por
Felipe José Hanauer

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof. Dr. Eduardo Kessler Piveta
(Presidente Orientador)

Profa. Dra. Marcia Pasin (UFSM)

Profa. Dra. Iara Augustin (UFSM)

Santa Maria, 06 de julho de 2012

DEDICATÓRIA

Este trabalho é dedicado a todos que, de certa forma, nunca me deixaram desistir e sempre me deram forças para seguir em frente.

“You rise

You fall

You're down, then you rise again

What don't kill ya make ya more strong”

– Metallica

AGRADECIMENTOS

Agradeço a Deus, por me deixar existir e me dar os pais José Canísio Hanauer e Inez Maria Hanauer, que me deram a educação necessária para chegar até esta fase da vida.

Ao meu irmão Gabriel Hanauer, que foi meu braço direito não só neste trabalho, mas em todo o percurso percorrido, e que foi fundamental para alcançar o final deste curso.

À minha irmã Monique Hanauer, por todo o apoio psicológico e carinhoso que só uma irmã pode dar.

À Márcia Horn, que se preocupou junto comigo e, que sem seu apoio, força e paciência, teria sido muito mais difícil alcançar meus objetivos.

Agradeço também ao meu professor orientador Eduardo Kessler Piveta, por aceitar o desafio desse trabalho.

Enfim, agradeço a todos que me deram alguma ajuda e que, com certeza, facilitou a minha caminhada até o final do curso de Ciência da Computação.

RESUMO

Sistemas de médio e grande porte apresentam um grande desafio para as empresas no que se refere à organização de seu projeto. É fácil encontrar um programa com código desorganizado e repetitivo. Isso faz com que a manutenção e até mesmo a implementação de tais programas se tornem muito mais complicadas. Seria mais simples se existisse uma ferramenta que organizasse melhor o código fazendo com que a tarefa de manutenção e implementação se tornasse menos árdua. Atualmente, existem vários programas IDE[1] que oferecem a facilidade de refatorar o código tornando-o mais simples, principalmente em código orientado a objetos. Porém, em código orientado a aspectos, essa facilidade ainda é pouco explorada. Este trabalho apresenta um modo de melhorar a refatoração em código orientado a aspectos, facilitando as atividades dos programadores. É descrito um processo de extração de meta-dados e busca em código escrito em AspectJ, bem como uma prova de conceito, na forma de uma ferramenta.

Palavras-chave: refatoração; código; ferramenta; IDE, aspectos;

[1] Do inglês Integrated Development Environment, ou Ambiente de Desenvolvimento Integrado, é um programa que reúne ferramentas úteis para desenvolvedores de softwares.

ABSTRACT

Organization of projects on medium and large systems is a big challenge for companies. It’s easy to find a program with disorganized and repetitive codes. This makes the maintenance and even the implementation of such programs much more complicated. If there were a tool to better organize the code, the task of maintenance and implementation would be less arduous. Today we have several IDEs tools that offer the facility of refactoring, which makes the code simpler, particularly in object-oriented languages. However, in aspect-oriented languages this facility is still poorly explored. This work shows a way to improve aspect-oriented code refactoring, facilitating the activities if developers. A process of extracting metadata and search in codes written in AspectJ is described, and a proof of concept, in the form of a tool.

Keywords: refactoring; code; tool; IDE, aspects;

LISTA DE FIGURAS

Figura 3.1 – Processo de extração de meta-dados 27
Figura 3.2 – Passos da busca de meta-dados. 28
Figura 3.3 – Estrutura que usa Visitantes (GAMMA, HELM, *et al.*, 1995). 30
Figura 4.1 – Arquivo ApectJ aberto no aplicativo. 34

LISTA DE SIGLAS

AJDT	*AspectJ Development Tools*
API	*Application Programming Interface*
AST	*Abstract Syntax Tree*
HTML	*HyperText Markup Language*
POA	Programação Orientada a Aspectos
POO	Programação Orientada a Objetos
W3C	*World Wide Web Consortium*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Objetivos 14
1.2 Justificativa 14
1.3 Estrutura 15

2 REVISÃO DE LITERATURA 16
2.1 Programação Orientada a Aspectos 16
2.1.1 AspectJ 17
2.2 Reflexão Computacional 18
2.2.1 API de Reflexão em Java 19
2.2.2 API de Reflexão em AspectJ 20
2.3 XML 21
2.3.1 XQuery 21

3 UM PROCESSO PARA EXTRAÇÃO DE META-DADOS EM CÓDIGO ORIENTADO A ASPECTOS 27
3.1 Ferramentas usadas no trabalho 28
3.2 Análise Sintática 28
3.3 Extração de meta-dados 29
3.3.1 O padrão Visitor 29
3.4 Busca 31
3.4.1 Escopo da busca 31
3.4.2 Consultas 32

4 ESTUDO DE CASO 33
4.1 Preparação 33
4.2 O aplicativo 33
4.2.1 Dependências 33
4.2.2 Usando o aplicativo 34
4.2.3 Saída 34
4.3 O código-fonte do processo de extração 34
4.4 Resultado da extração 39

5 CONCLUSÃO 42

6 BIBLIOGRAFIA 43

APÊNDICE A – ARQUIVO PRINCIPAL.JAVA 45

APÊNDICE B – ARQUIVO BUSCA.JAVA 46

APÊNDICE C – ARQUIVO AJADVICE.JAVA 48

APÊNDICE D – ARQUIVO AJASPECTO.JAVA 49

APÊNDICE E – ARQUIVO AJINTERTYPE.JAVA 50

APÊNDICE F – ARQUIVO AJPOINTCUT.JAVA 51

APÊNDICE G – ARQUIVO ASPECTJ.JAVA 52

APÊNDICE H – ARQUIVO JANELA.JAVA 53

APÊNDICE I – ARQUIVO PRINCIPAL_ABOUTBOXPANEL1.JAVA 57

APÊNDICE J – ARQUIVO SCFILTRO.JAVA 58

APÊNDICE L – ARQUIVO UTILS.JAVA 59

APÊNDICE M – ARQUIVO ASPECTASTVISITOR.JAVA 60

APÊNDICE N – ARQUIVO VISITAR.JAVA 62

1 INTRODUÇÃO

Um dos principais desafios de um gerente de projetos de um sistema computacional é conseguir evitar problemas de codificação nos programas. Isto é, tentar organizar o projeto para que ele fique bem estruturado e com uma fácil legibilidade e manutenção. Este desafio se torna mais difícil no momento em que o número de pessoas envolvidas é grande e, consequentemente, o código desenvolvido se torna facilmente confuso e desorganizado. Quando o código começa a ficar desorganizado, a tendência é que essa degradação seja contínua. Aparecem códigos repetitivos, pouco coesos, com o mesmo objetivo, etc.

Este tema é bastante estudado em Engenharia de Software, onde existem várias técnicas para organizar um projeto. Uma delas é a refatoração:

> Refatoração é o processo de alterar um sistema de software de modo que não se altera o comportamento externo do código e ainda melhora a estrutura interna. É uma forma disciplinada de limpar o código que minimiza as chances de produzir erros. Em essência, quando você refatora você está melhorando o *design* do código depois de ter sido escrito. (FOWLER, 2002, p. 9).

Em sistemas de pequeno porte, aplicar manualmente as transformações de refatoração é uma tarefa exaustiva. Em aplicações de grande porte, é praticamente inviável, demonstrando a importância de ferramentas automatizadas em processos de refatoração.

Existem vários tipos de refatoração para a Programação Orientada a Objetos (POO), como por exemplo, quebrar um método grande em vários métodos menores, renomear uma variável para um nome mais claro, etc. Vários ambientes de desenvolvimento, como o Eclipse, possuem opções para automatizar a refatoração na programação orientada a objetos. Entretanto, mesmo com essas ferramentas, a refatoração mostra-se incorreta em algumas situações (BOS, 2008, p. 21).

A refatoração pode trazer muitos problemas para o desenvolvedor. Martin Fowler (2002, p. 6) menciona que “Refatoração é um risco. Ela requer alterações no código de trabalho que podem introduzir erros sutis. Refatoração, se não for feito corretamente, pode fazer você perder dias, ou até semanas.”.

O paradigma da Programação Orientada a Aspectos (POA) tem como principal objetivo apresentar uma extensão ao paradigma da POO, introduzindo um novo conceito de modularização. Através da POA, os comportamentos transversais ao sistema (*crosscutting concerns*) (KICZALES, LAMPING, *et al.*, 1997) podem ser encapsulados em um único

módulo, o aspecto. Portanto, a POA complementa a POO. A linguagem de POA mais utilizada é o AspectJ (http://www.eclipse.org/aspectj/), que é uma linguagem orientada a aspectos que complementa a linguagem orientada a objetos Java.

Na POA a refatoração ainda não foi muito explorada e pouco se percebe nos avanços dessa técnica, ao contrário da refatoração em linguagens orientada a objetos, que está mais adiantada nesse recurso.

1.1 Objetivos

O objetivo deste trabalho é especificar um processo para realizar a extração de meta-dados de códigos-fonte orientados a aspecto que possa servir de base para a refatoração, bem como o desenvolvimento de uma prova de conceito através de uma ferramenta.

Mais especificamente, os objetivos são:

- analisar sintaticamente o código-fonte orientado a aspectos;
- extrair os meta-dados;
- buscar informações utilizando XQuery.

1.2 Justificativa

O trabalho de refatoração de código é importante, mas também arriscado, pois envolve modificar códigos sem, entretanto, modificar o resultado do mesmo. E, nem sempre, há a necessidade de melhorar o código que já pode estar bem organizado, ou porque o desenvolvedor não queira. Mesmo que o código não esteja bem estruturado, nem sempre a modificação dele pode levar a benefícios. Códigos complexos, às vezes, é melhor que não sejam modificados (ARSENOVSKI, 2009). Como saber se deve ser mesmo melhorado e o que deve ser melhorado?

A justificativa para o estudo e a criação deste trabalho é a necessidade de conseguir facilitar o trabalho de refatoração em códigos orientados a aspectos através da melhor extração de dados possível. Com os meta-dados certos, a tarefa de refatoração tem menos chances de erro, e mais chances de descobrir se realmente há a necessidade de modificar o código.

Portanto, o processo de extração de meta-dados e busca correta de informações nos permite ter melhor acuidade ao processo de refatoração.

1.3 Estrutura

O segundo capítulo deste trabalho apresenta uma breve revisão de literatura acerca da POA, em específico a linguagem AspectJ, uma reflexão computacional da *Application Programming Interface* (API) Java e API AspectJ e, sobre a linguagem de consulta XQuery.

O terceiro capítulo traz o desenvolvimento do trabalho, uma introdução, a escolha de uma forma de armazenamento, o estudo da navegação em árvores sintáticas e, por fim, a busca de meta-dados a partir das APIs.

Já o quarto capítulo descreve a implementação de um caso de estudo relativo ao estudo do capítulo 3.

Por fim, no quinto capítulo são apresentadas as conclusões deste trabalho.

2 REVISÃO DE LITERATURA

Para entender a tecnologia que está por trás da refatoração de códigos, especialmente de códigos orientados a aspectos, é necessário que sejam estudadas algumas definições importantes. Esta seção do trabalho tem o objetivo de explicar o conceito de programação orientada a aspectos, assim como AspectJ, e técnicas que podem ser usadas para extrair meta-dados de um programa em AspectJ.

2.1 Programação Orientada a Aspectos

A Programação Orientada a Aspectos (POA) (KICZALES, LAMPING, *et al.*, 1997) foi criada para solucionar problemas que a POO não consegue resolver. Os interesses de um sistema que não podem ser modularizados, muitas vezes, estão espalhados entre vários métodos e classes diferentes e, mesmo com a estrutura da POO, esses interesses não podem ser separados e definidos no código, fazendo com que fique difícil de manter e entender o sistema. Pelo fato de estarem espalhados e entrelaçados por todo o código, esses interesses são definidos como interesses transversais[2] (JÚNIOR, 2011). A POA foi criada justamente para solucionar esse problema, ou seja, modularizar os interesses transversais de um programa em uma única unidade de abstração bem definida.

POA é uma técnica adicional, e não uma técnica substitutiva para POO. Com os avanços dos paradigmas de programação, fica cada vez mais fácil resolver problemas complexos com maior eficiência. POA é uma nova técnica, que procura avançar um passo na resolução de problemas complexos e disponibilizar mais uma ferramenta ao desenvolvedor para melhorar sua produtividade (KISELEV, 2003).

POA busca encapsular as chamadas de métodos em aspectos. Ou seja, inserir, em código existente, blocos de códigos que são executados em momentos definidos pelo programador, e que muda o comportamento original do programa. Esse novos comportamentos são os adendos[3] e o momento, ou ponto da execução desse adendo, é chamado de ponto de junção[4]. Um conjunto de regras que define quais pontos de junção são selecionados, é chamado de pontos de atuação[5]. Os aspectos ainda podem possuir

[2] Do inglês: Crosscutting Concerns.
[3] Do inglês: Advices.
[4] Do inglês: Joinpoint.
[5] Do inglês: Pointcut.

declarações- intertipos[6]. Essas declarações podem ser atributos ou métodos a serem inseridos nas classes existentes ou em uma nova classe que estende a classe original (THE ASPECTJ TEAM, 2002).

Como exemplo, considere uma aplicação que, inicialmente, faz o registro de todas as ações de um sistema. Porém, depois de um tempo de uso, o administrador verifica a necessidade de fazer uma cópia dos registros antes de fazer um novo registro. Também necessita verificar se o usuário que faz o registro é um usuário confiável. Esses novos objetivos verificados pelo administrador são os interesses. Para adicionar essas funções, um desafio é lançado aos programadores que terão de fazer uma complexa modificação no código, que ao final, poderá se tornar mais desorganizado e suscetível a falhas. Entretanto, para não modificar o código existente, são criados aspectos que implementarão todas as novas funcionalidades exigidas pelo administrador, que ficarão separados em blocos distintos no código. Assim, a aplicação fica mais organizada e com suas funcionalidades importantes separadas em aspectos.

As principais aplicações desse paradigma de programação são: em aplicações com necessidade de sincronização em sistemas concorrentes; em sistemas distribuídos, colaborando com a transparência do projeto; no tratamento de exceções; na persistência de dados, principalmente na parte de manutenção; e na auditoria de sistemas, sendo que com a POA o código estaria centralizado em um único local (JÚNIOR, 2011).

A POA fornece uma ferramenta na elaboração de sistemas complexos. Porém devem-se tomar alguns cuidados para que a solução não gere problemas ainda maiores. Um problema que pode ocorrer é a dificuldade de depurar programas orientados a aspectos. A POA é bem separada sintaticamente no código, porém, um compilador pode mesclar o código em tempo de execução.

2.1.1 AspectJ

A POA teve sua consolidação a partir da criação da linguagem AspectJ (FILMAN, ELRAD, *et al.*, 2004), que foi desenvolvida com estudos feitos nos laboratórios da Xerox, mais especificamente de uma divisão de pesquisa chamada Xerox PARC em 1997 (FILMAN,

[6] Do inglês: Intertype Declarations.

ELRAD, *et al.*, 2004). Atualmente, é mantida pela Fundação Eclipse[7], que concentra seus objetivos na integração da linguagem na ferramenta de desenvolvimento Eclipse.

AspectJ é uma extensão de Java que implementa os conceitos da POA. Ela possui toda a estrutura básica da POA. Os pontos de junção podem ser a invocação ou execução de um método, a inicialização de objetos, a execução de construtores, o lançamento ou tratamento de exceções, o acesso a atributos, entre outros. Quem define essas regras são os pontos de atuação.

Os adendos representam a implementação do comportamento dos pontos de atuação. Alguns tipos de adendos são: *before*, que executa antes da computação do ponto de junção; *after returning*, que executa após a computação ter sida feita com sucesso; *after throwing*, que executa após a computação ter lançado um erro; *after*, que executa após a computação ter sida finalizada, com ou sem sucesso; e *around*, que executa quando o ponto é alcançado e toma o controle da computação, podendo decidir o futuro da execução.

A listagem 2.1 apresenta um exemplo básico da estrutura.

```
01 aspect NovoAspecto {
02
03   pointcut Altera(Metodo m): call(void Metodo.set(int));
04
05   after(Metodo m): Altera(m) {
06
07     /* Bloco de código de ações */
08
09   }
10 }
```

Listagem 2.1 – Exemplo de AspectJ.

Na linha 1 é feita a declaração do aspecto chamado *NovoAspecto*. A linha 3 apresenta a declaração do ponto de atuação onde define que o adendo *Altera()* é executado sempre que o método *set(int)* da classe *Metodo* é chamado. Por fim, da linha 5 a 7 é declarado o adendo *Altera(m)* que é chamado antes do ponto definido no ponto de atuação, ou seja, sempre antes de uma chamada ao método *set(int)* da classe *Metodo*.

2.2 Reflexão Computacional

Eventualmente, existe a necessidade de um programa ser alterado sem, no entanto, alterar seu código fonte, o que torna uma tarefa complexa e de muita dificuldade. Esta necessidade levou à criação de um novo paradigma: a reflexão computacional. A capacidade

[7] http://www.eclipse.org/aspectj/

de um programa que consegue obter informações e dados de sua própria execução é chamada de reflexão computacional. Esta capacidade é vantajosa, pois permite que o próprio programa use os dados coletados para melhorar seu desempenho e oferecer novas funcionalidades. O programa monitora a si mesmo. Essa capacidade é obtida através do paradigma chamado programação reflexiva ou programação orientada a reflexão (FERNANDES, 2000).

Em programação orientada a reflexão, deve-se entender computação como um processo ao invés de um programa. Assim, pode-se pensar em computação como a execução de um programa. Em paradigmas clássicos como Orientação a Objetos, Procedural, entre outros, especificam uma sequencia pré-definida em tempo de compilação, como as operações que modificam dados. Já a reflexão diz que as operações e o fluxo do programa é decidido dinamicamente em tempo de execução (SOBEL, 1996).

Quando um programa é compilado, toda a informação sobre sua estrutura é perdida. A reflexão possui um conceito de meta-informação, que salva várias informações a respeito da estrutura do programa, como nomes de classes, métodos e atributos, assim como seus valores em tempo de execução. Esses dados, então, são usados para decidir a execução por outro caminho, diferente do que seria usado originalmente, podendo melhorar o desempenho e coletar dados importantes para outras funções (SMITH, 1982).

Porém, apesar de ser uma poderosa ferramenta para programadores, também tem seus problemas. O código de um programa que usa o paradigma de reflexão é muito complexo. Isso o torna muito suscetível a erros. Um bom controle de versões também é necessário, já que a modificação de código é frequentemente necessária. A depuração também é prejudicada pois não há viabilidade de verificação sintática e semântica em tempo de compilação (SOBEL, 1996) e (SMITH, 1982).

2.2.1 API de Reflexão em Java

A linguagem Java possui suporte à reflexão. Ela conta com uma API com as principais funções para implementar a reflexão. A seguir, há a descrição dos principais métodos necessários para construir um programa reflexivo em Java (FORMAN e FORMAN, 2005).

Informações sobre classes:

- *java.lang.Class* – contém métodos para retornar um objeto da classe;
- *java.lang.reflect.Modifier* – contém métodos para obter informações sobre os modificadores de uma classe;

- *java.lang.reflect.Type* – contém métodos para obter informações sobre os tipos de uma classe.

Informações sobre os membros de uma classe:

- *java.lang.reflect.Constructor* – contém métodos para obter informações sobre os construtores declarados por uma classe;
- *java.lang.reflect.Field* – contém métodos para obter informações sobre os campos declarados por uma classe;
- *java.lang.reflect.Method* – contém métodos para obter informações sobre os métodos declarados por uma classe;
- *java.lang.reflect.Member* – interface implementada pelas classes *Field*, *Method* e *Constructor*;
- *java.lang.Package* – contém métodos para retornar um objeto de pacotes;
- *java.lang.reflect* – contém as classes básicas para a reflexão.

2.2.2 API de Reflexão em AspectJ

AspectJ fornece 3 objetos especiais, que podem ser usados em adendos, para acessar a reflexão (LADDAD, 2010):

- *thisJoinPoint* – contém as informações dinâmicas sobre o ponto de junção. Os seguintes métodos podem ser chamados através dele:
  - *getArgs*();
  - *getTarget*();
  - *getThis*();
  - *getStaticPart*().
- *thisJoinPointStaticPart* – contém as informações estáticas sobre o ponto de junção. Os seguintes métodos podem ser chamados através dele:
  - *getKind*();
  - *getSignature*();
  - *getSourceLocation*().
- *thisJoinPointEnclosingStaticPart* – contém informações estáticas sobre o contexto do ponto de junção. Possui os mesmos métodos do *thisJoinPointStaticPart*.

Outra classe frequentemente usada na reflexão é a classe *Signature*, que representa uma assinatura de um ponto de junção. Os principais métodos usados por ela são:

- *getDeclaringType*();
- *getName*().

Como AspectJ é uma linguagem que complementa a linguagem Java, a API de reflexão AspectJ apenas complementa a API de reflexão Java.

2.3 XML

Extensible Markup Language (XML) é uma linguagem de marcação de dados recomendada pela *World Wide Web Consortium* (W3C)[8]. Ela foi criada especificamente para guardar dados e informações e pode ser usada para criar marcas personalizadas, ao contrário da *HyperText Markup Language* (HTML) que possui marcas padrões e é utilizada somente para exibir conteúdos em navegadores (GABRICK, 2002).

Uma das principais características do XML é o poder de separar a interface dos dados estruturados, o que torna possível a integração com vários sistemas e plataformas diferentes. Depois que o cliente receber o dado, o mesmo pode ser manipulado e visualizado sem a necessidade de se comunicar com o servidor novamente. Assim os servidores ficam menos sobrecarregados. Ela pode ser usada por simples sistemas até programas com maior grau de complexidade (GABRICK e WEISS, 2002).

Como XML é estruturado na forma de árvore, ela pode ser usada para representar estruturas de dados como listas, registros e as próprias árvores. Isso também faz com que ela seja auto-documentada, ou seja, o próprio formato descreve a sua estrutura, o que é chamado de DTD (*Document Type Definition*) (GABRICK e WEISS, 2002).

2.3.1 XQuery

Com o aumento do uso de XML nos dias atuais, há um enorme volume de dados sendo guardado neste formato. Dados estruturados, semi-estruturados e não estruturados são salvos em XML. E a tendência é que cada vez mais se use essa tecnologia (WALMSLEY, 2007).

Todos esses dados são usados de alguma forma por sistemas. Por exemplo, em um sistema web, dados sobre informações sobre preços de produtos são buscados a todo o

[8] http://www.w3.org/

momento por clientes. O principal objetivo é ter a disposição os dados desejados para que possam ser formatados e exibidos ao usuário.

Justamente para ajudar a dispor esses dados foi desenvolvida uma linguagem de consulta chamada XQuery. Seu objetivo é buscar, de forma simples, os dados de arquivos XML para que possam ser reorganizados, transformados ou simplesmente exibidos ao usuário (WALMSLEY, 2007).

XQuery é uma linguagem de consulta, muito parecida com SQL, que foi desenvolvida pela W3C com um time de mais de 30 membros de companhias. Ela foi desenvolvida com base na linguagem XPath, usando suas principais expressões de consulta. Alguns dos principais objetivos de XQuery são (MCGOVERN, BOTHNER, *et al.*, 2003):

- consistência de plataforma;
- centrado em XML;
- trabalhar com conjunto de dados;
- facilidade de uso.

2.3.1.1 Características principais

XQuery tem uma variedade de características que permite uma série de operações em dados e documentos XML, como, por exemplo:

- seleção de informações baseadas em critérios específicos;
- filtrar informações indesejadas;
- pesquisa de informações dentro de um ou mais documentos XML;
- seleção, agrupamento e agregação de dados;
- transformação e reestruturação de dados XML;
- realizar cálculos aritméticos sobre números e datas;
- manipular textos para reformatá-los.

2.3.1.2 Usos de XQuery

XQuery pode ser usada não apenas para extrair dados de documentos XML, mas também para manipular e transformar os resultados.

Alguns exemplos de usos comuns para a linguagem XQuery são:

- extrair informações de um banco de dados relacional para uso em *web services*;
- geração de relatórios armazenados em bancos de dados para a apresentação na Web em formato XHTML;
- busca por documentos textuais em um banco de dados XML;
- extrair informações dos bancos de dados e transformá-los para a integração de aplicações;
- executar consultas *ad-hoc* em documentos XML para fins de teste ou pesquisa.

2.3.1.3 Estrutura de XQuery

A linguagem XQuery é constituída pelas seguintes áreas:

- for/let – são declarações análogas ao SQL SELECT. Essas cláusulas permitem definir variáveis ou iterar através de uma sequencia de valores;
- where – análogo ao SQL WHERE, essa cláusula fornece um conjunto de condições para filtrar ou limitar uma seleção inicial de uma cláusula for;
- order-by – análogo ao SORT BY em SQL, fornece as restrições de ordenação em uma sequência;
- return – análogo ao SQL RETURN, utiliza uma linguagem de formatação customizada para criar a uma saída. A saída não precisa ser necessariamente XML, embora seja otimizada para produzir XML;
- XPath – a maioria das funções XPath 2.0 são suportadas em XQuery;
- Functions – análogo as SQL stored procedures, podemos definir funções em XQuery usando a linguagem XPath que podem ser chamadas *inline* numa XML Query.

2.3.1.4 XQuery ou SQL

Uma das primeiras perguntas que pode surgir sobre XQuery é “Por que não criar uma versão atualizada do SQL e utilizá-lo com XML?”. A resposta é: porque cada fabricante de banco de dados implementa sua versão particular do SQL.

Quando uma consulta SQL é realizada, resumidamente cria-se uma tabela virtual a partir de várias outras tabelas, utilizando intersecções, uniões e filtros. Por isso, quanto mais tabelas forem consultas, maior será o esforço para se eliminar dados duplicados.

Bancos de dados relacionais são complexos e não são bons com tipos de dados semi-estruturados, como documentos, por exemplo.

O SQL requer um bom design das tabelas e relacionamentos e ele é otimizado para isso. Por isso, o SQL não manipula muito bem a imprevisibilidade e irregularidade na estrutura de dados.

2.3.1.5 Exemplo (MCGOVERN, BOTHNER, *et al.*, 2003)

Segue um exemplo de uma expressão XQuery contendo diversos elementos da linguagem:

```
01 declare namespace xs = "http://www.w3.org/2001/XMLSchema"
02 define function summaryText($char) returns xs:string
03 {
04   concat($char/name,' is a ',$char/gender,' ',$char/species)
05 }
06 <results>{
07   let $chars := input()//character[gender = 'Female']
08   for $char in $chars
09     where $char/level gt 5
10     return
11       <summary health="{$char/health}"> {
12       attribute level {$char/level},
13       attribute date {current-dateTime()},
14       summaryText($char)
15       }
16       </summary>
17   }
18 </results>
```

Listagem 2.2 – Exemplo de expressão em XQuery.

A consulta XQuery apresentada na listagem 2.2 é executada sobre um conjunto de dados que possui a seguinte formatação:

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <characters>
03   <character>
04     <name>Aleria</name>
05     <gender>Female</gender>
06     <species>Heroleim</species>
07     <vocation>Bard</vocation>
08     <level>5</level>
09     <health>25</health>
10   </character>
11   <character>
12     <name>Shar</name>
13     <gender>Male</gender>
14     <species>Human</species>
15     <vocation>Merchant</vocation>
16     <level>6</level>
17     <health>28</health>
18   </character>
19   <character>
```

```
    <name>Gite</name>
    <gender>Female</gender>
    <species>Aelvar</species>
    <vocation>Mage</vocation>
    <level>7</level>
    <health>18</health>
  </character>
  <character>
    <name>Horukkan</name>
    <gender>Male</gender>
    <species>Udrecht</species>
    <vocation>Warrior</vocation>
    <level>5</level>
    <health>32</health>
  </character>
  <character>
    <name>Gounna</name>
    <gender>Female</gender>
    <species>Noleim</species>
    <vocation>Mage</vocation>
    <level>8</level>
    <health>31</health>
  </character>
  <character>
    <name>Sheira</name>
    <gender>Female</gender>
    <species>Human</species>
    <vocation>Cleric</vocation>
    <level>4</level>
    <health>9</health>
  </character>
  <character>
    <name>Drue</name>
    <gender>Female</gender>
    <species>Voleim</species>
    <vocation>Warrior</vocation>
    <level>6</level>
    <health>32</health>
  </character>
  <character>
    <name>Paccu</name>
    <gender>Male</gender>
    <species>Human</species>
    <vocation>Merchant</vocation>
    <level>5</level>
    <health>24</health>
  </character>
</characters>
```

Listagem 2.3 – Conjunto de Dados em XML.

Quando a consulta da listagem 2.2 é executada, tem-se o resultado abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<results>
<summary date="2002-10-24T14:38:48" health="18" level="7">
   Gite is a Female Aelvar
</summary>
<summary date="2002-10-24T14:38:48" health="31" level="8">
   Gounna is a Female Noleim
</summary>
```

```
09 <summary date="2002-10-24T14:38:48" health="32" level="6">
10    Drue is a Female Voleim
11 </summary>
12 </results>
```

Listagem 2.4 – Resultado de uma consulta em XQuery.

3 UM PROCESSO PARA EXTRAÇÃO DE META-DADOS EM CÓDIGO ORIENTADO A ASPECTOS

Neste capítulo serão descritos os passos realizados para o desenvolvimento deste trabalho. Ele está dividido em duas principais partes: a primeira com o detalhamento da extração de meta-dados e a segunda com a explicação das consultas em código. Para ambas as partes são necessárias a instalação de ferramentas, que serão explicadas posteriormente em cada passo do desenvolvimento.

O processo consiste em receber um arquivo com código fonte em linguagem AspectJ que será, então, analisado sintaticamente pela ferramenta AJDT [9](*AspectJ Development Tools*). Após a análise será criado uma árvore sintática abstrata como resultado. A segunda parte é constituída da extração de meta-dados a partir dessa árvore sintática. Isso é feito através de AJ Visitors. E, assim, se completa a extração de dados. A figura 3.1 mostra o resumo do processo.

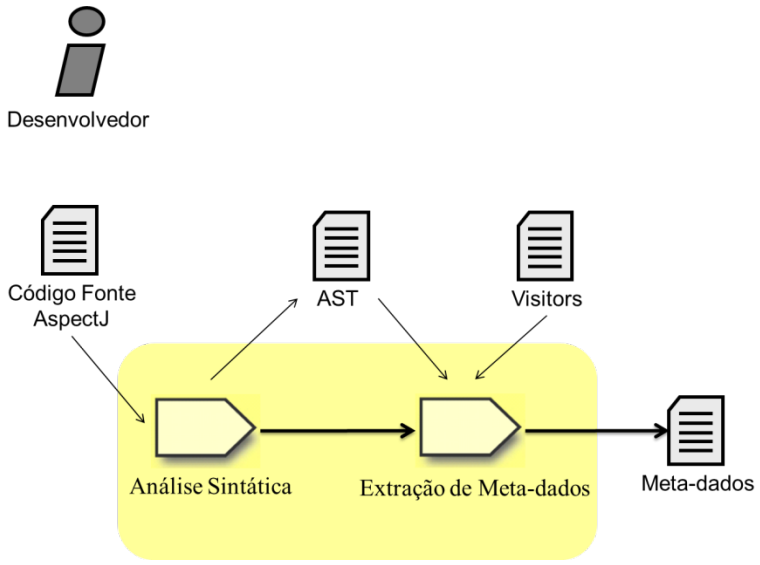

Figura 3.1 – Processo de extração de meta-dados.

O processo de extração de meta-dados trabalha em apenas um projeto por vez, sendo que somente arquivos com extensão *.aj* são processados. Também é necessário assegurar que o código não tenha erros de compilação, pois o processo é abortado caso algum erro seja encontrado no processo de compilação. No entanto, *warnings* são aceitos sem que haja interrupção do processo.

[9] http://www.eclipse.org/ajdt

A segunda etapa do trabalho é a realização de consultas utilizando a linguagem de consulta XQuery. A figura 3.2 apresenta os passos desta etapa.

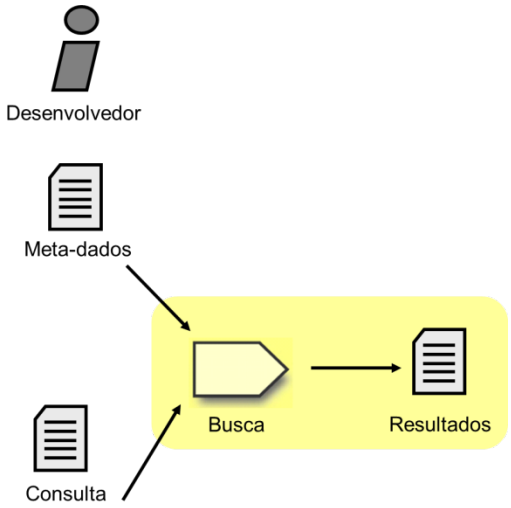

Figura 3.2 – Passos da busca de meta-dados.

A consulta ocorre com o repositório gerado no passo anterior, o arquivo XML com os meta-dados extraídos da árvore sintática abstrata. Esse arquivo XML é guardado localmente no mesmo diretório de onde é executado o aplicativo.

3.1 Ferramentas usadas no trabalho

A ferramenta base usada para realizar este trabalho foi o Eclipse[10]. Eclipse é uma plataforma de desenvolvimento de software, que foi iniciado pela IBM, e hoje é uma das plataformas mais usadas no mundo para programação em Java.

Junto com o Eclipse foi usado o AJDT, um *plugin* para o Eclipse que oferece vários recursos para a POA, entre eles a classe *AjASTVisitor* que faz a visita aos nós de uma árvore sintática abstrata de um programa em AspectJ.

3.2 Análise Sintática

A primeira parte do processo é executar a análise sintática do arquivo fonte recebido, em código de linguagem AspectJ. Essa análise é feita através do *parser* do AJDT.

Assim como o JDT possui unidades de compilação para arquivos .java, o AJDT tem unidades de compilação para arquivos .aj. A principal classe é a *AJCompilationUnitManager* que fornece métodos básicos para a compilação de um código em linguagem AspectJ, entre

[10] http://www.eclipse.org

eles, a criação de uma AST. A listagem abaixo mostra um exemplo de um programa usando AspectJ AST *parser*.

```
01 import java.util.*;
02 import org.aspectj.org.eclipse.jdt.core.dom.*;
03
04 public class Program {
05   public static void main(String[] argv) {
06     ASTParser parser = ASTParser.newParser(AST.JLS2);
07     parser.setCompilerOptions(new HashMap());
08     parser.setSource(argv[0].toCharArray());
09     CompilationUnit cu2 = (CompilationUnit) parser.createAST(null);
10     AjNaiveASTFlattener visitor = new AjNaiveASTFlattener();
11     cu2.accept(visitor);
12     System.err.println(visitor.getResult());
13   }
14 }
```

Listagem 3.1 – Programa usando AspectJ AST *parser*.

Na linha 06 é instanciado um novo *parser* do tipo *ASTParser*. Esse objeto é responsável por criar a AST. Na linha 08, o objeto recebe o arquivo fonte que será analisado. Por fim, na linha 09, a AST é criada e atribuída a um objeto do tipo *CompilationUnit*.

O *parser* percorre todo código fonte e através dele cria uma árvore sintática abstrata.

3.3 Extração de meta-dados

Com a análise sintática pronta, é feita a extração dos meta-dados. Esta extração é realizada a partir da classe *AjASTVisitor* que estende a classe *ASTVisitor* e é gerenciada pelo AJDT. A classe obedece ao padrão Visitor.

3.3.1 O padrão Visitor

O padrão *Visitor* resume-se a uma operação realizada em elementos de uma estrutura. Ele permite realizar uma operação sem, no entanto, afetar as classes onde atua. Por exemplo, se há a necessidade de analisar dados de uma linguagem que está representada em forma de árvore sintática abstrata, para descobrir o número de variáveis definidas, pode-se, então, usar um *Visitor* que passará por cada nó até o final, contando todas as variáveis definidas na árvore. Também é possível usar *Visitors* para analisar dados que possam ser usados para melhorar o desempenho do programa, evitar lançamentos de erros, e verificar se o programa está com uma segurança definida (GAMMA, HELM, *et al.*, 1995).

Dependendo da linguagem usada, a AST criada pode ter vários tipos de nós. Um nó de variáveis definidas, um nó com variáveis atribuídas a valores, outra com classes e assim por diante. Isso pode tornar a hierarquia complexa, tornando-a difícil de manter e atualizar.

Poderia ainda ficar pior se fosse adicionada uma nova operação, o que tornaria necessária a recompilação de todas as classes do programa.

Porém, ao separar as novas operações de cada classe em diferentes objetos, o problema pode ser contornado. Esses objetos são os *Visitors* ou os visitantes. Cada elemento tem um método que aceita os visitantes. O método tem um argumento que representa o tipo do objeto que deve fazer a visita. Isso faz com que o método execute as operações correspondentes ao tipo da classe que foi passado no argumento do visitante (GAMMA, HELM, *et al.*, 1995). A figura 3.3 mostra a estrutura básica de um programa que usa o padrão Visitor.

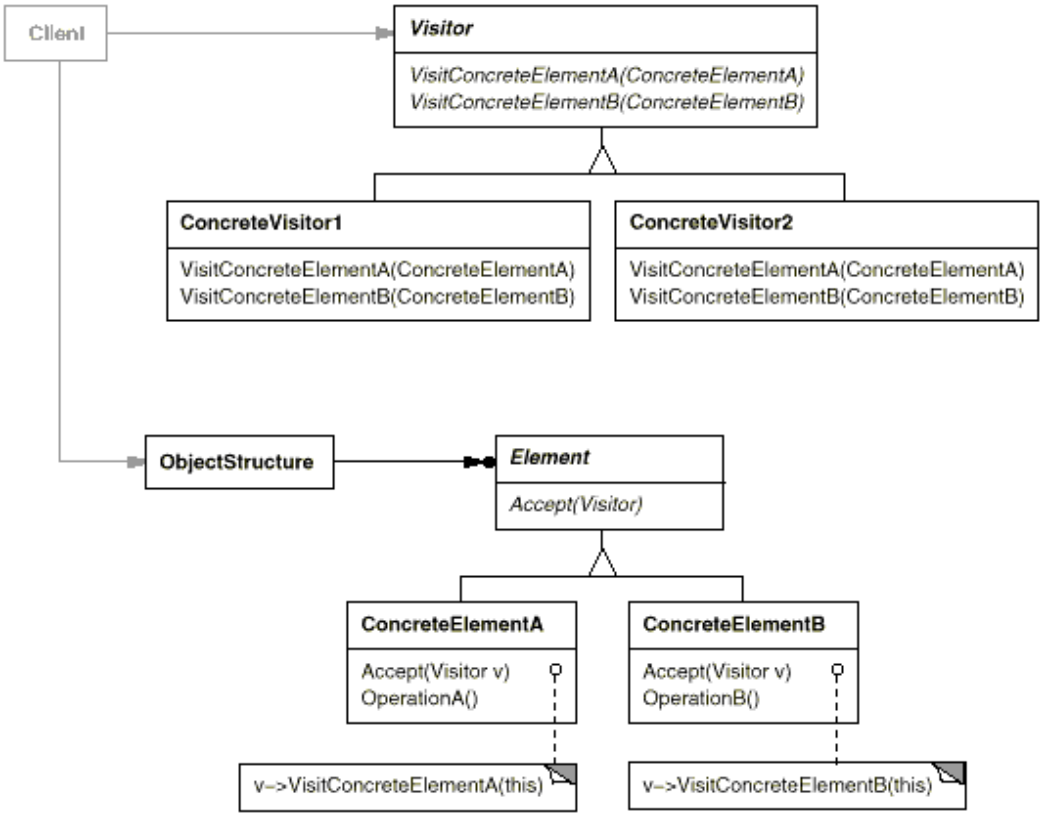

Figura 3.3 – Estrutura que usa Visitantes (GAMMA, HELM, *et al.*, 1995).

A interface principal da hierarquia do projeto, que no caso da figura 3.3 é a *Element(),* deve declarar um método *accept(),* o que obriga todas as classes elementos da interface *Element()* a implementar esse método. O método *accept()* deve receber como argumento um visitante.

Também é necessária a criação de uma nova interface que representará os visitantes. Normalmente, essa interface é chamada de *Visitor()* e ela deve declarar métodos *visit()* para

cada tipo de elemento presente da hierarquia. Para isso, cada método *visit()* deve ter um argumento representando o tipo de classe que deseja fazer a operação. Assim, toda nova operação deve implementar esta interface e seu método *visit()* para visitar a função correspondente ao tipo adequado.

Tendo essa estrutura definida, basta declarar os tipos e operações desejados. Então, para realizar as operações, é chamado o método *accept()* dos tipos passando como argumento as operações desejadas.

Salienta-se que se for necessário incluir novas operações, basta declará-las como novas subclasses da interface *Visitor()*, o que se tornaria mais complexo sem um padrão de comportamento como o padrão *Visitor*.

O padrão *Visitor* é suportado pelo projeto AJDT do Eclipse. Ele percorre e faz uma visita a cada nó da árvore sintática gerada no passo anterior. Em cada visita é extraída as principais informações sobre o nó. Após o último nó ser visitado é gerado um arquivo com todos os meta-dados extraídos pelo *Visitors*.

O arquivo gerado pelo *Visitor* é um arquivo XML, que contém todos os dados em ordem hierárquica, o que torna as informações organizadas e fáceis de serem usadas para a busca de oportunidades de refatoração.

3.4 Busca

Nesta segunda etapa do trabalho, é feita a busca de informações a partir do resultado da extração dos meta-dados. A busca é realizada através da linguagem de consulta XQuery.

3.4.1 Escopo da busca

Um código fonte de um programa de médio ou grande porte pode trazer uma grande quantidade de informações, muitas delas úteis, outras nem tanto, como por exemplo os comentários do programa. Assim, definiu-se uma granularidade baixa, na qual os dados serão pesquisados, visto que uma granularidade muito fina implicaria em uma variedade de dados que não são pertinentes ao objetivo do trabalho.

Como o trabalho tem por objetivo mostrar como funciona o processo de extração de meta-dados e não tanto no que é pesquisado, foi definida uma granularidade baixa da busca de dados. Porém, para trazer uma lista de informações mais interessante ao desenvolvedor, seria preciso um estudo mais criterioso sobre as principais informações de um código fonte de

programa em AspectJ. As informações que foram pesquisadas no experimento do trabalho realizado são:

- declarações de Aspectos;
- declarações de pontos de atuação e detalhes;
- declarações de adendos e o tipo;
- declarações de inter-tipos e fragmentos.

3.4.2 Consultas

Como forma de exemplificar uma execução de busca de dados em um arquivo XML, foram definidas algumas consultas em XQuery:

- //codigo/Aspectos/Aspecto/nome
- //codigo/Aspectos/Aspecto/PointCuts/PointCut/nome
- //codigo/Aspectos/Aspecto/PointCuts/PointCut/detalhes
- //codigo/Aspectos/Aspecto/Advices/Advice/tipo
- //codigo/Aspectos/Aspecto/InterTypes/InterType/tipo
- //codigo/Aspectos/Aspecto/InterTypes/InterType/Fragmentos/Fragmento

4 ESTUDO DE CASO

Como parte deste trabalho foi realizada a implementação de um pequeno aplicativo que apresenta o processo de extração de meta-dados e buscas.

4.1 Preparação

O primeiro passo realizado foi a instalação da plataforma de programação Eclipse. Essa ferramenta pode ser baixada no site da fundação Eclipse: http://www.eclipse.org. Com o Eclipse instalado, foi instalado o *plugin* AJDT para Eclipse, que também pode ser encontrado no site da fundação Eclipse. Foi instalado também o *plugin* JAXB, que tem como objetivo mapear dados de objetos em um arquivo XML. Por fim, foi instalado o *plugin* XQDT, outro recurso que pode ser obtido do mesmo site e que é utilizado para editar, executar e depurar consultas XQuery.

4.2 O aplicativo

O aplicativo criado para testar os processos estudados neste trabalho é uma simples janela que permite abrir um arquivo do tipo AspectJ. Ele também foi implementado usando o Eclipse.

4.2.1 Dependências

Para a compilação do aplicativo é necessário que as seguintes dependências sejam incluídas no *build path* do projeto:

- *ajde.jar*;
- *aspectjrt.jar*;
- *aspectjweaver.jar*;
- *org.eclipse.core.jobs_XXXX.jar;*
- *org.eclipse.core.resources_XXXX.jar;*
- *org.eclipse.equinox.common_XXXX.jar;*
- *org.eclipse.wst.xquery.core_XXXX.jar;*
- *XQEngine.jar;*
- *xqjapi.jar.*

Onde XXXX é a versão da dependência.

4.2.2 Usando o aplicativo

Ao escolher um arquivo para abri-lo, seu conteúdo é listado na área central do aplicativo. A figura 4.1 mostra o aplicativo com um arquivo aberto.

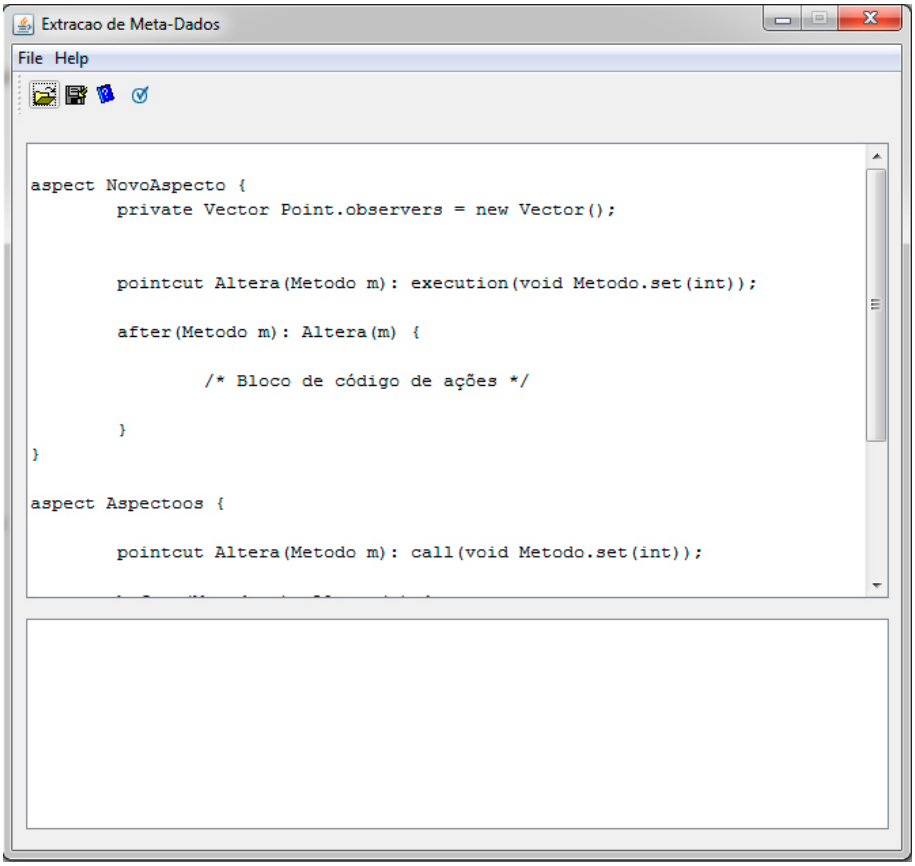

Figura 4.1 – Arquivo ApectJ aberto no aplicativo.

Com o arquivo aberto, pode-se alterar o código na tela, porém não há possibilidade de salvar a mudança em arquivo. Basta clicar no botão “Extrair” para começar o processo de extração de meta-dados.

4.2.3 Saída

O resultado final das consultas será informado na janela inferior do aplicativo, de acordo com a consulta selecionada na lista de opções, logo acima da janela inferior. No caso da consulta resultar em mais de um item, cada item é mostrado em uma linha na janela.

4.3 O código-fonte do processo de extração

Com o arquivo selecionado no aplicativo, é necessário construir a AST do código fornecido. Esse processo está representado no método *check* da classe *Visitar*. A Listagem 4.1 apresenta o trecho de código responsável por criar a AST.

```
17 public class Visitar {
18   public String check(String source) throws JAXBException, IOException {
19     ASTParser parser = ASTParser.newParser(AST.JLS2);
20     parser.setCompilerOptions(new HashMap());
21     parser.setSource(source.toCharArray());
22     CompilationUnit cu2 = (CompilationUnit) parser.createAST(null);
```

Listagem 4.1 – Trecho de código do arquivo Visitar.java.

Na linha 19, é instanciado um novo *parser* do tipo ASTParser. Na linha 21, o parser recebe o código fonte que será analisado. E na linha 22 é criada a AST. O nó raiz da árvore é atribuído a uma variável do tipo *CompilationUnit*. Essa variável possui um método *accept* que recebe um visitante para fazer a visita em toda a árvore.

O processo de extração foi desenvolvido com a ajuda da classe *AjASTVisitor*, presente no *plugin* AJDT. Essa classe fornece todos os recursos necessários para percorrer pela árvore sintática abstrata.

Foi criada uma classe chamada *AspectASTVisitor*, que estende a classe *AjASTVisitor*. Por ser uma classe visitante, ela deve implementar todos os métodos *visit()* desejados para fazer a visita dos nós. A Listagem 4.2 mostra trecho da classe *AspectASTVisitor* bem como a implementação dos métodos *visit()*.

```
024 public class AspectASTVisitor extends AjASTVisitor {
025
026   public AspectJ aj = new AspectJ();
027   public AjAspecto aspecto;
028
029   public AspectASTVisitor() {}
030
031   public boolean visit(TypeDeclaration no) {
032     if (((AjTypeDeclaration) no).isAspect()) {
033       this.visit((AspectDeclaration) no);
034     }
035     return true;
036   }
037
038   public boolean visit(AspectDeclaration no) {
039     aspecto = new AjAspecto();
040     aspecto.setNome(no.getName().toString());
041     aj.as.add(aspecto);
042
043     return true;
044   }
045
046   public boolean visit(PointcutDeclaration no) {
047     AjPointcut pc = new AjPointcut();
048     pc.setNome(no.getName().toString());
049
050     PointcutDesignator pd = no.getDesignator();
051     String s="";
052     if (pd instanceof DefaultPointcut) {
053       s = s + ((DefaultPointcut) pd).getDetail();
054     }
055
```

```
056     pc.setDetalhes(s);
057     aspecto.pc.add(pc);
058
059     return true;
060   }
061
062   public boolean visit(AdviceDeclaration no) {
063     AjAdvice ad = new AjAdvice();
064     ad.setTipo("default");
065     aspecto.ad.add(ad);
066
067     return true;
068   }
069
070   public boolean visit(AroundAdviceDeclaration no) {
071     AjAdvice ad = new AjAdvice();
072     ad.setTipo("around");
073     aspecto.ad.add(ad);
074
075     return true;
076   }
077
078   public boolean visit(BeforeAdviceDeclaration no) {
079     AjAdvice ad = new AjAdvice();
080     ad.setTipo("before");
081     aspecto.ad.add(ad);
082
083     return true;
084   }
085
086   public boolean visit(AfterAdviceDeclaration no) {
087     AjAdvice ad = new AjAdvice();
088     ad.setTipo("after");
089     aspecto.ad.add(ad);
090
091     return true;
092   }
093
094   public boolean visit(AfterThrowingAdviceDeclaration no) {
095     AjAdvice ad = new AjAdvice();
096     ad.setTipo("afterThrowing");
097     aspecto.ad.add(ad);
098
099     return true;
100   }
101
102   public boolean visit(AfterReturningAdviceDeclaration no) {
103     AjAdvice ad = new AjAdvice();
104     ad.setTipo("afterReturning");
105     aspecto.ad.add(ad);
106
107     return true;
108   }
109
110   public boolean visit(InterTypeFieldDeclaration no) {
111     List<FieldDeclaration> fd = no.fragments();
112     AjInterType its = new AjInterType();
113     its.setTipo("InterTypeFieldDeclaration");
114
115     for (int i=0; i < fd.size(); i++) {
116       its.fd.add(""+fd.get(i));
117     }
118
119     aspecto.it.add(its);
120
121     return true;
122   }
123
```

```
124   public boolean visit(InterTypeMethodDeclaration no) {
125     AjInterType its = new AjInterType();
126     its.setTipo("InterTypeMethodDeclaration");
127     aspecto.it.add(its);
128
129     return true;
130   }
131 }
```

Listagem 4.2 – Trecho de código do arquivo AjASTVisitor.java.

Os tipos de objetos que os métodos *visit()* aceitam são:

- *TypeDeclaration*: declarações de tipos. No método que possui esse argumento, é checado se o tipo de declaração é um aspecto. Em caso positivo, uma nova chamada ao método *visit*() é feita, desta vez com o parâmetro do tipo *AspectDeclaration*;
- *AspectDeclaration*: declarações de aspectos. Nesse método é salvo o nome da declaração de aspecto;
- *PointcutDeclaration*: declaração de um ponto de ação. O método salva o nome e verifica se o tipo do ponto de ação é um tipo padrão. Caso afirmativo é salvo o tipo da declaração;
- *AdviceDeclaration*, *AroundAdviceDeclaration*, *AfterAdviceDeclaration*, *BeforeAdviceDeclaration*, *AfterThrowingAdviceDeclaration* e *AfterReturningAdviceDeclaration*: declarações de adendos. Nesses métodos é salvo o tipo de adendo declarado;
- *InterTypeFieldDeclaration*: declaração de variáveis inter-tipos. Nesse método é salvo o tipo de declaração inter-tipo;
- *InterTypeMethodDeclaration*: declaração de métodos inter-tipos. O método salva o tipo de declaração.

Para representar os dados extraídos da AST foi criada uma abstração desses dados em classes. O código da Listagem 4.3 mostra essas classes.

```
07 @XmlRootElement(name = "codigo")
08 @XmlAccessorType(XmlAccessType.PUBLIC_MEMBER)
09 public class AspectJ {
10
11   @XmlElementWrapper(name="Aspectos")
12   @XmlElement(name="Aspecto")
13   public List<AjAspecto> as;
14
15   public AspectJ() {
16     as = new ArrayList<AjAspecto>();
17   }
18 }
```

Listagem 4.3 – Trecho de código do arquivo AspectJ.java.

A classe *AspectJ* representa um arquivo em AspectJ. Ela possui uma lista de aspectos. Nas linhas 07, 08, 11 e 12 é feita a anotação para posteriormente salvar os dados em um arquivo XML. Na linha 16 a lista de aspectos é instanciada.

A Listagem 4.4 mostra um trecho de código da classe que representa um aspecto.

```
07 @XmlRootElement(name="Aspectos")
08 @XmlType(propOrder = {"nome", "pc", "ad", "it"})
09 public class AjAspecto {
10
11   private String nome;
12
13   @XmlElementWrapper(name="PointCuts")
14   @XmlElement(name="PointCut")
15   public List<AjPointcut> pc;
16
17   @XmlElementWrapper(name="Advices")
18   @XmlElement(name="Advice")
19   public List<AjAdvice> ad;
20
21   @XmlElementWrapper(name="InterTypes")
22   @XmlElement(name="InterType")
23   public List<AjInterType> it;
24
25   public AjAspecto() {
26     pc = new ArrayList<AjPointcut>();
27     ad = new ArrayList<AjAdvice>();
28     it = new ArrayList<AjInterType>();
29   }
```

Listagem 4.4 – Trecho de código do arquivo AjAspecto.java.

A classe que representa um aspecto possui uma lista de pontos de atuação (linha 15), outra de adendos (linha 19) e uma de declarações inter-tipos (linha 23). Acima de cada declaração dos atributos e da declaração da classe há a anotação para salvar os dados em XML.

A Listagem 4.5 apresenta um trecho de código que representa um ponto de ação.

```
07 @XmlRootElement(name="PointCuts")
08 @XmlAccessorType(XmlAccessType.FIELD)
09 public class AjPointcut {
10
11   private String nome;
12
13   private String detalhes;
14
15   public AjPointcut() {
16     // TODO Auto-generated constructor stub
17   }
```

Listagem 4.5 – Trecho de código do arquivo AjPointcut.java.

A classe *AjPointcut* possui um atributo que representa o nome do ponto de ação (linha 11) e um atributo para os detalhes (linha 13).

O trecho de código apresentado na Listagem 4.6 representa uma declaração inter-tipo.

```
12 @XmlRootElement(name="InterTypes")
13 @XmlAccessorType(XmlAccessType.PUBLIC_MEMBER)
14 @XmlType(propOrder = {"tipo", "fd"})
15 public class AjInterType {
16
17   private String tipo;
18
19   @XmlElementWrapper(name="Fragmentos")
20   @XmlElement(name="Fragmento")
21   public List<String> fd;
22
23   public AjInterType() {
24     // TODO Auto-generated constructor stub
25     fd = new ArrayList<String>();
26   }
```

Listagem 4.6 – Trecho de código do arquivo AjInterType.java.

Uma declaração inter-tipo pode ter vários fragmentos. Por isso, a classe *AjInterType* tem um atributo para salvar o tipo da declaração (linha 17) e uma lista de fragmentos (linha 21).

A representação dos adendos pode ser visualizada na Listagem 4.7.

```
05 @XmlRootElement(name="Advices")
06 public class AjAdvice {
07
08   private String tipo;
09
10   public AjAdvice() {
11     // TODO Auto-generated constructor stub
12   }
```

Listagem 4.7 – Trecho de código do arquivo AjAdvice.java.

A classe *AjAdvice* possui um atributo para salvar o tipo do adendo (linha 08).

4.4 Resultado da extração

À medida que os nós são visitados, as informações extraídas são guardadas na memória através das classes de representação. Então, esses dados são transferidos para um arquivo XML com a ajuda da ferramenta JAXB. Ela funciona com um sistema de anotação das classes que devem ser mapeadas em XML.

A Listagem 4.8 representa um código AspectJ que é usado para extrair meta-dados.

```
01 aspect NovoAspecto {
02   private Vector Point.observers = new Vector();
03
04   pointcut Altera(Metodo m): execution(void Metodo.set(int));
05
06   after(Metodo m): Altera(m) {
07
08     /* Bloco de código de ações */
09
10   }
11 }
12
13 aspect Aspectos {
14
15   pointcut Atualiza(OutroMetodo m): call(void OutroMetodo.set(int));
16
17   before(OutroMetodo m): Atualiza(m) {
18
19     /* Bloco de código de ações */
20     String st = "Teste";
21
22   }
23 }
```

Listagem 4.8 – Arquivo de entrada em AspectJ.

O resultado da extração pode ser visto no arquivo XML resultante na Listagem 4.9.

```
01 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
02 <codigo>
03   <Aspectos>
04     <Aspecto>
05       <nome>NovoAspecto</nome>
06       <PointCuts>
07         <PointCut>
08           <nome>Altera</nome>
09           <detalhes>execution(void Metodo.set(int))</detalhes>
10         </PointCut>
11       </PointCuts>
12       <Advices>
13         <Advice>
14           <tipo>after</tipo>
15         </Advice>
16       </Advices>
17       <InterTypes>
18         <InterType>
19           <tipo>InterTypeFieldDeclaration</tipo>
20           <Fragmentos>
21             <Fragmento>observers=new Vector()</Fragmento>
22           </Fragmentos>
23         </InterType>
24       </InterTypes>
25     </Aspecto>
26     <Aspecto>
27       <nome>Aspectos</nome>
28       <PointCuts>
29         <PointCut>
30           <nome>Atualiza</nome>
31           <detalhes>call(void OutroMetodo.set(int))</detalhes>
32         </PointCut>
33       </PointCuts>
34       <Advices>
35         <Advice>
```

```
36            <tipo>before</tipo>
37          </Advice>
38        </Advices>
39        <InterTypes />
40      </Aspecto>
41    </Aspectos>
42  </codigo>
```

Listagem 4.9 – Resultado em XML da extração de dados.

Depois dos dados terem sido salvos em um arquivo XML, a busca pode ser realizada. Essa busca está representada na classe *Busca*. O trecho de código onde a busca é feita é mostrada na Listagem 4.10.

```
50      XQEngine xqEngine = new XQEngine();
51      xqEngine.setMinIndexableWordLength(5);
52
53      SAXParserFactory factory = SAXParserFactory.newInstance();
54      try {
55        SAXParser saxParser = factory.newSAXParser();
56        XMLReader xmlReader = saxParser.getXMLReader();
57        xqEngine.setXMLReader(xmlReader);
58
59        xqEngine.setDocument(arquivo);
60
61        ResultList result = xqEngine.setQuery(expressao);
62
63        resultado = result.emitXml(true);
```

Listagem 4.10 – Trecho de código do arquivo Busca.java.

Na linha 50 é instanciado o objeto que fará a consulta propriamente dita. Porém, esse objeto precisa definir um *parser* para converter o XML para a memória. Esses passos são feitos nas linhas 53 a 59. Na linha 61, o objeto *xqEngine* faz a consulta e atribui o resultado em uma variável do tipo *ResultList*. Então, na linha 63, o resultado é formatado e salvo na variável *resultado*.

Um exemplo de consulta que busca todos os adendos do tipo *before* de um aspecto pode ser vista na Listagem 4.11.

```
01  for $aspecto in doc("saida.xml")/codigo/Aspectos/Aspecto[nome=“Aspectos“]
02  return $aspecto/Advices/Advice[tipo=“before“]
```

Listagem 4.11 – Exemplo de uma consulta em XQuery.

5 CONCLUSÃO

A refatoração de códigos orientados a aspectos oferece uma ferramenta de apoio para os desenvolvedores em seus trabalhos. Porém, essa refatoração ainda é pouco explorada em comparação com outras linguagens, como Java. Uma das causas dessa pouca exploração pode ser pela recente inclusão do paradigma orientado a aspectos no mercado. Os programadores ainda se sentem inseguros em usar essa nova tecnologia, por ter pouco suporte e poucos especialistas no assunto.

Porém, deve-se tomar cuidado ao mudar o código-fonte, pois o programa não deve mudar o seu comportamento em virtude da refatoração. Por isso é preciso analisar se realmente a refatoração irá trazer mais benefícios do que prejuízos. E para tornar essa análise mais conclusiva, uma boa amostra dos dados do código é fundamental.

Este trabalho ressaltou que uma refatoração que gere poucos riscos depende de boas informações a respeito do código. Esses dados podem ser obtidos através de processos de extração de meta-dados e busca em códigos orientados a aspectos. O trabalho desenvolvido mostrou um exemplo de um processo de extração de meta-dados e busca em linguagens orientadas a aspectos.

Também nota-se que existem várias alternativas à extração e busca de meta-dados, como por exemplo, a reflexão, que pode ser muito útil para extrair dados em tempo de execução.

Como trabalhos futuros, sugere-se um estudo e implementação de uma nova linguagem de busca de dados, o que poderia tornar uma busca mais específica para a tarefa de obter dados de um código-fonte. Também pode-se sugerir um estudo sobre a refatoração a partir dos dados buscados neste trabalho, e analisar o desempenho e qualidade da refatoração em cima desses dados.

6 BIBLIOGRAFIA

ARSENOVSKI, D. **Esclarecendo os Equívocos Mais Comuns Sobre Refatoração**, 2009. Disponível em: <http://www.infoq.com/br/articles/RefactoringMyths>. Acesso em: 15 maio 2012.

BOS, J. V. D. **Refactoring (in) Eclipse**. Universiteit van Amsterdam. [S.l.]. 2008.

DE MELO JUNIOR, L. S. **Uma Estratégia de Refatoração para AspectJ utilizando Leis de Programação e XML**. Universidade de Fortaleza. Fortaleza, p. 100. 2007.

FERNANDES, A. P. Home Page de Acauan. **Reflexão computacional**, 2000. Disponível em: <http://attila.urcamp.tche.br/~acauan/art_ccei_rc.html>. Acesso em: 20 maio 2012.

FILMAN, R. E. et al. **Aspect-Oriented Software Development**. 1. ed. [S.l.]: Addison-Wesley, 2004.

FORMAN, I. R.; FORMAN, N. **Java Reflection In Action**. [S.l.]: Manning, 2005.

FOWLER, M. et al. **Refactoring:** Improving the Design of Existing Code. 1. ed. [S.l.]: Addison-Wesley, 2002.

FURTADO JÚNIOR, M. B. **Tutorial XML**. Disponível em: <http://www.gta.ufrj.br/grad/00_1/miguel/index.html>. Acesso em: 20 maio 2012.

GABRICK, K.; WEISS, D. **J2EE and XML Development**. 1. ed. Greenwich: Manning Publications Co., 2002.

GAMMA, E. et al. **Design Patterns:** Elements of Reusable Object-Oriented Software. [S.l.]: Addison-Wesley, 1995.

JÚNIOR, A. C. D. S. **Refatoração de Sistemas Orientados a Objetos Para Auxiliar A Construção de um Testbed para Manutenção de Software Orientado a Aspectos**. Universidade Federal De Pernambuco. Recife, p. 37. 2011.

KICZALES, G. et al. **Aspect-Oriented Programming**. [S.l.]. 1997.

KISELEV, I. **Aspect-Oriented Programming with AspectJ**. Indianapolis: Sams Publishing, 2003.

LADDAD, R. **AspectJ In Action**. 2. ed. Greenwich: Manning Publications Co., 2010.

MCGOVERN, J. et al. **XQuery Kick Start**. 1ª. ed. [S.l.]: Sams Publishing, 2003.

SMITH, B. C. **Procedural reflection in programming languages**. Massachusetts Institute of Technology. [S.l.]. 1982.

SOARES, G.; GHEYI, R.; MASSONI, T. **A Catalog of Refactoring Bugs in Eclipse**. Universidade Federal de Campina Grande. [S.l.].

SOBEL, J. M. **An Introduction to Reflection-Oriented Programming**. Indiana University. Bloomington, p. 20. 1996.

THE ASPECTJ TEAM. **The AspectJTM Programming Guide**, 2002. Disponível em: <http://www.eclipse.org/aspectj/doc/released/progguide/index.html>. Acesso em: 23 maio 2012.

THE ECLIPSE FOUNDATION. **The Eclipse Foundation open source community website**. Disponível em: <http://eclipse.org/>. Acesso em: 23 set. 2012.

THE ECLIPSE FOUNDATION. **The Eclipse Foundation open source community website**. Disponível em: <http://eclipse.org/>. Acesso em: 23 set. 2012.

THE JAVA TUTORIALS. **Trail:** The Reflection API. Disponível em: <http://docs.oracle.com/javase/tutorial/reflect/>. Acesso em: 20 maio 2012.

W3SCHOOLS.COM. **XML Tutorial**. Disponível em: <http://www.w3schools.com/xml/default.asp>. Acesso em: 20 maio 2012.

WALMSLEY, P. **XQuery:** Search Across a Variety of XML Data. 1ª. ed. [S.l.]: O'Reilly Media, 2007.

APÊNDICE A – ARQUIVO PRINCIPAL.JAVA

```java
package extracao;

import extracao.janela.Janela;
import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.UIManager;
import java.awt.Dimension;
import java.awt.Toolkit;

public class Principal {
  public Principal() {
    Frame frame = new Janela();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2,
        (screenSize.height - frameSize.height) / 2);
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
    frame.setVisible(true);
  }

  /**
   *
   * @param args
   */
  public static void main(String[] args) {
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
      e.printStackTrace();
    }
    new Principal();
  }
}
```

APÊNDICE B – ARQUIVO BUSCA.JAVA

```java
package extracao.busca;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.XMLReader;
import com.fatdog.xmlEngine.ResultList;
import com.fatdog.xmlEngine.XQEngine;

public class Busca {

  private static final String[] expressoes = {
      "//codigo/Aspectos/Aspecto/nome",
      "//codigo/Aspectos/Aspecto/PointCuts/PointCut/nome",
      "//codigo/Aspectos/Aspecto/PointCuts/PointCut/detalhes",
      "//codigo/Aspectos/Aspecto/Advices/Advice/tipo",
      "//codigo/Aspectos/Aspecto/InterTypes/InterType/tipo",
      "//codigo/Aspectos/Aspecto/InterTypes/InterType/Fragmentos/Fragmento" };

  public Busca() {
    // TODO Auto-generated constructor stub
  }

  public String consulta(int index) {
    String resultado = "";
    String expressao = "";

    switch (index) {
    case 0:
      expressao = expressoes[0];
      break;
    case 1:
      expressao = expressoes[1];
      break;
    case 2:
      expressao = expressoes[2];
      break;
    case 3:
      expressao = expressoes[3];
      break;
    case 4:
      expressao = expressoes[4];
      break;
    case 5:
      expressao = expressoes[5];
      break;
    }

    String arquivo = "Resultado.xml";

    XQEngine xqEngine = new XQEngine();
    xqEngine.setMinIndexableWordLength(5);

    SAXParserFactory factory = SAXParserFactory.newInstance();
    try {
      SAXParser saxParser = factory.newSAXParser();
      XMLReader xmlReader = saxParser.getXMLReader();
      xqEngine.setXMLReader(xmlReader);

      xqEngine.setDocument(arquivo);

      ResultList result = xqEngine.setQuery(expressao);

      resultado = result.emitXml(true);

    } catch (Exception e) {
```

```
66          System.err.println("Could not set a XML Reader for the xqEngine:" +
e.getMessage());
67     }
68     return resultado;
69   }
70 }
```

APÊNDICE C – ARQUIVO AJADVICE.JAVA

```java
package extracao.dados;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="Advices")
public class AjAdvice {

  private String tipo;

  public AjAdvice() {
    // TODO Auto-generated constructor stub
  }

  /**
   * @return the tipo
   */
  public String getTipo() {
    return tipo;
  }

  /**
   * @param tipo the tipo to set
   */
  public void setTipo(String tipo) {
    this.tipo = tipo;
  }
}
```

APÊNDICE D – ARQUIVO AJASPECTO.JAVA

```java
package extracao.dados;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.*;

@XmlRootElement(name="Aspectos")
@XmlType(propOrder = {"nome", "pc", "ad", "it"})
public class AjAspecto {

  private String nome;

  @XmlElementWrapper(name="PointCuts")
  @XmlElement(name="PointCut")
  public List<AjPointcut> pc;

  @XmlElementWrapper(name="Advices")
  @XmlElement(name="Advice")
  public List<AjAdvice> ad;

  @XmlElementWrapper(name="InterTypes")
  @XmlElement(name="InterType")
  public List<AjInterType> it;

  public AjAspecto() {
    pc = new ArrayList<AjPointcut>();
    ad = new ArrayList<AjAdvice>();
    it = new ArrayList<AjInterType>();
  }

  /**
   * @return the nome
   */
  public String getNome() {
    return nome;
  }

  /**
   * @param nome the nome to set
   */
  public void setNome(String nome) {
    this.nome = nome;
  }
}
```

APÊNDICE E – ARQUIVO AJINTERTYPE.JAVA

```java
package extracao.dados;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name="InterTypes")
@XmlAccessorType(XmlAccessType.PUBLIC_MEMBER)
@XmlType(propOrder = {"tipo", "fd"})
public class AjInterType {

  private String tipo;

  @XmlElementWrapper(name="Fragmentos")
  @XmlElement(name="Fragmento")
  public List<String> fd;

  public AjInterType() {
    // TODO Auto-generated constructor stub
    fd = new ArrayList<String>();
  }

  /**
   * @return the tipo
   */
  public String getTipo() {
    return tipo;
  }

  /**
   * @param tipo the tipo to set
   */
  public void setTipo(String tipo) {
    this.tipo = tipo;
  }
}
```

APÊNDICE F – ARQUIVO AJPOINTCUT.JAVA

```java
package extracao.dados;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="PointCuts")
@XmlAccessorType(XmlAccessType.FIELD)
public class AjPointcut {

  private String nome;

  private String detalhes;

  public AjPointcut() {
    // TODO Auto-generated constructor stub
  }

  /**
   * @return the nome
   */
  // @XmlElement(name="Nombre")
  public String getNome() {
    return nome;
  }

  /**
   * @param nome the nome to set
   */
  public void setNome(String nome) {
    this.nome = nome;
  }

  /**
   * @return the detalhes
   */
  public String getDetalhes() {
    return detalhes;
  }

  /**
   * @param detalhes the detalhes to set
   */
  public void setDetalhes(String detalhes) {
    this.detalhes = detalhes;
  }
}
```

APÊNDICE G – ARQUIVO ASPECTJ.JAVA

```java
package extracao.dados;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.*;

@XmlRootElement(name = "codigo")
@XmlAccessorType(XmlAccessType.PUBLIC_MEMBER)
public class AspectJ {

  @XmlElementWrapper(name="Aspectos")
  @XmlElement(name="Aspecto")
  public List<AjAspecto> as;

  public AspectJ() {
    as = new ArrayList<AjAspecto>();
  }
}
```

APÊNDICE H – ARQUIVO JANELA.JAVA

```
001 package extracao.janela;
002
003 import extracao.busca.Busca;
004 import extracao.utils.SCFiltro;
005 import extracao.visitors.Visitar;
006
007 import java.util.logging.Level;
008 import java.util.logging.Logger;
009 import javax.swing.JFrame;
010 import java.awt.BorderLayout;
011 import javax.swing.JPanel;
012 import javax.swing.JLabel;
013 import javax.swing.ImageIcon;
014 import javax.swing.JButton;
015 import javax.swing.JToolBar;
016 import javax.swing.JMenu;
017 import javax.swing.JMenuBar;
018 import javax.swing.JMenuItem;
019 import java.awt.Dimension;
020 import java.awt.event.ActionEvent;
021 import java.awt.event.ActionListener;
022 import javax.swing.JOptionPane;
023 import java.awt.Rectangle;
024 import javax.swing.JScrollPane;
025 import javax.swing.JTextArea;
026 import javax.swing.JFileChooser;
027 import java.io.*;
028 import java.awt.Font;
029 import javax.swing.JComboBox;
030 import javax.swing.DefaultComboBoxModel;
031
032 public class Janela extends JFrame {
033
034                             private      ImageIcon      imageHelp      =      new
ImageIcon(Janela.class.getResource("help.gif"));
035                            private      ImageIcon      imageClose      =      new
ImageIcon(Janela.class.getResource("closefile.gif"));
036                             private      ImageIcon      imageOpen      =      new
ImageIcon(Janela.class.getResource("openfile.gif"));
037                          private      ImageIcon      imageCompilar      =      new
ImageIcon(Janela.class.getResource("tip.gif"));
038     private JButton buttonHelp = new JButton();
039     private JButton buttonClose = new JButton();
040     private JButton buttonCompilar = new JButton();
041     private JButton buttonOpen = new JButton();
042     private JToolBar toolBar = new JToolBar();
043     private JLabel statusBar = new JLabel();
044     private JMenuItem menuHelpAbout = new JMenuItem();
045     private JMenu menuHelp = new JMenu();
046     private JMenuItem menuFileExit = new JMenuItem();
047     private JMenu menuFile = new JMenu();
048     private JMenuBar menuBar = new JMenuBar();
049     private JPanel panelCenter = new JPanel();
050     private BorderLayout layoutMain = new BorderLayout();
051     private JScrollPane jScrollPane1 = new JScrollPane();
052     private JTextArea jTextArea1 = new JTextArea();
053     private JComboBox comboBox = new JComboBox();
054
055     /* filtro na opcao
056      * para abrir documentos */
057     public static final JFileChooser FILE_CHOOSER = new JFileChooser();
058     private static File file = null;
059     private static Janela singleton = null;
060
061     private BufferedReader arq;
```

```
062     private JTextArea jTextArea2 = new JTextArea();
063     private JScrollPane jScrollPane2 = new JScrollPane();
064     private JTextArea jTAErro = new JTextArea();
065
066     private StringReader arq1;
067
068     public Janela() {
069         try {
070             jbInit();
071         } catch(Exception e) {
072             e.printStackTrace();
073         }
074     }
075
076     private void jbInit() throws Exception {
077
078         this.setJMenuBar(menuBar);
079         this.getContentPane().setLayout(layoutMain);
080         panelCenter.setLayout(null);
081         jScrollPane1.setBounds(new Rectangle(10, 20, 635, 335));
082         jTextArea2.setBounds(new Rectangle(0, 0, 0, 17));
083         jScrollPane2.setBounds(new Rectangle(10, 370, 635, 155));
084         jScrollPane2.setFont(new Font("Courier New", 0, 8));
085         jTAErro.setFont(new Font("Courier New", 0, 11));
086         FILE_CHOOSER.addChoosableFileFilter(new SCFiltro());
087         this.setSize(new Dimension(660, 620));
088         this.setTitle("Extracao de Meta-Dados");
089         this.setResizable(false);
090         menuFile.setText("File");
091         menuFileExit.setText("Sair");
092         menuFileExit.addActionListener(new ActionListener() {
093             public void actionPerformed(ActionEvent ae) {
094                 fileExit_ActionPerformed(ae);
095             }
096         });
097         menuHelp.setText("Help");
098         menuHelpAbout.setText("About");
099         menuHelpAbout.addActionListener(new ActionListener() {
100             public void actionPerformed(ActionEvent ae) {
101                 helpAbout_ActionPerformed(ae);
102             }
103         });
104         statusBar.setText("");
105         buttonOpen.setToolTipText("Open File");
106         buttonOpen.setIcon(imageOpen);
107         buttonOpen.addActionListener(new ActionListener() {
108             public void actionPerformed(ActionEvent e) {
109                 buttonOpen_actionPerformed(e);
110             }
111         });
112         buttonClose.setToolTipText("Close File");
113         buttonClose.setIcon(imageClose);
114         buttonHelp.setToolTipText("About");
115         buttonHelp.setIcon(imageHelp);
116         buttonCompilar.setToolTipText("Extrair");
117         buttonCompilar.setIcon(imageCompilar);
118         buttonCompilar.setPreferredSize(new Dimension(29, 29));
119         buttonCompilar.setMaximumSize(new Dimension(29, 29));
120         buttonCompilar.setMinimumSize(new Dimension(29, 29));
121         buttonCompilar.addActionListener(new ActionListener() {
122             public void actionPerformed(ActionEvent e) {
123                 try {
124                     buttonCompilar_actionPerformed(e);
125                 } catch (Exception ex) {
126                       Logger.getLogger(Janela.class.getName()).log(Level.SEVERE,
null, ex);
127                 }
128             }
```

```
129         });
130
131         menuFile.add(menuFileExit);
132         menuBar.add(menuFile);
133         menuHelp.add(menuHelpAbout);
134         menuBar.add(menuHelp);
135         this.getContentPane().add(statusBar, BorderLayout.SOUTH);
136         toolBar.add(buttonOpen);
137         toolBar.add(buttonClose);
138         toolBar.add(buttonHelp);
139         toolBar.add(buttonCompilar);
140         this.getContentPane().add(toolBar, BorderLayout.NORTH);
141         jScrollPane1.setViewportView(jTextArea1);
142         jScrollPane2.setViewportView(jTAErro);
143         panelCenter.add(jScrollPane2, null);
144
145         comboBox.addActionListener(new ActionListener() {
146           public void actionPerformed(ActionEvent e) {
147             JComboBox cb = (JComboBox)e.getSource();
148             int i = cb.getSelectedIndex();
149
150             Busca busca = new Busca();
151             String resultado = busca.consulta(i);
152
153             jTAErro.setText(resultado);
154             jTAErro.repaint();
155           }
156         });
157
158         String[] opcoes = {"Todos os nomes de Aspectos",
159             "Todos os nomes de Pointcuts",
160             "Detalhes dos Pointcuts",
161             "Todos tipos de advices",
162             "Todos os inter-tipos",
163             "Todos fragmentos dos inter-tipos"};
164
165         jScrollPane2.setColumnHeaderView(comboBox);
166         comboBox.setModel(new DefaultComboBoxModel(opcoes));
167         comboBox.setEnabled(false);
168
169         panelCenter.add(jTextArea2, null);
170         panelCenter.add(jScrollPane1, null);
171         this.getContentPane().add(panelCenter, BorderLayout.CENTER);
172     }
173
174       /**
175        *
176        * @param e
177        */
178       void fileExit_ActionPerformed(ActionEvent e) {
179           System.exit(0);
180       }
181
182       /**
183        *
184        * @param e
185        */
186       void helpAbout_ActionPerformed(ActionEvent e) {
187             JOptionPane.showMessageDialog(this, new Principal_AboutBoxPanel1(),
"About", JOptionPane.PLAIN_MESSAGE);
188       }
189
190       private void buttonOpen_actionPerformed(ActionEvent e) {
191           try {
192                       if (FILE_CHOOSER.showOpenDialog(Janela.getInstance()) ==
JFileChooser.APPROVE_OPTION) {
193                   file = FILE_CHOOSER.getSelectedFile();
194                   arq = new BufferedReader(new FileReader (file));
```

```
195
196                 String linha = null;
197                 while ((linha = arq.readLine()) != null) {
198                     jTextArea1.append(linha+"\n");
199                 }
200                 StringBuffer sb = new StringBuffer(jTextArea1.getText());
201                 arq1 = new StringReader(jTextArea1.getText());
202             }
203         } catch (IOException e1) {
204             e1.printStackTrace();
205         }
206     }
207
208     public static Janela getInstance() {
209         if (singleton == null) {
210             singleton = new Janela();
211         }
212
213         return singleton;
214     }
215
216           private  void  buttonCompilar_actionPerformed(ActionEvent  e)  throws
Exception {
217       jTAErro.setText("");
218       jTAErro.repaint();
219       Visitar v = new Visitar();
220
221       String ERRO="";
222       ERRO = v.check(jTextArea1.getText());
223
224         jTAErro.append(ERRO);
225         comboBox.setEnabled(true);
226     }
227 }
```

APÊNDICE I – ARQUIVO

PRINCIPAL_ABOUTBOXPANEL1.JAVA

```
01 package extracao.janela;
02 import javax.swing.JPanel;
03 import java.awt.GridBagLayout;
04 import java.awt.GridBagConstraints;
05 import java.awt.Insets;
06 import javax.swing.border.Border;
07 import javax.swing.BorderFactory;
08 import javax.swing.JLabel;
09
10 public class Principal_AboutBoxPanel1 extends JPanel
11 {
12   private Border border = BorderFactory.createEtchedBorder();
13   private GridBagLayout layoutMain = new GridBagLayout();
14   private JLabel labelCompany = new JLabel();
15   private JLabel labelCopyright = new JLabel();
16   private JLabel labelAuthor = new JLabel();
17   private JLabel labelTitle = new JLabel();
18
19   public Principal_AboutBoxPanel1()
20   {
21     try
22     {
23       jbInit();
24     }
25     catch(Exception e)
26     {
27       e.printStackTrace();
28     }
29  }
30
31   private void jbInit() throws Exception
32   {
33     this.setLayout(layoutMain);
34     this.setBorder(border);
35         labelTitle.setText("Ferramenta desenvolvida como parte do Trabalho de
Graduacao");
36     labelAuthor.setText("Felipe Jose Hanauer");
37     //labelCopyright.setText("Copyright");
38     labelCompany.setText("UFSM");
39          this.add(labelTitle,  new  GridBagConstraints(0,  0,  1,  1,  0.0,  0.0,
GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(5, 15, 0, 15), 0, 0));
40          this.add(labelAuthor,  new  GridBagConstraints(0,  1,  1,  1,  0.0,  0.0,
GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0, 15, 0, 15), 0, 0));
41         this.add(labelCopyright,  new  GridBagConstraints(0,  2,  1,  1,  0.0,  0.0,
GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0, 15, 0, 15), 0, 0));
42          this.add(labelCompany,  new  GridBagConstraints(0,  3,  1,  1,  0.0,  0.0,
GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0, 15, 5, 15), 0, 0));
43   }
44 }
```

APÊNDICE J – ARQUIVO SCFILTRO.JAVA

```
01 package extracao.utils;
02
03 import java.io.File;
04 import javax.swing.filechooser.*;
05
06 public class SCFiltro extends FileFilter {
07   // Accept all directories and all gif, jpg, tiff, or png files.
08   public boolean accept(File f) {
09     if (f.isDirectory()) {
10       return true;
11     }
12
13     String extension = Utils.getExtension(f);
14     if (extension != null) {
15       if (extension.equals(Utils.aj)) {
16         return true;
17       } else {
18         return false;
19       }
20     }
21     return false;
22   }
23
24   // The description of this filter
25   public String getDescription() {
26     return "Arquivos AspectJ (*.aj)";
27   }
28 }
```

APÊNDICE L – ARQUIVO UTILS.JAVA

```java
package extracao.utils;

import java.io.File;

public class Utils {
  public final static String aj = "aj";

  /*
   * Get the extension of a file.
   */
  public static String getExtension(File f) {
    String ext = null;
    String s = f.getName();
    int i = s.lastIndexOf('.');

    if (i > 0 && i < s.length() - 1) {
      ext = s.substring(i + 1).toLowerCase();
    }
    return ext;
  }
}
```

APÊNDICE M – ARQUIVO ASPECTASTVISITOR.JAVA

```
001 package extracao.visitors;
002
003 import java.util.List;
004
005 import org.aspectj.org.eclipse.jdt.core.dom.FieldDeclaration;
006 import org.aspectj.org.eclipse.jdt.core.dom.InterTypeFieldDeclaration;
007 import org.aspectj.org.eclipse.jdt.core.dom.AdviceDeclaration;
008 import org.aspectj.org.eclipse.jdt.core.dom.AfterAdviceDeclaration;
009 import org.aspectj.org.eclipse.jdt.core.dom.AfterReturningAdviceDeclaration;
010 import org.aspectj.org.eclipse.jdt.core.dom.AfterThrowingAdviceDeclaration;
011 import org.aspectj.org.eclipse.jdt.core.dom.AjASTVisitor;
012 import org.aspectj.org.eclipse.jdt.core.dom.AjTypeDeclaration;
013 import org.aspectj.org.eclipse.jdt.core.dom.AroundAdviceDeclaration;
014 import org.aspectj.org.eclipse.jdt.core.dom.BeforeAdviceDeclaration;
015 import org.aspectj.org.eclipse.jdt.core.dom.DefaultPointcut;
016 import org.aspectj.org.eclipse.jdt.core.dom.InterTypeMethodDeclaration;
017 import org.aspectj.org.eclipse.jdt.core.dom.TypeDeclaration;
018 import org.aspectj.org.eclipse.jdt.core.dom.AspectDeclaration;
019 import org.aspectj.org.eclipse.jdt.core.dom.PointcutDeclaration;
020 import org.aspectj.org.eclipse.jdt.core.dom.PointcutDesignator;
021
022 import extracao.dados.*;
023
024 public class AspectASTVisitor extends AjASTVisitor {
025
026   public AspectJ aj = new AspectJ();
027   public AjAspecto aspecto;
028
029   public AspectASTVisitor() {}
030
031   public boolean visit(TypeDeclaration no) {
032     if (((AjTypeDeclaration) no).isAspect()) {
033       this.visit((AspectDeclaration) no);
034     }
035     return true;
036   }
037
038   public boolean visit(AspectDeclaration no) {
039     aspecto = new AjAspecto();
040     aspecto.setNome(no.getName().toString());
041     aj.as.add(aspecto);
042
043     return true;
044   }
045
046   public boolean visit(PointcutDeclaration no) {
047     AjPointcut pc = new AjPointcut();
048     pc.setNome(no.getName().toString());
049
050     PointcutDesignator pd = no.getDesignator();
051     String s="";
052     if (pd instanceof DefaultPointcut) {
053       s = s + ((DefaultPointcut) pd).getDetail();
054     }
055
056     pc.setDetalhes(s);
057     aspecto.pc.add(pc);
058
059     return true;
060   }
061
062   public boolean visit(AdviceDeclaration no) {
063     AjAdvice ad = new AjAdvice();
064     ad.setTipo("default");
065     aspecto.ad.add(ad);
```

```
066
067     return true;
068   }
069
070   public boolean visit(AroundAdviceDeclaration no) {
071     AjAdvice ad = new AjAdvice();
072     ad.setTipo("around");
073     aspecto.ad.add(ad);
074
075     return true;
076   }
077
078   public boolean visit(BeforeAdviceDeclaration no) {
079     AjAdvice ad = new AjAdvice();
080     ad.setTipo("before");
081     aspecto.ad.add(ad);
082
083     return true;
084   }
085
086   public boolean visit(AfterAdviceDeclaration no) {
087     AjAdvice ad = new AjAdvice();
088     ad.setTipo("after");
089     aspecto.ad.add(ad);
090
091     return true;
092   }
093
094   public boolean visit(AfterThrowingAdviceDeclaration no) {
095     AjAdvice ad = new AjAdvice();
096     ad.setTipo("afterThrowing");
097     aspecto.ad.add(ad);
098
099     return true;
100   }
101
102   public boolean visit(AfterReturningAdviceDeclaration no) {
103     AjAdvice ad = new AjAdvice();
104     ad.setTipo("afterReturning");
105     aspecto.ad.add(ad);
106
107     return true;
108   }
109
110   public boolean visit(InterTypeFieldDeclaration no) {
111     List<FieldDeclaration> fd = no.fragments();
112     AjInterType its = new AjInterType();
113     its.setTipo("InterTypeFieldDeclaration");
114
115     for (int i=0; i < fd.size(); i++) {
116       its.fd.add(""+fd.get(i));
117     }
118
119     aspecto.it.add(its);
120
121     return true;
122   }
123
124   public boolean visit(InterTypeMethodDeclaration no) {
125     AjInterType its = new AjInterType();
126     its.setTipo("InterTypeMethodDeclaration");
127     aspecto.it.add(its);
128
129     return true;
130   }
131 }
```

APÊNDICE N – ARQUIVO VISITAR.JAVA

```
01 package extracao.visitors;
02
03 import java.io.File;
04 import java.io.IOException;
05 import java.util.HashMap;
06
07 import javax.xml.bind.JAXBContext;
08 import javax.xml.bind.JAXBException;
09 import javax.xml.bind.Marshaller;
10 import org.aspectj.org.eclipse.jdt.core.compiler.IProblem;
11 import org.aspectj.org.eclipse.jdt.core.dom.AST;
12 import org.aspectj.org.eclipse.jdt.core.dom.ASTParser;
13 import org.aspectj.org.eclipse.jdt.core.dom.CompilationUnit;
14
15 import extracao.dados.AspectJ;
16
17 public class Visitar {
18   public String check(String source) throws JAXBException, IOException {
19     ASTParser parser = ASTParser.newParser(AST.JLS2);
20     parser.setCompilerOptions(new HashMap());
21     parser.setSource(source.toCharArray());
22     CompilationUnit cu2 = (CompilationUnit) parser.createAST(null);
23
24     IProblem[] problem = cu2.getProblems();
25     String erros = new String("Extracao realizada com sucesso.");
26
27     if (problem.length > 0) {
28       erros = "";
29       for (int i=0; i < problem.length; i++) {
30         System.out.println(problem[i].getMessage());
31         erros += problem[i].getMessage()+"\n";
32       }
33     } else {
34
35       //Map ma = cu2.properties();
36
37       AspectASTVisitor visitor = new AspectASTVisitor();
38
39       cu2.accept(visitor);
40
41       AspectJ aj = visitor.aj;
42
43       File ar = new File("Resultado.xml");
44
45       JAXBContext context = JAXBContext.newInstance(AspectJ.class);
46       Marshaller m = context.createMarshaller();
47       m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
48       m.marshal(aj, ar);
49     }
50     return erros;
51   }
52 }
```