Uma Ferramenta para Busca Não Estruturada em Código AspectJ

Alternative Title: A Tool for Searching in Unstructured Code AspctJ

Rodrigo Castro Gil
Universidade Federal de Santa Maria
Avenida Roraima, 100
Santa Maria/RS, Brasil
rcgil@inf.ufsm.br

Eduardo Kessler Piveta
Universidade Federal de Santa Maria
Avenida Roraima, 100
Santa Maria/RS, Brasil
piveta@inf.ufsm.br

Deise de Brum Saccol
Universidade Federal de Santa Maria
Avenida Roraima, 100
Santa Maria/RS, Brasil
deise@inf.ufsm.br

Cristiano de Faveri
Universidade Nova de Lisboa
Caparica, Portugal
c.faveri@campus.fct.unl.pt

RESUMO

Com o aumento do tamanho dos sistemas de informação e de sua complexidade e com as limitações que as IDEs disponíveis atualmente possuem em relação a buscas mais complexas em código fonte, ferramentas que possam auxiliar desenvolvedores de software na recuperação de informações relevantes tornam-se muito úteis. Neste contexto, este artigo apresenta uma ferramenta que viabiliza a recuperação de informações em código AspectJ de forma não estruturada. Essa ferramenta permite a busca nas estruturas utilizando apenas o valor sintático da consulta ou agregando valor semântico e com isso melhorando seus resultados.

Palavras-Chave

Recuperação de Informação, Código Fonte, AspectJ

ABSTRACT

With the increasing size of information systems and their complexity and the limitations that currently have available the IDEs for more complex searches in source code, tools that can help software developers in the retrieval of relevant information becomes very useful . In this context, this paper presents a tool that enables the retrieval, informations in AspectJ code in an unstructured way. This tool allows you to search the structures using only the syntactic value of the query or adding semantic value and thereby improving their results.

Categories and Subject Descriptors

D.3.0 [**Progamming Languages**]: General—*Standards*

General Terms

Languages

Keywords

Information Retrieval, Code Sources, AspectJ

1. INTRODUÇÃO

No âmbito de recuperação de informação (RI), a recuperação de dados significa, principalmente, determinar quais documentos de uma coleção têm as palavras-chave pesquisadas em um determinado banco de dados, o que muitas vezes pode não satisfazer, de modo completo, a consulta realizada pelo usuário. Já a recuperação de informação é a interpretação dos conteúdos de informações presentes nos documentos de uma coleção e os seleciona de acordo com um grau de relevância. Esta interpretação envolve extrair, por meio de processamento de linguagem natural, informações do documento de texto [8].

A principal diferença entre a recuperação de informação e a de dados fundamenta-se no fato de que a primeira abrange textos de linguagem natural que muitas vezes não são bem estruturados e podem apresentar ambiguidade; já a segunda, lida com dados caracterizados pela estrutura e semântica bem definidas [1].

Os sistemas informatizados crescem de tamanho e de complexidade a cada dia. A busca pela qualidade em um sistema de software envolve, não somente práticas estabelecidas em um processo de desenvolvimento, mas também a análise das estruturas de código (análise estática).

No contexto de busca em código fonte existem ferramentas que realizam a analise estática, as quais operam basicamente através da obtenção de dados de um programa, da execução de algoritmos de análise e da apresentação dos resultados. A obtenção de dados de um programa é um ponto crítico para essas ferramentas, pois de acordo com o seu propósito, diferentes visões podem ser necessárias [7].

A análise estática visa a diversos objetivos, tais como descobrir oportunidades de refatoração em código, buscar falhas de segurança, assegurar convenções de escrita, etc. Em último caso, a busca e a recuperação também se aplicam a propósitos mais simples, como encontrar determinado elemento no código fonte a fim de compreender suas estruturas. A recuperação de informações de um programa é um ponto crítico na análise estática. As principais IDEs podem ser ineficientes para buscas mais complexas. Como, por exemplo, buscar todas as classes que sejam públicas e tenham um método que retorne um inteiro.

Diante da necessidade de busca e de recuperação de informação em código fonte, este artigo apresenta uma aplicação para busca textual, utilizando técnicas de busca não estruturada, em código fonte com suporte para programas orientados a objetos e orientados a aspectos. O objetivo é possibilitar que os programadores possam refinar as consultas nos repositórios de código, dando sentido semântico às palavras reservadas da linguagem, possibilitando assim definir contextos nas consultas. Esses contextos são definidos através de regras opcionais que ajudam a refinar as consultas.

Esse artigo está estruturado da seguinte forma: a Seção 2 mostra informações sobre as tecnologias utilizadas, a Seção 3 mostra a visão geral da abordagem assim como a forma de marcação dos arquivos, a indexação dos mesmos e apresenta algumas regras definidas com o objetivo de refinar as consultas, a Seção 4 apresenta um estudo de caso e seus resultados, a Seção 5 apresenta um breve comparativo com alguns trabalhos relacionados e a Seção 6 apresenta as conclusões do trabalho.

2. TECNOLOGIAS ASSOCIADAS

Para a implementação da aplicação foram utilizadas duas tecnologias associadas: Lucene e AOPJungle, as quais são apresentadas a seguir.

2.1 Lucene

O Apache Lucene [10] é uma biblioteca de software livre para indexação e recuperação de informações, originalmente escrita em Java. As APIs do Lucene focam principalmente na indexação e na procura de texto. Elas podem ser usadas para criar recursos de procura para aplicativos, como clientes de e-mail, listas de correspondência, procuras na Web, procuras em banco de dados, etc. Web sites como Wikipedia[1], TheServerSide[2], jGuru[3] e LinkedIn[4] foram desenvolvidos utilizando como motor de busca o Lucene [11].

As APIs do Lucene focam principalmente na indexação e na procura de texto. Elas podem ser usadas para criar recursos de procura para aplicativos, como clientes de e-mail, listas de correspondência, procuras na Web, procuras em banco de dados, etc. Embora tenha sido implementado em Java, devido à sua popularidade e à sua comunidade de desenvolvedores atualmente o Lucene conta com portabilidade e integração com várias outras linguagens (C/C++,C#, Ruby, Perl, Python, PHP, entre outras).

[1]www.wikipedia.org
[2]www.theserverside.com
[3]www.jguru.com
[4]www.linkedin.com

Dentre os recursos fornecidos pelo Lucene, os seguintes se destacam:

- retorna os documentos classificados como relevantes a partir do cálculo de uma pontuação atribuída a eles.
- possibilita vários tipos de consultas, como consultar uma frase específica (*PhraseQuery*), utilizar um ou mais caracteres como coringa (*WildcardQuery*), realizar a busca em um intervalo específico (*RangeQuery*), buscar por semelhança (*FuzzyQuery*), entre outros;
- admite a análise de expressões de consultas completas digitadas pelo usuário;
- usa um mecanismo de bloqueio baseado em arquivo para impedir modificações de índices simultâneos;
- permite a procura e a indexação simultaneamente.

2.2 AspectJ

A orientação a aspectos (OA) é um paradigma que estende a orientação a objetos (e outros, como o paradigma estruturado) introduzindo novas abstrações. Estes novos elementos são destinados a suprir deficiências na capacidade de representação de algumas situações [12].

Um dos elementos centrais da OA é o conceito de interesse, que são as características relevantes de uma aplicação. A OA introduz novas abstrações de modularização e mecanismos de composição para melhorar a separação de interesses transversais[5]. Essas abstrações são: os aspectos *(aspects)*, pontos de junção *(joinpoints)*, pontos de corte *(pointcuts)*, adendos *(advices)* e declaração intertipos *inter-type declarations*. A seguir é mostrada uma breve definição dessas construções.

Aspectos

Aspecto é o termo usado para denotar a abstração da OA que dá suporte a um melhor isolamento de interesses transversais. Em outras palavras, um aspecto corresponde a um interesse transversal e constitui uma unidade modular projetada para entrecortar um conjunto de classes e objetos do sistema.

Pontos de junção

Um ponto de junção é um ponto bem específico da execução de um sistema. Alguns exemplos de pontos de junção são: chamadas a métodos, execuções de métodos, leituras de atributos, modificações de atributos, etc. Através do conceito de pontos de junção, torna-se possível especificar o relacionamento entre aspectos e classes.

Pontos de corte

São os pontos responsáveis por detectarem quais pontos de junção o aspecto deverá interceptar.

[5]São chamados interesses transversais aqueles que não podem ser modularizados em classes.

Adendos

Adendo é um construtor especial semelhante a um método de uma classe, que define o comportamento dinâmico executado quando são alcançados um ou mais conjuntos de junção definidos previamente. Em outras palavras, adendos são recursos de entrecorte transversal que afetam o comportamento dinâmico de classes e objetos.

Declarações intertipos

As declarações intertipos especificam novos atributos e/ou métodos nas classes que um aspecto entrecorta, ou modificam o relacionamento entre classes e entre classes e interfaces. Ao contrário de adendos, que operam de forma dinâmica, as declarações intertipos operam estaticamente, em tempo de compilação. Além de especificarem elementos que entrecortam classes de um sistema, aspectos podem possuir métodos e atributos internos, como classes OO.

AspectJ estende Java com suporte para dois tipos de interesses transversais, o dinâmico e o estático. O primeiro torna possível definir qual execução adicional deve se executada em um determinado ponto durante a execução do programa. O segundo torna possível definir novas operações em tipos existentes.

2.3 AOPJungle

O AOPJungle [7] é um *framework* desenvolvido para simplificar o trabalho de extração e cálculo de fragmentos de programas orientados a aspectos, organizando os elementos estruturais, seus relacionamentos e partes do código em um modelo, o qual pode ser estendido para finalidades específicas. Além disso, esse *framework* disponibiliza algumas informações relevantes em relação ao programa, tais como o número de linhas de um método, de um *advice*, de um aspecto, o número de elementos descendentes de um aspecto, a profundidade da árvore de herança, etc., sendo útil a outras finalidades tais como atividades de análise estática e de extração de métricas.

O *framework* fornece uma API para acesso à informações de um programa e permite ainda consulta através da OQL (*Object Query Language*) [4], a fim de filtrar fragmentos de interesse do código. O AOPJungle foi implementado em AspectJ [9] para coleta de informações de programas escritos em AspectJ, e utiliza APIs da plataforma Eclipse para extração das informações acerca do programa. A Figura 1 ilustra a arquitetura geral do *framework* AOPJungle.

O módulo de metamodelo representa o programa em uma estrutura orientada a objetos na qual a API busca informações solicitadas pelos clientes do *framework*. O módulo de extração é responsável por ler e por transformar os elementos do programa no metamodelo correspondente.

À medida que os elementos estruturais são carregados, suas dependências estáticas são resolvidas pelo analisador de dependência. É esse módulo, por exemplo, que resolve quais pontos do programa um *advice* afeta. O processador de incrementos é responsável por computar determinados cálculos em relação ao programa e deixá-los disponíveis para consulta, como por exemplo, o número de linhas de cada método, *advice*, classe ou aspecto. Uma característica importante do AOPJungle é seu suporte de metamodelo a multiprojetos, o qual permite consultas considerando diversos sistemas ao mesmo tempo.

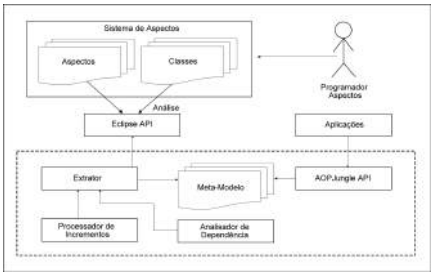

Figura1: Arquitetura do AOPJungle [7]

3. UMA FERRAMENTA PARA BUSCA NÃO ESTRUTURADA

A Figura 2 mostra a arquitetura da ferramenta proposta. A ferramenta é composta pelo *Index Engine* e pelo *Search Engine*. O *Index Engine* realiza a indexação dos arquivos, a qual se divide em extração de código (feita pelo AOPJungle), anotação de código (feita pela ferramenta de busca), e indexação do código anotado implementada utilizando o Lucene . Logo após, é mostrado onde os índices criados são mantidos. O *Search Engine* realiza pesquisa nos índices arquivados, a qual se divide em uma interface onde o usuário realiza a sua consulta, um analisador e transformador dessa consulta, um executor da consulta e um visualizador dos resultados.

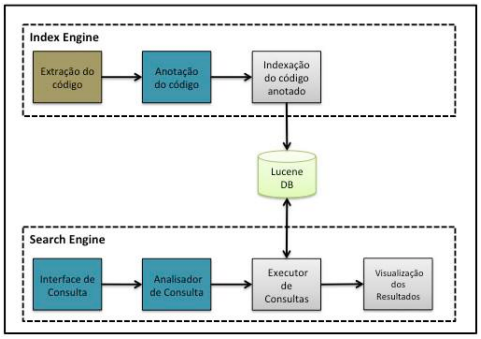

Figura2: Arquitetura da Aplicação

Buscas não estruturadas, ou seja, sem um formato, regra ou sequência padronizados, dependem dos critérios de classificação dos algoritmos usados pelos motores de busca aos quais elas são submetidas. O usuário final não pode mudar a maneira como um algoritmo classifica as palavras ou os arquivos, mas ele pode interagir com o sistema formulando ou reformulando uma consulta [6]. Essa interação é uma parte crucial para o processo de recuperação de informação, e pode determinar se o motor de busca está realizando um serviço eficaz.

Os clientes desta aplicação em geral saberão quais informações são relevantes nos resultados. Para melhorar esses resultados, é proposto dar sentido semântico aos termos da busca, utilizando palavras reservadas para isso, como por exemplo *public, String, return, int*, etc.

3.1 Indexação

O AOPJungle realiza a extração dos dados contidos nos documentos e toda vez que for encontrada uma palavra reservada será incluída a marcação definida. Por exemplo, ao encontrar o termo "public", ele será identificado como um modificador e se essa palavra for parte da definição de uma classe esta receberá a marcação "«classModifier»PUBLIC". O mesmo acontecerá com os métodos, aspectos etc.

Com as informações importantes retiradas e marcadas é criado um metamodelo, o qual é indexado. Esse metamodelo permite a construção de consultas com sentido semântico, facilitando, ao autor das consultas, fornecer informações mais precisas acerca das suas necessidades, diminuindo assim os falsos positivos, ou seja arquivos retornados após uma consulta que não satisfaçam corretamente as necessidades de informação dassas consultas.

Para uma melhor identificação dos arquivos com as informações retiradas dos arquivos originais, esse metamodelo foi definido com a extensão ".idx", apenas para o processo de análise da aplicação.

A seguir é mostrado um exemplo de uma classe Java com alguns atributos, métodos de leitura e de escrita, e logo após o modelo marcado gerado a partir dessa classe.

```
public class Aluno{
    public Integer matricula;
    public String nome;
    public String curso;

    public String getCurso() {
          return curso;
        }
    public void setCurso(String curso) {
          this.curso = curso;
        }
    public Integer getMatricula() {
          return matricula;
        }
    public void setMatricula(Integer matricula
         ) {
          this.matricula = matricula;
        }
    public String getNome() {
          return nome;
        }
    public void setNome(String nome) {
         this.nome = nome;
        }
  }
```

Arquivo Marcado:
«packageName»aluno«className»Aluno
«classModifier»PUBLIC«attributeName»matricula
«attributeName»nome«attributeName»curso
«attributeType»Integer«attributeType»String
«attributeType»String«attributeModifier»PUBLIC
«attributeModifier»PUBLIC«attributeModifier»PUBLIC
«methodName»getCurso«methodName»setCurso
«methodName»getMatricula«methodName»setMatricula
«methodName»getNome«methodName»setNome
«returnType»String«returnType»void«returnType»Integer
«returnType»void«returnType»String«returnType»void
«methodModfier»PUBLIC«methodModfier»PUBLIC
«methodModfier»PUBLIC«methodModfier»PUBLIC
«methodModfier»PUBLIC«methodModfier»PUBLIC
«parameterName»curso«parameterName»matricula
«parameterName»nome«parameterType»String
«parameterType»Integer«parameterType»String

A Figura 3 mostra o resultado da indexação de um repositório onde se encontra a classe mostrada como exemplo, sendo possível então, recuperar as informações presentes nos documentos.

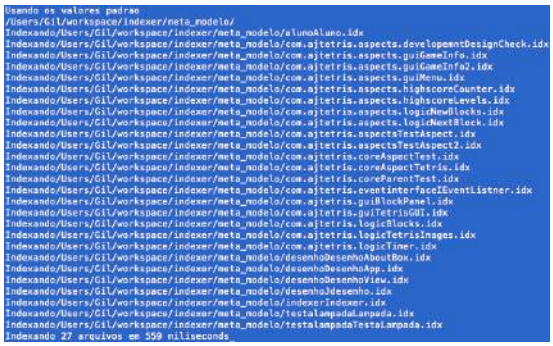

Figura3: Saída Indexação

3.2 Transformação das Consultas

Com o objetivo de melhorar a precisão das buscas realizadas nos repositórios, foram adotadas algumas regras para a realização das consultas. Para facilitar a transformação das consultas e oferecer aos usuários a possibilidade de refinar suas buscas melhorando assim os resultados, a seguir são listadas essas regras.

Regra 1: Para dar sentido semântico às consultas elas devem iniciar com o símbolo "::", o qual define que a próxima palavra definirá o contexto da consulta.

Se a consulta não for inciada pelo símbolo proposto, a busca será realizada levando em conta apenas o valor textual, não observando o valor semântico das palavras reservadas. Por exemplo, se o usuário digitar apenas "String", a consulta retornará todas a ocorrências dessa palavra, não importando se ela define um tipo de retorno ou um tipo de um atributo, etc.

Regra 2: Para realizar uma consulta que retorne o nome de um pacote, o seguinte texto é necessário: "::package nome do pacote", onde "package"define que a busca será realizada apenas nos pacotes presentes no repositório indexado. Por exemplo, para buscar um pacote com o nome "aluno", o processo seria esse:
Consulta do Usuário: **::package aluno**
Consulta traduzida: **«packageName»aluno**

Regra 3: Para buscar as informações de uma classe a consulta deve ser iniciada por "::class", seguida das informações as quais pretende-se encontrar. Por exemplo, para buscar uma classe que tenha o nome "Aluno", as consultas seriam essas:
Consulta do usuário: **::class Aluno**
Consulta traduzida: **«className»Aluno**

Uma obervação importante, se for pretendido buscar uma classe utilizando o nome da classe, o nome deve vir logo após a definição "::class", mas se a procura pretendida não precisar do nome da classe, esse não precisa constar na consulta. A seguir um exemplo de uma consulta feita com base no modificador da classe.
Consulta do usuário: **::class public**
Consulta traduzida **«classModifier»public**

Regra 4: Para recuperar as informações acerca de um interface a consulta deve ser iniciada por "::interface", seguida das informações as quais pretende-se encontrar. Por exemplo, para buscar uma interface que tenha o nome "MinhaInterface", as consultas seriam essas:
Consulta do usuário: **::interface MinhaInterface**
Consulta traduzida: **«interfaceName»MinhaInterface**

Da mesma forma que nas classes não é necessário que a consulta tenha como termo o nome da interface, mas se ele for utilizado deve vir logo após da definição "::interface".

Regra 5: Para procurar informações acerca de um método a consulta deve ser iniciada por "::method", completada das informações as quais serão buscadas. A seguir um exemplo de uma consulta que busca todos os métodos privados.
Consulta do usuário: **::method private**
Consulta transformada: **«methodModifier»private**

Regra 6: Se a busca pretendida for acerca do retorno de um método ou de um *advice around*, a mesma deve ser feita diretamente nos tipos de retornos, e deve ser iniciada por "::return". Aqui um exemplo de uma pesquisa feita para encontrar todos os métodos, ou *advices around*, que tenham retorno do tipo *String*:
Consulta do usuário: **::return String**
Consulta transformada: **«returnType»String**

Regra 7: Para consultar informações sobre parâmetros de um método a consulta deve ser inciada por "::parameter", se for pretendido buscar através do nome do parâmetro, o nome deve ser o primeiro item da consulta, assim como nas classes e métodos. A seguir o exemplo de uma consulta para buscar todos os parâmetros com o nome "args".
Consulta do usuário: **::parameter args**
Consulta transformada: **«parameterName»args**

Regra 8: Para consultar informações acerca dos atributos *(fields)* a consulta dever iniciar com "::field". A seguir um exemplo de uma consulta que retorna todos os atributos privados.
Consulta do usuário: **::field private**
Consulta transformada: **«attributeModifier»private**

Regra 9: Para consultar informações acerca de um aspecto a consulta deve ser iniciada por "::aspect"seguida das informações desejadas. A seguir um exemplo da uma consulta que busca todos os aspectos privados:
Consulta do usuário: **::aspect private**
Consulta transformada: **«aspectModifier»private**

Regra 10: Para consultar acerca de um *advice*, seja um *adviceAfter*, um *adviceAfterReturning*, um *adviceAfterThrowing*, um *adviceBefore* ou um adviceAround, é necessário buscar direto pelo *kind* que define a forma do *advice*. Por exemplo, uma consulta que retorna todos os *advices before* deve ter a seguinte estrutura:
Consulta do usuário: **::adviceKind Before**
Consulta transformada: **«adviceKind»Before**

Regra 11: Para consultar acerca de parâmetros de um advice, a consulta dever ser iniciada por "::aspectParameterType", se a consulta tomar como base o tipo do parâmetro, e se a consulta tomar como base o nome do parâmetro esta deve ser iniciada por "::aspectParameterName". A seguir um exemplo de cada consulta:
Exemplo 1:
Consulta do usuário: **::aspectParameterType Graphics**
Consulta transformada: **«aspectParameterType»Graphics**
Exemplo 2:
Consulta do usuário: **::aspectParameterName gr**
Consulta transformada: **«aspectParameterName»gr**

Regra 12: Também é possível construir consultas mais elaboradas utilizando os operadores lógicos "AND"e "OR". Para isso basta o usuário inserir o "AND"ou o "OR"entre os termos da consulta. Se o separador dos termos for apenas um espaço em branco o Lucene assume ser um "OR". Lembrando que para a troca de contexto é necessário que o usuário insira o símbolo "::", para que o analisador das consultas reconheça essa troca. Por exemplo, se a primeira parte da consulta for sobre uma classe e a segunda for sobre um método para marcar a troca de contexto é preciso inserir "::method", para que o analisador troque de contexto. A seguir é mostrado um exemplo utilizando o operador "AND".

Exemplo 1: Consulta para buscar os métodos que tenham o modificador *public* e sejam da classe "Aluno".
Consulta do usuário: **::method public AND ::class Aluno**
Consulta transformada: **«methodModifier»public AND «className»Aluno**

Exemplo 2 : Uma busca que retorne todos métodos com o modificador *public* ou todos os métodos que tenham o retorno do tipo *String*
Consulta do usuário: **::method public ::return String**
Consulta transformada: **«methodModifier»public OR «returnType»String**

Regra 13: Outro recurso do Lucene que é explorado para refinar as consultas, é a utilização dos operadores unários de adição e subtração. A seguir é mostrada a aplicação desses operadores.

Para buscar os documentos que tenham um aspecto com o nome "NextBlock"e não possam ser públicos basta entrar com a seguinte consulta: **::aspect NextBlock - public**, pois o operador de subtração exclui todos os documentos que tenham o termo que vem logo após ele.

Da mesma forma é possível usar o operador de adição para buscar todos os arquivos que tenham obrigatoriamente um aspecto público e que possam ter o nome de "NextBlock". Para tal basta entrar com a seguinte consulta: **::aspect NextBlock + public**, pois o operador de adição exige que os documentos, para serem considerados relevantes, tenham o termo que vem logo após ele. Por convenção da aplicação,

esses operadores deverão sempre ser usados antes do segundo termo da consulta.

Todos os itens de uma consulta recebem os caracteres barra e aspas no início e no fim. Isso é necessário para que o Lucene processe corretamente os caracteres *guillemets* inseridos para marcar cada item, dessa forma o Lucene reconhece esses caracteres e retorna os resultados com precisão.

As regras definidas nessa seção não são exaustivas para todas as possibilidades de consultas a serem feitas. Elas abrangem as estruturas mais utilizadas da linguagem, e foram criadas com a preocupação de não deixar as consultas com uma estrutura rígida. A Tabela 1 mostra de forma resumida as regras definidas.

Tabela1: Regras para Consultas

Regra	Função	Sintaxe
1	Definir todos os contextos	::
2	Contexto pacotes	::package
3	Contexto classes	::class
4	Contexto interfaces	::interface
5	Contexto métodos	::method
6	Contexto retornos	::return
7	Contexto parâmetros métodos	::parameter
8	Contexto atributos	::field
9	Contexto aspectos	::aspect
10	Contexto adendos	::adviceKind
11	Contexto parâmetros adendos	::asp
12	AND e OR	AND, OR ou um espaço em branco
13	Exigência Ou negação	+ para exigência, e - para exclusão, sempre antes do segundo termo da consulta

4. ESTUDO DE CASO

Para demonstração do funcionamento da aplicação foi utilizado, como repositório, um *workspace* com alguns projetos Java e alguns projetos AspectJ, o qual teve a sua indexação mostrada na Figura 3. A maior parte das consultas foram realizadas no projeto AOPTetris. Esse projeto foi escolhido por ser implementado em AspectJ e possuir aspectos e classes possibilitando assim consultas mais elaboradas.

A Figura 4 mostra o resultado da consulta "::aspect NextBlock", a qual retornou o arquivo: *NextBlock.idx*, do pacote *logic*, sendo esse o único arquivo que satisfaz essa consulta entre os arquivos indexados.

```
Encontrados  1  documento(s)  (em 91milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNextBlock.idx
```

Figura4: Consulta ::aspect NextBlock

A Figura 5 mostra o retorno da consulta "::aspect NextBlock public", a qual retorna o arquivo *NextBlock.idx* que satisfaz o primeiro elemento da consulta, por ter um aspecto com o nome *NextBloc*, e todos os outros arquivos satisfazem o segundo elemento dessa consulta, por terem aspectos públicos.

```
Encontrados  10  documento(s)  (em 30milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNextBlock.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.developemntDesignCheck.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.guiGameInfo2.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.coreAspectTest.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.guiGameInfo.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspectsTestAspect2.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.highscoreCounter.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.guiMenu.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.highscoreLevels.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNewBlocks.idx
```

Figura5: Consulta ::aspect NextBlock public

A Figura 6 mostra o resultado de uma consulta onde cada um dos termos da consulta está em um contexto. O primeiro termo busca pelo nome de um aspecto e o segundo por um ponto de corte do tipo *call*.O primeiro arquivo retornado satisfaz o primeiro termo da consulta e os outros três arquivos correspondem ao retorno do segundo termo. Essa consulta é construída da seguinte forma: "::aspect NextBlock ::pointcut call".

Como foi abordado na regra 3.2, tanto na consulta da Figura 5 quando na consulta da Figura 6, o que separa os termos da consulta é um espaço em branco, o qual é interpretado como um "OR", portanto os aquivos precisam satisfazer apenas um dos termos da consulta para serem considerados como relevantes.

```
Encontrados  4  documento(s)  (em 33milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNextBlock.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.highscoreCounter.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNewBlocks.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.highscoreLevels.idx
```

Figura6: Consulta ::aspect NextBlock ::pointcut call

Uma consulta com os mesmos termos da consulta anterior, mas que os arquivos precisam satisfazer os dois termos da consulta para serem considerados relevantes, é construída da seguinte forma, "::aspect NextBlock + ::pointcut call". E tem seu retorno mostrado na Figura 7, e apesar de ter o mesmo retorno da consulta mostrada na figura 4.

```
Encontrados  1  documento(s)  (em 118milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNextBlock.idx
```

Figura7: Consulta ::aspect NextBlock + ::pointcut call

O retorno mostrado na Figura 8 refere-se à seguinte consulta: "::adviceKind Around + ::returnType int". Essa consulta retorna todos os arquivos que tenham um *advice Around* e um retorno do tipo *int*. O resultado dessa consulta não garante é que esse retorno seja desse *advice*. O retorno poderá ser de um método do mesmo aspecto que faz parte do mesmo arquivo.

```
Encontrados  2  documento(s)  (em 39milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNewBlocks.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNextBlock.idx
```

Figura8: Consulta ::adviceKind Around + ::returnType int

A Figura 9 mostra o resultado da seguinte consulta: "(::aspect NextBlock + ::adviceKind After) ::class GameInfo".

Ela mostra que é possível utilizar os operadores "AND", e "OR", com resultados combinados possibilitando assim um maior refinamento de consultas. No exemplo citado, para um documento ser considerado relevante, ele precisa satisfazer as duas condições do primeiro termo da consulta ou satisfazer o segundo termo dela. Também é mostrado no exemplo que é possível trocar de contexto inúmeras vezes, na consulta mostrada o contexto começa em um aspecto, muda para um adendo, e depois muda para uma classe.

```
Encontrados  2  documento(s)  (em 142milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNextBlock.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.guiGameInfo.idx
```

Figura9: (Consulta ::aspect NextBlock + ::advice-Kind After) ::class GameInfo

Como a coleção de documentos indexados é conhecida, conforme mostrado na figura 3, foi possível aplicar as duas técnicas de medição de desempenho em recuperação de informação, mais utilizadas, que são:

- Precisão (P), que é a fração de documentos retornados que é relevante;
- Revocação (R), que é a fração dos documentos relevantes retornados.

Para uma melhor compreensão da aplicação dessas técnicas o conjunto de arquivos indexados, foi dividido em quatro subconjuntos como mostra a Figura 10. Com base nessa divisão é possível definir a Precisão por : $P = \frac{A}{A+B}$, e a Revocação por: $R = \frac{A}{A+D}$.

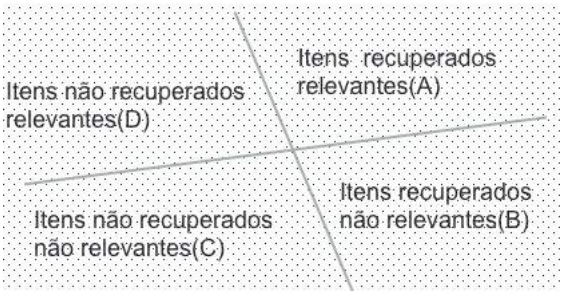

Figura10: Divisão do conjunto de arquivos

A Figura 11, mostra o resultado de uma busca por: "GameInfo". O resultado mostrado é obtido através da busca com base apenas no valor textual da palavra. Neste caso o resultado é o mesmo da consulta que agrega valor semântico, assumindo-se que o usuário procura pela classe "GameInfo", realizando a seguinte consulta: "::class GameInfo", com o mesmo resultado a Precisão e a *Recall* têm o mesmo valor nas duas consultas, como é mostrado na Tabela 2.

```
Encontrados  1  documento(s)  (em 49milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.guiGameInfo.idx
```

Figura11: Consulta GameInfo

A Figura 12 mostra o resultado da consulta pelo termo "int". Como é possível observar a consulta teve como resultado quatorze documentos, assumindo que o usuário procura pelos documentos que tenham atributos do tipo *int*, esta pode ser melhorada para: "::field int", a qual tem seu resultado mostrado na Figura 13, e apresenta doze arquivos como retorno. Num primeiro olhar, essa diferença pode não parecer significativa, mas como é mostrado na Tabela 2 a Precisão tem uma redução em quase 15%.

```
Encontrados  14  documento(s)  (em 27milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNewBlocks.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.logicBlocks.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.eventinterfaceIEventListner.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.guiBlockPanel.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.highscoreLevels.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNextBlock.idx
/Users/Gil/workspace/indexer/meta_modelo/testalampadaLampada.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.coreAspectTetris.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.guiGameInfo.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.highscoreCounter.idx
```

Figura12: Consulta int

```
Encontrados  12  documento(s)  (em 45milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.eventinterfaceIEventListner.idx
/Users/Gil/workspace/indexer/meta_modelo/testalampadaLampada.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.logicBlocks.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.guiGameInfo.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.guiBlockPanel.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.highscoreCounter.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.coreAspectTetris.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.highscoreLevels.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNextBlock.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNewBlocks.idx
```

Figura13: Consulta ::field int

A diferença de precisão é maior ainda, quando realizada uma busca apenas pelo termo "String", a qual tem seu resultado mostrado na Figura 14. Assumindo-se que o usuário busca por todos os arquivos que possuam um retorno do tipo *String*, essa consulta pode ser melhorada para: "::return String", e tem seu resultado mostrado na Figura 15. Na Tabela 2 é possível constatar que a Precisão, entre as consultas, tem uma variação de mais de 70%, sendo possível constatar que o refinamento de consulta proposto tem resultados positivos.

```
Encontrados  11  documento(s)  (em 19milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspectsTestAspect.idx
/Users/Gil/workspace/indexer/meta_modelo/alunoAluno.idx
/Users/Gil/workspace/indexer/meta_modelo/testalampadaTestaLampada.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.logicTetrisImages.idx
/Users/Gil/workspace/indexer/meta_modelo/indexerIndexer.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNewBlocks.idx
/Users/Gil/workspace/indexer/meta_modelo/desenhoDesenhoApp.idx
/Users/Gil/workspace/indexer/meta_modelo/testalampadaLampada.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.guiBlockPanel.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.coreAspectTetris.idx
```

Figura14: Consulta String

```
Encontrados  3  documento(s)  (em 34milliseconds)
/Users/Gil/workspace/indexer/meta_modelo/alunoAluno.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.aspects.logicNewBlocks.idx
/Users/Gil/workspace/indexer/meta_modelo/com.ajtetris.logicBlocks.idx
```

Figura15: Comsulta ::return String

Tabela2: Comparação da Precisão e da Revocação entre as consultas

Consulta	Revocação	Precisão
"GameInfo"	$R = \frac{1}{1+0} = 1$	$P = \frac{1}{1+0} = 1$
"::class GameInfo"	$R = \frac{1}{1+0} = 1$	$P = \frac{1}{1+0} = 1$
"int"	$R = \frac{1}{1+0} = 1$	$P = \frac{12}{12+2}$ =0,8571
"::field int"	$R = \frac{1}{1+0} = 1$	$P = \frac{12}{12+0} = 1$
"String"	$R = \frac{1}{1+0} = 1$	$P = \frac{3}{3+8}$ =0,2727
"::return String"	$R = \frac{1}{1+0} = 1$	$P = \frac{3}{3+0} = 1$

Como pode ser constatado na Tabela 2 a revocação se manteve igual, em todas as consultas comparadas, isso acontece pelo fato de que nenhuma das consultas deixou de retornar algum arquivo relevante.

5. TRABALHOS RELACIONADOS

NerdyData [3] é um motor de busca com a finalidade de pesquisar o código de *websites*. Com a capacidade de indexar HTML, JavaScript, CSS e texto simples, o Nerdydata permite uma série de consultas como: uma forma livre de busca para uma determinada frase. Também é possível realizar uma pesquisa comparativa de até cinco termos para encontrar domínios, entre outras. A ferramenta apresentada neste artigo tem como finalidade pesquisar em grandes repositórios de código fonte enquanto o NerdyData tem como foco repositórios de sites web.

Krugle é um portal de busca de código aberto que em repositórios como o Apache, JavaDocs e SourceForge entre outros [5]. Você pode pesquisar por código em C + +, Java, Perl, Python, SQL, Ruby, XML, HTML, etc. Krugle tem um recurso de busca avançada que pode ajudar a diminuir a APIs certas, bibliotecas, código de amostra ou documentação. A partir da página de resultados, você pode navegar para o projeto desenvolvido com o código. O Krugle também tem suporte para AspectJ porém ele busca pelo valor sintáxico das palavras enquanto a aplicação apresentada permite agregar valor semântico às consultas.

Ohloh Code é apresentado como um dos maiores e mais abrangentes motores de busca de código com mais de 10 bilhões de linhas de código indexados e atualizados nos diretórios de software FOSS [2]. Ele indexa todos os arquivos de texto para pesquisa e tem destaque de sintaxe pois suporta 43 linguagens de programação. A sintaxe de consulta de pesquisa possui flexibilidade para procurar diferentes classes de código. O Ohloh Code atualmente não tem suporte a expressões regulares, enquanto a proposta apresentada possui.

6. CONCLUSÃO

Esse artigo apresentou uma ferramenta que auxilia o trabalho dos programadores, possibilitando o refinamento de consultas em código fonte. Para desenvolver a aplicação com suporte à linguagem AspectJ, foi necessário um estudo para um melhor conhecimento das estruturas da linguagem. Em seguida foi feita uma investigação acerca do funcionamento do *framework* AOPjungle para entender como é feita a extração das informações presentes no código sendo possível então realizar as marcações necessárias e o refinamento das consultas.

As consultas, para que seja possível serem refinadas, apresentam uma estrutura mais definida o que foge um pouco da proposta do trabalho. Por essa razão será desenvolvido um processo para que seja possível apresentar os resultados com base na localização dos objetos das consultas deixando assim mais livre a estrutura das consultas.

A aplicação proposta foi desenvolvida com suporte para as linguagens de programação AspectJ e Java, mas pode ser estendida para outras linguagens de programação. Com a utilização dessa ferramenta os usuários podem construir consultas com complexidade maior obtendo resultados, com base nos testes realizados na Seção 4, com boa precisão. Vale lembrar que essa aplicação tem como usuário final programadores, ou outros programas, pois ela depende do *feedback* desse usuário para que se possa ter certeza se as necessidades de informação pretendidas foram atendidas.

Essa aplicação não tem como objetivo classificar os documentos retornados por ordem de relevância. Isso é feito pelo Lucene o qual usa um algoritmo de classificação próprio para esse propósito. A classificação com base nas prioridades da linguagem é um ponto importante principalmente para repositórios muito grandes, por isso os autores já estão trabalhando em pesquisas para oferecer essa classificação. Também será feita a integração com as IDEs para a visualização dos resultados.

7. REFERÊNCIAS

[1] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[2] S. Basu. Ohloh. http://www.makeuseof.com/tag/open-source-matters-6-source-code-search-engines-you-can-use-for-programming-projects/, 2013. Acessado em Outubro/2013.

[3] D. Bielik and S. Sonnes. Nerdydata. http://nerdydata.com/, 2013. Acessado em Dezembro/2013.

[4] R. G. G. Cattell and D. K. Barry. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.

[5] D. Chung and K. Wang. krugle. http://www.krugle.com/, 2013. Acessado em Novembro/2013.

[6] W. Croft, D. Metzler, and T. Strohman. *Seach Engines: Information Retrieval in Pratice*. Pearson Education, Boston, MA, USA, 1th edition, 2010.

[7] C. de Faveri. Uma linguagem específica de domínio para busca em código orientado a aspectos. Mestrado, Pós Graduação em Informática, Universidade Federal de Santa Maria, 2013.

[8] D. R. M. F. dos Santos. Recuperação de informação utilizando apache lucene e wordnet, 2012.

[9] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of aspectj. In *Proceedings of the 15th European Conference on Object-Oriented Programming*, ECOOP '01, pages 327–353, London, UK, UK, 2001. Springer-Verlag.

[10] M. McCandless, E. Hatcher, and O. Gospodnetic. *Lucene In Action*. Manning Publications Co., Stanford, CT, USA, 2th edition, 2010.

[11] A. Sonawane. Usando o apache lucene para procura de texto. http://www.ibm.com/developerworks/br/java/library/os-apache-lucenesearch/, 2009. Acessado em Maio/2013.

[12] D. V. Winck and V. Goetten. Aspectj em 20 minutos. page 01, 2006.