Otavio Augusto Fuck Pereira

Suporte à Engenharia Reversa para o ambiente SEA

Florianópolis - SC

2004

Otavio Augusto Fuck Pereira

Suporte à Engenharia Reversa para o ambiente SEA

Orientador:
Ricardo Pereira e Silva

Universidade Federal de Santa Catarina - UFSC
Departamento de Informática e Estatística - INE

Florianópolis - SC

2004

Sumário

Lista de Figuras

1 Introdução

1.1 Tema

Este trabalho tem como tema a inclusão do suporte à Engenharia Reversa no Ambiente SEA. O Ambiente SEA foi desenvolvido por Ricardo Pereira e Silva, como fruto de sua tese de Doutorado, e é uma ferramenta voltada ao projeto de artefatos[1] reutilizáveis de software, e ainda carece de um suporte à Engenharia Reversa.

1.2 Delimitação do Tema

A Engenharia Reversa pode ser abordada em diversos níveis, desde código de máquina para código *assembly*, ou deste para uma linguagem de alto nível, ou desta para o projeto. O foco é trabalhar com a recuperação do projeto a partir do código fonte, ou seja, partimos de uma linguagem em alto nível e chegamos nos diagramas que "deram origem" ao código.

O código fonte é necessariamente uma linguagem Orientada a Objetos (Java, C++ , Smalltalk, etc.), e os diagramas são utilizados no formato especificado pelo ambiente.

1.3 Objetivo Geral

Apresentar uma estrutura de suporte para construção de ferramentas de engenharia reversa para o ambiente SEA.

1.4 Objetivos Específicos

Os objetivos específicos do trabalho são a construção da ferramenta, com suporte à linguagem Java e alterações no ambiente SEA e no framework OCEAN necessárias a esse

[1] Por artefato de software entende-se frameworks, componentes, bibliotecas, etc.

suporte.

1.5 Justificativa

É consenso entre os especialistas em projeto e desenvolvimento de software que durante o desenvolvimento orientado a objetos, tanto de aplicações como de artefatos de software em geral é necessário a criação de uma documentação, que compreende desde o projeto da arquitetura do artefato (ou aplicação) até o código gerado. A prática recomendada é manter o código gerado seguindo a risca as especificações do projeto, porém, a criação de um projeto que possa ser convertido em código sem modificações é algo difícil de ser atingido. Existem procedimentos que fornecem um meio de alterar o projeto durante o desenvolvimento do software, porém geralmente as pressões para fazer cumprir os prazos tendem a colocar estes procedimentos em desuso. Como resultado, normalmente temos um projeto diferente do resultado final, e o último será utilizado efetivamente.

Quando se trata do desenvolvimento de aplicações, essa diferença entre projeto e resultado final trará conseqüências graves na fase da manutenção, pois a equipe responsável terá uma dificuldade maior em entender o código. Agora, quando o artefato desenvolvido será usado para gerar novas aplicações, como é o caso de frameworks, por exemplo, essa lacuna entre projeto e código pode ser gravíssima, uma vez que isso altera a forma como o desenvolvedor da aplicação precisa utilizar o framework, e normalmente esse uso foi escolhido no projeto da aplicação, com base no projeto do framework.

Para evitar esses problemas, ao final do desenvolvimento do software (ou artefato), é feito um processo de atualização do projeto, refletindo as alterações causadas pela geração de código. Porém essa tarefa é tanto mais complexa quanto o tamanho do código, que na maioria das vezes não é pequeno. Para auxiliar a equipe de desenvolvimento nessa parte, existem ferramentas Computer Aided Software Engineering (CASE) de Engenharia Reversa, que visam construir o projeto a partir do código. Infelizmente, algumas informações são perdidas com a geração de código, como os *casos de uso*, mas diagramas como os *diagramas de classe*, *diagramas de seqüência*[2], entre outros especificados pela Unified Modeling Language (UML) são possíveis de serem reconstruídos.

O ambiente SEA foi construído a partir do framework OCEAN e é uma ferramenta CASE especialmente voltada para a construção de frameworks e componentes (1). Este ainda não possui uma ferramenta de engenharia reversa, apesar de planejado pelo autor.

[2]Diagramas de seqüência podem ser recuperados, porém com limitações

1.6 Estrutura do trabalho

No capítulos 2 e 3 são apresentadas as bases de engenharia reversa e do framework OCEAN, que juntos com o JavaCC, apresentado no capítulo 5 formam a fundação da ferramenta de engenharia reversa. A ferramenta é efetivamente apresentada nos capítulos 4, 6 e 7.

2 Engenharia Reversa

2.1 O que é

SILVA (apud Chikofsky e Cross(2), 1990) define a engenharia reversa como sendo

> O processo de análise de um software para identificar os componentes do sistema e seus interrelacionamentos, e para criar representações do sistema em outra forma ou em um nível de abstração mais elevado.

Podemos dizer ainda que ela é o exame e entendimento de aplições existentes, podendo se dar a partir do código fonte, ou até mesmo do código objeto. Quando neste último, técnicas de *disassembly* são utilizadas para recuperar o código fonte.

Quando se desenvolve um software, por exemplo, com um modelo de desenvolvimento baseado no modelo em cascata, o software passa primeiro por um estágio de levantamento de requisitos, seguido por estágio de análise, onde se desenvolvem diagramas como os de casos de uso e interação, que serão usados para gerar um diagrama de classes e outros diagramas, conforme a metodologia em uso, e só então será gerado o código para o software, que dará origem ao código objeto. Na engenharia reversa, nos interessa percorrer o caminho contrário, já temos o código do software pronto, mas sua documentação é inexistente, incompleta ou desatualizada.

2.2 Aplicação

A engenharia reversa pode ser aplicada em diversos contextos, mas a principal contribuição que ela traz é na manutenção de sistemas legados, por isso muita pesquisa é feita levando em consideração técnicas para engenharia reversa de sistemas desenvolvidos utilizando programação estruturada. No entanto, grande parte dos sistemas que estão sendo desenvolvidos hoje seguem o paradigma da orientação a objetos, e possivelmente

virão a ser os alvos da engenharia reversa em um futuro talvez não muito distante.

Apesar dos processos de desenvolvimento de software tentarem assegurar que a documentação do sistema esteja sempre de acordo com o código, dificilmente isso é cumprido. Com esse cenário em vista, a engenharia reversa de sistemas Orientado a Objetos (OO) se apresenta como uma solução simples para auxiliar a manutenção da documentação.

2.3 Limitações

Quando se dá o processo de engenharia reversa sobre um código, alguns diagramas são simples de se reconstruir, como os digramas de corpo de método, introduzido com outro nome na UML 2.0 (3), mas já presente na especificação original do OCEAN (1), e diagramas de classes.

Diagramas que representam o comportamento dinâmico ou funcionalidades do sistema são em geral mais complicados de gerar a partir do código. Diagramas como os de seqüência podem ser gerados a partir do código, porém a complexidade reside em determinar quais seqüências de chamada devem ser representadas em cada diagrama, ou em outras palavras, qual caso de uso aquele diagrama estará refinando. Algumas técnicas estão sendo pesquisadas nesse aspecto, como por exemplo em (4), onde os autores apresentam uma metodologia baseada no estudo de fluxos (*threads*, como ele chama). Com essa abordagem, os casos de uso são escolhidos através de seqüências de processamento iniciadas por uma entrada do usuário, e terminadas por uma saída para o mesmo. Além dessa seleção, são aplicados alguns processamentos adicionais de forma a detectar reuso ou inclusão de casos de uso.

2.4 Engenharia Reversa hoje

As ferramentas de engenharia reversa encontradas hoje estão normalmente presentes em ferramentas CASE, como módulo adicional, e limitam-se a recuperar o diagrama de classes a partir do código fonte. Exemplos dessas ferramentas com suporte a engenharia reversa são o *Rational Rose*, *Poseidon UML* e *Togheter*.

3 *Framework OCEAN e o ambiente SEA*

O framework OCEAN é um framework voltado para construção de ambientes de apoio ao desenvolvimento de artefatos reutilizáveis de software. O ambiente SEA é uma aplicação construída utilizando-se o framework OCEAN.

Para tratar uma vasta gama de artefatos, o OCEAN utiliza uma abstração, chamada de "conceito", que são organizados visualmente na forma de "modelos conceituais". Os conceitos podem representar, por exemplo, classes ou atributos em um diagrama de classes. Já o diagrama de classes em si é um modelo conceitual.

Os modelos e conceitos estão organizados em especificações, que determinam o que pode ser representado. Por exemplo, o ambiente SEA tem capacidade para representar tanto um sistema baseado em componentes quanto um sistema orientado a objetos. Com isso, existem tanto conceitos úteis para o sistema orientado a objetos, quanto para o de componentes. O ambiente distingue os diversos tipos de sistemas através das especificações, de forma que o sistema orientado a objetos será representado na forma de uma especificação orientada a objetos, ou simplesmente, uma especificação OO. As diferentes especificações limitam os conceitos que podem ser utilizados, bem como coordenam regras de relacionamento entre eles.

Como uma forma de aumentar o reuso, não existe um acoplamento entre conceitos, exceto onde eles são realmente necessários. No contexto de uma especificação OO, por exemplo, um conceito classe não tem um acoplamento com o conceito método, apenas uma relação de sustentação. A figura 1 apresenta o núcleo do modelo de conceitos do framework OCEAN.

3.1 Ferramentas de apoio

Além de oferecer suporte para as diferentes especificações, o framework OCEAN provê suporte a ferramentas de apoio. As ferramentas podem servir para análise semântica de especificações, inserção de padrões (de projeto, exemplo), geração de código, etc. Elas podem ser utilizadas, também, para criação de especificações, como será o caso da ferramenta de engenharia reversa. A figura 2 representa de maneira genérica o suporte existente à inclusão de ferramentas no SEA.

No capítulo 4 é explicado em um nível de detalhe um pouco maior a criação da ferramenta de engenharia reversa, bem como alterações introduzidas no framework para comportá-la.

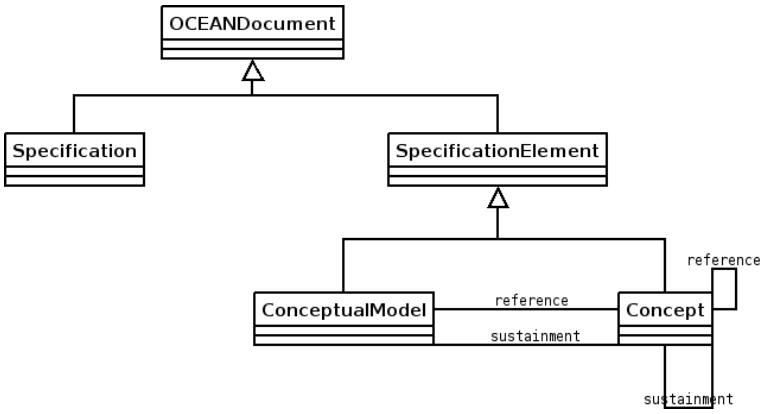

Figura 1: Suporte a ferramentas genéricas no ambiente SEA

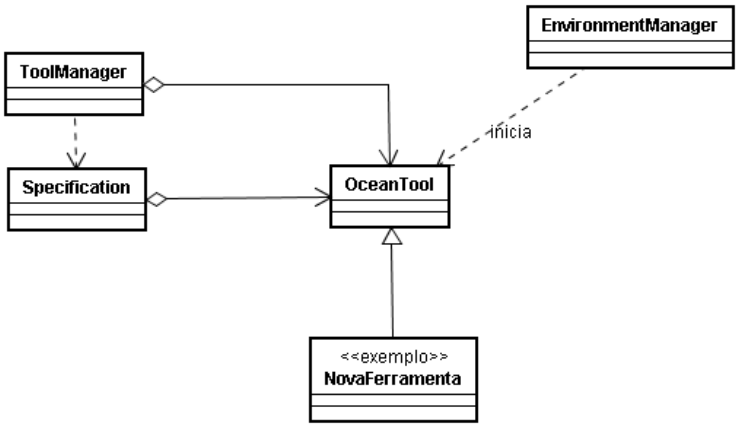

Figura 2: Suporte a ferramentas genéricas no ambiente Sea

4 *Construindo a ferramenta de Engenharia Reversa*

4.1 Visão geral da ferramenta

Para aumentar o reuso, a lógica de geração dos modelos foi separada da lógica de interpretação do código fonte. Com essa decisão, fica como responsabilidade de cada parser específico simplesmente gerar os conceitos na especificação e fornecer seus relacionamentos. Após o trabalho de parsing estar completo, a ferramenta irá analisar os dados fornecidos pelo parser e gerar os modelos, procurando mantê-los úteis e legíveis.

Com base nos requisitos e nas decisões tomadas, a ferramenta foi quebrada em alguns "módulos", cada qual com uma responsabilidade. A figura 3 apresenta o diagrama de classes da ferramenta. Como dados de entrada, a ferramenta recebe qual a linguagem de programação será utilizada, selecionando assim o parser que melhor se adapta, e uma lista de arquivos para serem processados.

A interface com o usuário será inicializada com a chamada da ferramenta pelo framework. Após configurar a ferramenta adequadamente, o usuário terá configurado o parser e o gerador dos modelos. O parser, sub classe de *OceanParser* terá como responsabilidade gerar os conceitos que são sustentados pela especificação, para que na etapa seguinte do processo, o gerador de modelos possa criar os modelos necessários.

Uma vez em execução, a ferramenta irá solicitar alguns dados para o usuário, como a linguagem de programação utilizada, quais dos modelos suportados ele deseja gerar e algumas informações de ajuste fino dos modelos gerados, como o número preferencial de classes por diagrama de classe e o número máximo de classes nesses diagramas.

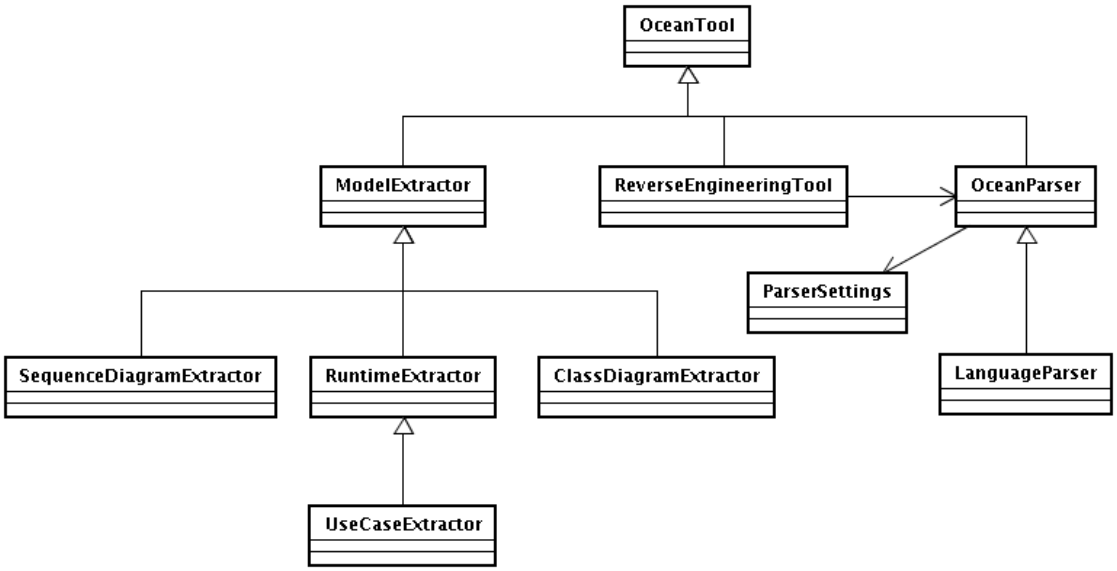

Figura 3: Diagrama de classes da ferramenta

4.2 Extração das informações do código

Para alcançar a automação do processo de engenharia reversa é necessário um suporte computacional similar ao utilizado na compilação de código. A utilização de técnicas de compilação e *parsing* permite extrair as informações existentes no código a respeito do artefato de software, tornando possível a aplicação das outras técnicas de análise para completar o processo.

A ferramenta de engenharia reversa apresentada neste trabalho se valerá de um parser construído com JavaCC. Algumas das técnicas utilizadas estão descritas no capítulo 6, enquanto que o JavaCC em si está superficialmente explicado no capítulo 5. O parser é necessariamente dependente da linguagem de programação utilizada, mas ele é construído buscando maximizar as oportunidades de reuso.

4.3 Geração de modelos

Um dos principais desafios enfrentados no desenvolvimento da ferramenta foi a extração dos modelos a partir das informações fornecidas pelo parser. No projeto de um sistema, diversos modelos são gerados representando vários aspectos do sistema, sendo posteriormente refinados, até que seja possível a geração do código. Durante esse processo, algumas informações são "perdidas", ficando disponíveis apenas em documentos textuais com sintaxe muito variada, ou às vezes, na cabeça dos desenvolvedores.

O diagrama de classes é facilmente extraído do código fonte, porém existem alguns fatores que consideramos importante levar em consideração, como o número de classes por diagrama, utilidade da informação contida, etc. Por exemplo, optando-se por gerar um único diagrama de classes, um sistema médio, com cerca de 50 classes geraria um diagrama difícil de ser compreendido. Por outro lado, ao quebrar o sistema em diversos diagramas, deve-se tomar o cuidado para que as informações sejam corretamente representadas, por exemplo, mesmo que se considere importante que duas classes ligadas por uma associação estejam representadas em 2 diagramas diferentes, pelo menos um deles deve apresentar a associação.

Outro modelo que pode ser facilmente extraído do código fonte é o diagrama de corpo de método, uma adição aos diagramas padrões da UML 1.5 feitas pelo ambiente SEA (1). O único cuidado que se deve tomar com esse modelo, será o de associá-lo ao método correto de cada classe.

Em geral, sistemas utilizam diversas bibliotecas externas. No processo de extração das classes e métodos, quando uma classe de uma dependência externa for encontrada, ela será primeiro procurada no conjunto de especificações disponíveis, para ser referenciada. Se ela não for encontrada, será marcada simplesmente como uma classe externa, sem detalhes adicionais sobre seus métodos e atributos. Como para o processo de engenharia reversa assume-se que o código de entrada é funcional, conceitos não declarados são inseridos como entidades externas para manter a especificação coerente.

5 *JavaCC - gerador de compiladores*

Para a geração dos parser, foi optado em utilizar o JavaCC (5). Existem diversas outras ferramentas de geração semi automatizada de compiladores, porém a que se mostrou mais atrativa foi o JavaCC, conforme justificativa em 5.1.

5.1 Comparativo entre as opções

Algumas opções foram analisadas, por exemplo, o Bison (6), Flex (7) e GALS (8) , além do próprio JavaCC. Bison e Flex foram descartados por não gerarem código em linguagem Java. O GALS acabou não sendo utilizado por dificultar o reuso do código. A principal diferença entre o GALS e o JavaCC está na estrutura do compilador gerado. No caso do GALS, o compilador gerado possui uma estrutura OO melhor definida, mas a forma do tratamento das ações semânticas dificulta um pouco o reuso do código. As informações de *tokens* são passadas como um parâmetro da ação, de forma que ações que necessitem de muita informação devem ser divididas em ações que coletem todas essas informações e uma outra para processá-la. No JavaCC, cada produção da gramática é representado por um método, e essas informações podem ser mais facilmente coletadas. Por exemplo, a produção de definição de uma classe em GALS seria conforme representado em 4, e na figura 5 a produção em JavaCC. Apesar das notações não estarem completas, é possível perceber que a sintaxe do JavaCC oferece ligeira vantagem com relação ao reuso. No caso do GALS, precisamos definir ações semânticas para serem tratadas pelo parser para cada possível modificador, além de uma ação adicional (#1) para sinalizar que estaremos criando uma classe e finalmente criamos uma no tratamento da ação #2. Na sintaxe do JavaCC, criamos uma construção para tratar e retornar o conjunto de modificadores (representados aqui como inteiro por opção), e com uma única ação definida podemos inserir uma classe já com seus modificadores. A definição da gramática com JavaCC é ligeiramente mais complexa do que com o GALS, mas oferece um esforço menor

```
CLASSE ::= <MODIFICADORES> "class" #1 id #2
           (<EXTENDS>)? (<IMPLEMENTS>)?
MODIFICADORES ::= <MODIFICADOR> <MODIFICADORES> | ^
MODIFICADOR ::= public #2 | protected #3 | static #4
                | private #5 | final #6
```

Figura 4: Produção para reconhecer uma classe em Java, no GALS

```
int Modifiers() {
{ int modifiers = 0; }
{
   "public" { modifiers |= ModifierSet.PUBLIC; } |
   "protected" { modifiers |= ModifierSet.PROTECTED; } |
   "static" { modifiers |= ModifierSet.STATIC; } |
   "private" { modifiers |= ModifierSet.PRIVATE; } |
   "final" { modifiers |= ModifierSet.FINAL; }
   { return modifiers; }
}
void ClassDeclaration() {
{ Token tk = null; int modifiers = 0; }
{
  modifiers = Modifiers()
  "class"
  tk = <ID> { insertClass(tk.image,modifiers); }
  [Extends()]
  [Implements()]
}
}
```

Figura 5: Produção para reconhecer uma classe em Java, no JavaCC

no reuso das ações semânticas, sendo que ele fornece a opção de tornar o parser sub classe de qualquer outra.

5.2 Especificação de gramáticas com o JavaCC

A criação de um compilador novo com o JavaCC começa pela definição da gramática, normalmente em um arquivo com extensão *jj*, apesar de não obrigatório. Uma descrição completa da gramática do JavaCC pode ser obtida em (5). De um modo geral, um arquivo do JavaCC segue uma sintaxe simples, sendo formado pela definição das opções de geração, seguido pela especificação do parser e produções da gramática. A figura 6 contém a base da gramática do JavaCC.

```
javacc_input  ::=  javacc_options
"PARSER_BEGIN" "(" <IDENTIFIER> ")"
java_compilation_unit
"PARSER_END" "(" <IDENTIFIER> ")"
( production )*
<EOF>
```

Figura 6: Definição formal de um arquivo de entrada para o JavaCC

As produções no JavaCC podem ser expressões regulares para definição de tokens, produções de controle do gerenciador de tokens, código Java ou produções em BNF. As produções do tipo *JAVACODE*, que simplesmente transcrevem um código em linguagem Java e as produções em BNF definem a gramática do compilador, enquanto que as produções na forma de expressão regulares definem os símbolos terminais da gramática e as produções de controle do gerenciador de tokens ajudam no controle do processo de leitura.

6 Parser de código Java

Para construção da ferramenta, foi adotada a linguagem Java como foco de estudo, e uma especificação da linguagem com sintaxe compatível com a versão 5.0 (9). Essa especificação foi estendida para realizar as chamadas necessárias para comunicação com o framework.

Devido à algumas limitações de representação por parte do framework, alguns aspectos da sintaxe foram ignorados. Esses aspectos, que são em geral adições da linguagem Java não diretamente representada em UML, foram ignorados para não extender demasiadamente a complexidade da ferramenta devido ao limite de tempo existente.

Outras adições realizadas à linguagem, como por exemplo a nova sintaxe para iteradores em listas podem ser facilmente adaptadas, e essa adaptação não causa prejuízo de informação, apesar não permitir que a mesma estrutura de código seja representada.

Nas seções a seguir os pontos mais interessantes da especificação final será apresentado.

6.1 Declaração inicial do parser

Toda parser no JavaCC é declarado como um objeto dentro de uma seção especial, delimitada por *PARSER_BEGIN* e *PARSER_END*. Por exemplo, um parser poderia ser declarado como:

```
PARSER_BEGIN(JavaParser)

package ocean.documents.oo.tools.reverseengineering.parsers;

import java.io.*;

public class JavaParser {
```

```
// métodos e atributos relevantes
}

PARSER_END{JavaParser}

// declaração da gramática
```

Para promover o reuso, os parsers construídos para a ferramenta de engenharia reversa no OCEAN podem estender *ocean.documents.oo.tools.reverseengineering.OceanParser*, de forma que a maioria das necessidades que o parser terá estarão sanadas. Por exemplo, para criar uma classe nova para uma especificação, o usuário do framework OCEAN deve criar o objeto, inseri-lo no repositório e marcar as relações de sustentação com a especificação que contém aquele conceito. Toda essa tarefa é encapsulada por uma única chamada de *OceanTool.pushClass(className,modifiers)*. Na criação de métodos, estes devem ser inseridos no repositório e uma classe deve ser marcada como sustentadora deles, de forma que ao criar uma classe utilizando o método *pushClass* os métodos criados por *createMethod* serão marcados como sustentados pela última classe criada e ainda não finalizada, de forma que o autor do parser pode ficar livre da tarefa de estar marcando relações de sustentação o tempo todo. Na hora de construir os corpos de métodos, a *OceanTool* pode, de acordo com um parâmetro de configuração, marcar relações de dependência entre as classes. Um parser feito para utilizar o máximo proveito desse reuso poderia ser declarado como:

```
PARSER_BEGIN(JavaParser)

package ocean.documents.oo.tools.reverseengineering.parsers;

import ocean.documents.oo.tools.reverseengineering.*;
import java.io.*;

public class JavaParser extends OceanParser {
// restante da declaração do parser e gramática
```

Um parser que não seja sub classe de OceanParser deve ter um objeto desse tipo como *wrapper* para ser aceito como um parser válido para a ferramenta.

6.2 Principais aspectos da gramática

Aqui vamos apresentar os principais pontos de tratamento da linguagem, na tentativa de esclarecer como um código na linguagem Java é convertido para uma especificação válida do ambiente SEA. Chamamos um arquivo que serve como entrada para o compilador de unidade de compilação, ou *CompilationUnit* no fonte do JavaCC. Toda unidade de compilação Java pode possuir uma declaração de pacote, podendo ser seguida por declarações de importação e em seguida declarações de tipos. Pela especificação formal da linguagem, em (10), uma unidade de compilação só pode declarar um tipo público, que deve ter o mesmo nome da unidade. Para simplificar a construção do parser, estaremos partindo do princípio que o código é aceito por um compilador oficial da linguagem, e portanto diversas verificações podem ser deixadas de lado sem prejuízo, como é o caso da nomenclatura da unidade de compilação. A figura 7 apresenta a definição da gramática para este trecho.

As unidades de compilação são normalmente organizadas em uma estrutura de contextos, portanto o trabalho pode ser bastante simplificado com a utilização de uma pilha. Uma classe declarada é inserida na pilha, e as unidades subseqüentes são então tratadas como pertencentes a essa classe, até que ela esteja completamente definida. Com esse tratamento, fica simples o reuso do código para classes internas, por exemplo, ou até mesmo classes anônimas. Na figura 8 está representada a produção da gramática que aceita a declaração de uma classe.

As produções *ExtendsList* e *ImplementsList* tem um comportamento diferenciado se a declaração for de uma interface ou de uma classe, e para evitar a definição de novas produções, foi utilizado uma das potencialidades do JavaCC que é passar parâmetros para as produções. Com essa definição podemos observar o comportamento do parser ao encontrar uma funcionalidade da linguagem que não é totalmente suportada, os parâmetros de tipo, ou *generics*. Para evitar a parada da compilação, o parser irá tratar todos os parâmetros de tipo como *java.lang.Object*, que com o suporte do *autoboxing* da linguagem não irá causar problemas no reconhecimento de tipos. No entanto, o usuário será avisado com um alerta, informando que o parser encontrou algo que ele não suporta totalmente.

No reconhecimento de métodos, algumas construções requerem tratamentos especiais, como é o caso da sintaxe do *for*. Na versão 5.0 da linguagem foi introduzida uma sintaxe estendida do laço *for*, simplificando o trabalho com iteradores. Para representar essa estrutura, optou-se por traduzi-la em um laço do tipo *while*. Essa transformação é bastante

simples, dado que a sintaxe com um iterador:

```
void iteratorSample(List<String> aList) {
    for (String s : aList) {
        // faz alguma coisa com a String "s"
    }
}
```

pode ser facilmente representada por:

```
void iteratorSample(List<String> aList) {
    Iterator i = aList.iterator();
    while (i.hasNext()) {
        String s = (String)i.next();
        // faz alguma coisa com a String "s"
    }
}
```

No diagrama de corpo de método, as estruturas de repetição suportadas são da forma *enquanto..faça*, *repita..até que* e *faça N vezes*, de forma que até mesmo o laço simples do *for* deve sofrer uma transformação para uma sintaxe do tipo *while*. A figura 9 mostra o tratamento dos laços *while*, enquanto que a figura 10 mostra o tratamento dos laços *for*, na sintaxe simples e na versão estendida para iteradores.

Como esforço para simplificar a escrita da gramática, a lógica de tratamento dos laços *for* foi encapsulada nos métodos *createIteratorFor* e *pushFor*, sendo que o primeiro é definido como método do parser Java e o último como método de *OceanParser*. Essa decisão foi tomada levando-se em consideração que os laços *for* seguindo a sintaxe simples existem em outras linguagens, enquanto que a sintaxe com iterador não é tão utilizada.

6.3 Criando os conceitos dentro do framework

A classe OceanParser contém a lógica de criação e inserção dos conceitos no framework, bem como gerenciamento da pilha dos objetos em tratamento. Quando a ferramenta é chamada para execução, ela apresenta ao usuário uma tela de configuração, e quando este manda iniciar o processo de engenharia reversa, a ferramenta irá se encarregar de gerar a especificação, e começar a alimentar o parser com as unidades de compilação

selecionadas. Algumas linguagens, como Smalltalk, contém a definição de todas as classes em um único arquivo, já linguagens como Java contém a definição de cada tipo público em um único arquivo, nomeado em função do tipo, e normalmente localizado em função do pacote. Linguagens como C++ permitem que o desenvolvedor defina a assinatura da classe em um arquivo de cabeçalho e concretize suas classes em um único ou diversos arquivos fontes. Tendo isso em vista, a lógica de controle dos arquivos foi deixada a cargo de cada parser.

O processo de geração dos modelos começa com a criação de uma nova especificação, seguida pela inserção dos conceitos reconhecidos pelo parser. Durante o processo de engenharia reversa utilizando-se o parser para a linguagem Java, quando uma classe não existente for referenciada, um novo conceito de classe será criado na especificação e marcado como uma entidade externa. Quando, e se, posteriormente a unidade de compilação que define essa classe for analisada, a classe deixará de ser externa e será definida normalmente. No caso do usuário optar por selecionar pelo menos um diretório contendo códigos fontes, o parser irá se encarregar de alterar a lista de arquivos para conter os fontes das classes não encontradas automaticamente, seguindo a convenção de organização da linguagem Java. Por essa convenção, que pode ser encontrada em (11), as classes são organizadas em diretórios de acordo com os pacotes em que elas são definidas. Por exemplo, a classe *java.lang.Object* estará localizada no diretório *lang*, dentro do diretório *java*, que por sua vez deve estar em um dos diretórios informados. Somente se esses arquivos não puderem ser encontrados o conceito será marcado como externo.

6.4 Resolvendo nomes de tipos

Como as entidades das diversas linguagens estão normalmente organizadas em espaços de nomes (*namespaces*), é possível que dois tipos com o mesmo nome coexistam sem que isso cause necessariamente um conflito, porém quando esse tipo é referenciado, é necessário distinguir um do outro. Essa distinção ocorre seguindo as regras da linguagem, que no caso da Java está descrito em (10).

As linguagens que fornecem separação por espaços de nomes oferecem também maneiras do usuário "importar" tipos definidos em outros pacotes, de maneira que é necessário que o parser ofereça um suporte a essa resolução de nomes. É possível, por exemplo, que a classe *java.util.Vector* seja utilizada referenciando-se somente *Vector*, desde que ela tenha sido importada, e conseqüentemente uma entrada de resolução de nomes foi acrescentada

no parser. Um cuidado especial deve ser tomado com importações sob demanda, ou seja, importações que deixam disponíveis todos os tipos de um determinado pacote, por exemplo, com *import java.util.*;* em Java. Nesse caso, um nome pode vir de qualquer um dos pacotes importados "sob demanda". Pela especificação da linguagem Java, se uma importação desse tipo causar um conflito na resolução do nome, um erro de compilação será gerado. Por exemplo, se forem importados os pacotes *java.util* e *other.util*, e ambos os pacotes contenham um tipo público *PublicType*, e ambos os pacotes estejam importados sob demanda em uma mesma unidade de compilação, quando o tipo *PublicType* for referenciado sem usar seu Nome Completamente Qualificado (Fully Qualified Name) (FQN), um erro de compilação será gerado. Como na construção da ferramenta partimos do princípio de que o código é aceito sem erros por um compilador, resolvemos qualquer referência de nome procurando por uma declaração no mesmo escopo (unidade de compilação, seguido por mesmo pacote), seguido por uma cláusula de importação desse tipo, e só então nas importações sob demanda, escolhendo o primeiro tipo que fechar com o nome.

Como regra na linguagem, todas as classes no pacote *java.lang* estão disponíveis automaticamente, independente de qual pacote o tipo novo está sendo declarado, assim como em C++ os tipos declarados no *namespace* padrão estão disponíveis automaticamente. Dessa forma, o parser permite uma etapa de inicialização da tabela de resolução de nomes, onde as regras pré-existentes podem ser inseridas.

```
void CompilationUnit():
{}
{
  [ PackageDeclaration() ]
  ( ImportDeclaration() )*
  ( TypeDeclaration() )*
  <EOF>
}
```

Figura 7: Definição de uma unidade de compilação

```
/*
 * Declaração de tipos.
 */
void TypeDeclaration():
{
   int modifiers;
}
{
  ";"
|
  modifiers = Modifiers()
  (
     ClassOrInterfaceDeclaration(modifiers)
   |
     EnumDeclaration(modifiers)
   |
     AnnotationTypeDeclaration(modifiers)
  )
}


void ClassOrInterfaceDeclaration(int modifiers):
{
   boolean isInterface = false;
   Token tk = null;
}
{
  ( "class" | "interface" { isInterface = true; } )
  tk = <IDENTIFIER> { if (isInterface) { pushInterface(tk.image,modifiers); } else
  [ TypeParameters() ]
  [ ExtendsList(isInterface) ]
  [ ImplementsList(isInterface) ]
  ClassOrInterfaceBody(isInterface)
}

void TypeParameters():
{}
{
   "<" TypeParameter() ( "," TypeParameter() )* ">"
   { warning("Type parameters are not fully supported. They'll be treated as Object
}
```

Figura 8: Definição de um tipo

```
void WhileStatement():
{
     String expression = null;
}
{
  "while" "("
  expression = Expression()
  ")" { pushWhile(expression); }
  Statement()
}
```

Figura 9: Estrutura de reconhecimento de um laço *while*

```
void ForStatement():
{
    String init;
    String conditional;
    String update;
    String type;
    Token tk;
    String iteratable;
}
{
  "for" "("

  (
      LOOKAHEAD(Type() <IDENTIFIER> ":")
      { disableGeneration(); }
      type = Type()
      tk = <IDENTIFIER>
      ":"
      iteratable = Expression()
      {
        enableGeneration();
        createIteratorFor(type,tk.image,iteratable);
      }
    |
     [
        { disableGeneration(); }
        init = ForInit() {enableGeneration();}
     ]
     ";"
     [
        conditional = Expression()
     ]
     ";"
     [
        update = ForUpdate()
     ]
     { pushFor(init,conditional,update); }
  )

  ")" Statement() { pop(); }
}
```

Figura 10: Estrutura de reconhecimento de um laço *for* na sintaxe simples

7 Geração dos modelos

Uma vez que o parser tenha gerado todos os conceitos encontrados na especificação, é necessário organizar esses conceitos na forma de modelos, que serão efetivamente apresentados ao usuário. Cada modelo exige seu próprio tratamento, de forma que as técnicas utilizadas para cada um deles está explicada nas seções abaixo.

7.1 Diagrama de classes

Os diagramas de classes serão gerados procurando-se não inserir muitas classes em um único diagrama, para evitar o problema de diagramas complexos e, conseqüentemente, de pouca utilidade. O número de classes por diagrama pode ser configurado pelo usuário, mas o valor sugerido é de 9 classes por diagrama.

Nessa abordagem, é necessário observar alguns fatores importantes na expressão de características do projeto, por exemplo, uma relação de herança ou agregação entre duas classes deve estar representada em pelo menos um diagrama. O principal problema enfrentado é a distribuição das classes e relacionamentos nos diagramas.

Para distribuir as classes é utilizado uma representação das classes em forma de grafo dirigido, de maneira que as classes são os vértices e as associações relevantes (herança, agregação, composição, etc) são representadas por uma aresta entre os vértices das classes relacionadas. Para montar os diagramas, é utilizado um algoritmo bastante simples, baseado no grau de entrada dos vértices.

Esse algoritmo consiste em, para todos os vértices, determinar o conjunto dos vértices que possuem uma aresta entrando naquele e mais o próprio vértice. Conjuntos com um número maior de elementos do que o recomendado são quebrados em subconjuntos com esse tamanho. Os conjuntos com apenas o próprio vértice são descartados. Depois dos conjuntos criados e subdivididos, todo conjunto com um número de elementos igual ao tamanho escolhido pelo usuário será transformado em um diagrama. Os demais conjuntos

serão agrupados de forma a conter o maior número possível de elementos iguais, procurando montar novos conjuntos com o tamanho recomendado. Finalmente, os conjuntos restantes são unidos para formar os demais diagramas. Depois que todos as classes estiverem representadas em seus modelos, as arestas que ainda não foram representadas serão adicionadas, dando preferência aos modelos com menor número de elementos. Esse algoritmo não garante que todos os diagramas tenham exatamente o número de classes informado pelo usuário, e nem que aquele será o número máximo.

Para demonstrar um pouco melhor o algoritmo, suponha o diagrama de classes da figura 11 e um número recomendado de 4 classes por modelo. O grafo gerado a partir daquele diagrama será o da figura 12. Com a análise deste, obtemos os seguintes conjuntos para cada nodo:

Conjunto	Vértice	Conjunto
C_A	A	A, B, D
C_B	B	B, C
C_C	C	C, E, G, H
C_D	D	D, E
C_E	E	E, B, A
C_F	F	F, A, C
C_G	G	G
C_H	H	H

Aqui, os conjuntos dos vértices G e H são descartados. Um modelo é gerado com o conjunto C_C, dado que ele já possui o número de vértices recomendado para os modelos. Procedendo o algortimo, os conjuntos C_A e C_B, dado que sua união resulta um novo conjunto com o número recomendado de vértices, e o mesmo ocorrendo entre C_D e C_E. Os conjuntos que ainda não foram consumidos, são então unidos e particionados. A operação de particionamento leva em consideração a quantidade de arestas representadas, de forma a manter as classes que se relacionam num mesmo diagrama. No final, o grafo e os diagramas serão comparados, e as arestas que por ventura não foram representadas serão adicionadas.

As figuras 13 a 15 são diagramas gerados extraídos pela ferramenta do código de um software de Jogo da Velha especificado pelo Prof. Ricardo e implementado por alunos, enquanto que a figura 16 mostra a lista dos diagramas de classes gerados.

7.2 Diagrama de Corpo de Método

O diagrama de corpo de método é o mais trivial para ser gerado. Cada método de cada classe dará origem a um diagrama, que é uma representação do fluxo do método. As transformações necessárias, por exemplo, para representar um estrutura do tipo *for* da linguagem Java ou C++ ficam sob responsabilidade do *parser*, e não do gerador de modelos.

A figura 17 apresenta a representação de um Diagrama de Corpo de Método (DCM) gerado pela ferramenta de engenharia reversa. Como o ambiente foi originalmente escrito em SmallTalk e a conversão para Java foi realizada em paralelo com este trabalho, no momento da geração do texto final a representação gráfica do DCM não estava funcional.

7.3 Casos de uso e diagramas de seqüência

Os diagramas de casos de uso e de seqüência são dois dos mais complicados de se obter, conforme já comentado em 2.3. Para gerar esses diagramas, optou-se por uma simplificação da técnica apresentada em (4). Essa simplificação consiste em detectar os métodos que não são explicitamente chamados por nenhum outro método da aplicação que está sofrendo a engenharia reversa, por exemplo, métodos que são chamados como conseqüência da execução do framework ou toolkit sobre o qual a aplicação foi construída, e gerar os diagramas de seqüência e seu caso de uso sustentador seguindo-se o fluxo de execução daquele método. Essa abordagem possui algumas deficiências nítidas, podendo, por exemplo, gerar um diagrama de seqüência e um caso de uso para métodos que são redefinidos em função de uma classe de framework. Por exemplo, o *Java Collections Framework* fornece alguns algoritmos eficientes de ordenação de listas, que podem ser aproveitados simplesmente definindo-se uma sub classe de *java.util.Comparator* e passando um objeto desse tipo como parâmetro do método correspondente ao algoritmo escolhido. Uma construção desse tipo em Java potencialmente gerará uma classe com um único método, que não é explicitamente chamado por nenhum outro método no código analisado. Uma outra deficiência dessa técnica é a incapacidade de detectar os atores. Uma opção para contornar essa deficência seria, possivelmente, gerar um único ator ("mundo externo") e considerá-lo relacionado com todos os casos de uso. Como esta opção não apresenta vantagens aparentes, ela não foi adotada, de sorte que os diagramas de casos de uso e de seqüência são gerados sem atores.

Apesar dessas deficiências, essa técnica foi adotada por detectar uma boa parte de possíveis casos de uso, partindo do princípio de que casos de uso são ações da interface com o usuário. Essas ações em Java, por exemplo no contexto de interfaces gráficas com o usuário podem ser criadas na forma de sub classes de *javax.swing.Action* ou na forma de tratadores de evento, por exemplo, um *ActionListener*. Nos dois casos, existe a declaração de um método que será invocado pelo framework gráfico fornecido pela linguagem. Como medida para reduzir o número de falsos casos de uso encontrados, a ferramenta só irá gerar diagramas com um certo número de objetos, informados pelo usuário, com valor padrão em 7±2. Por exemplo, um diagrama de seqüencia que não contenha pelo menos 5 elementos, entre objetos e mensagens trocadas, não será considerado um caso de uso e não será gerado.

Para nomear os casos de uso, a técnica utilizada será simplesmente informar o nome do método que está iniciando a seqüência de chamadas. Por exemplo, se o método inicial do diagrama for o método *execute*, definido na segunda classe anônima dentro do método *init* na classe *myPackage.MainWindow*, o caso de uso gerado será "myPackage.MainWindow_init_$2_execute".

A figura 19 representa um diagrama de casos de uso extraído pela ferramenta. Infelizmente, no momento de finalização deste trabalho a geração de um modelo gráfico de diagramas de seqüência através de ferramentas não era possível. A figura 18 mostra um diagrama de seqüência em sua representação textual no ambiente.

7.4 Demais diagramas

Os demais diagramas, por exemplo, estados e desdobramentos não serão considerados como tema de estudo desse trabalho, de forma que eles serão ignorados pela ferramenta de Engenharia Reversa.

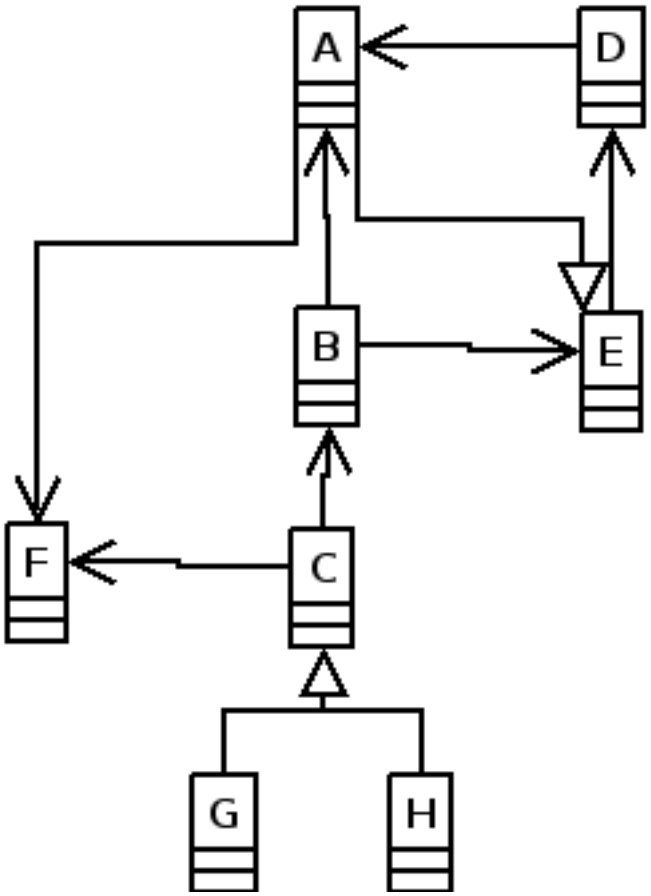

Figura 11: Diagrama de classes para demonstrar algoritmo de particionamento

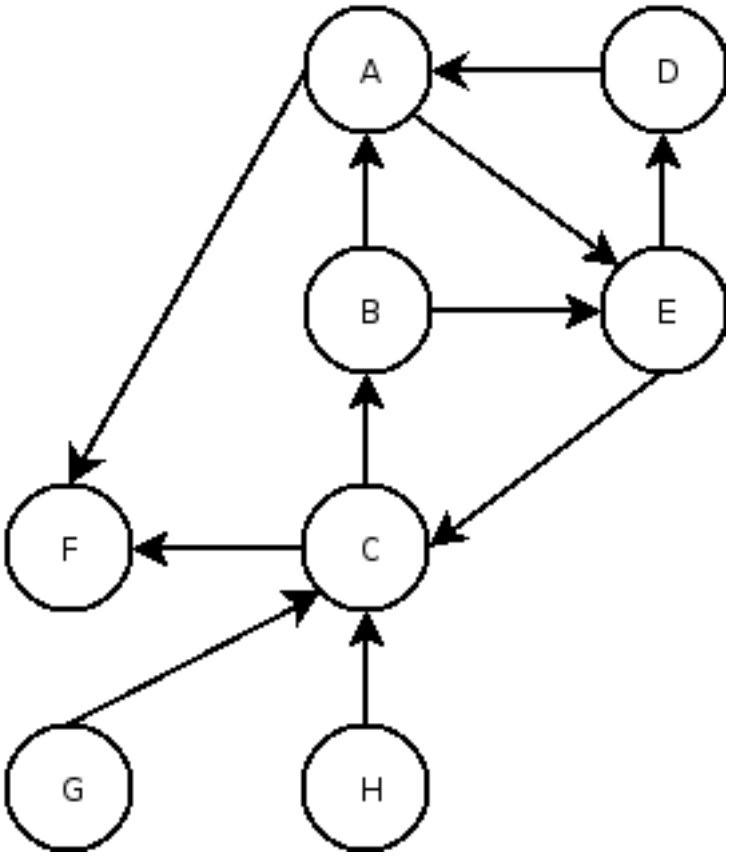

Figura 12: Grafo com as classes para demonstrar algoritmo de particionamento

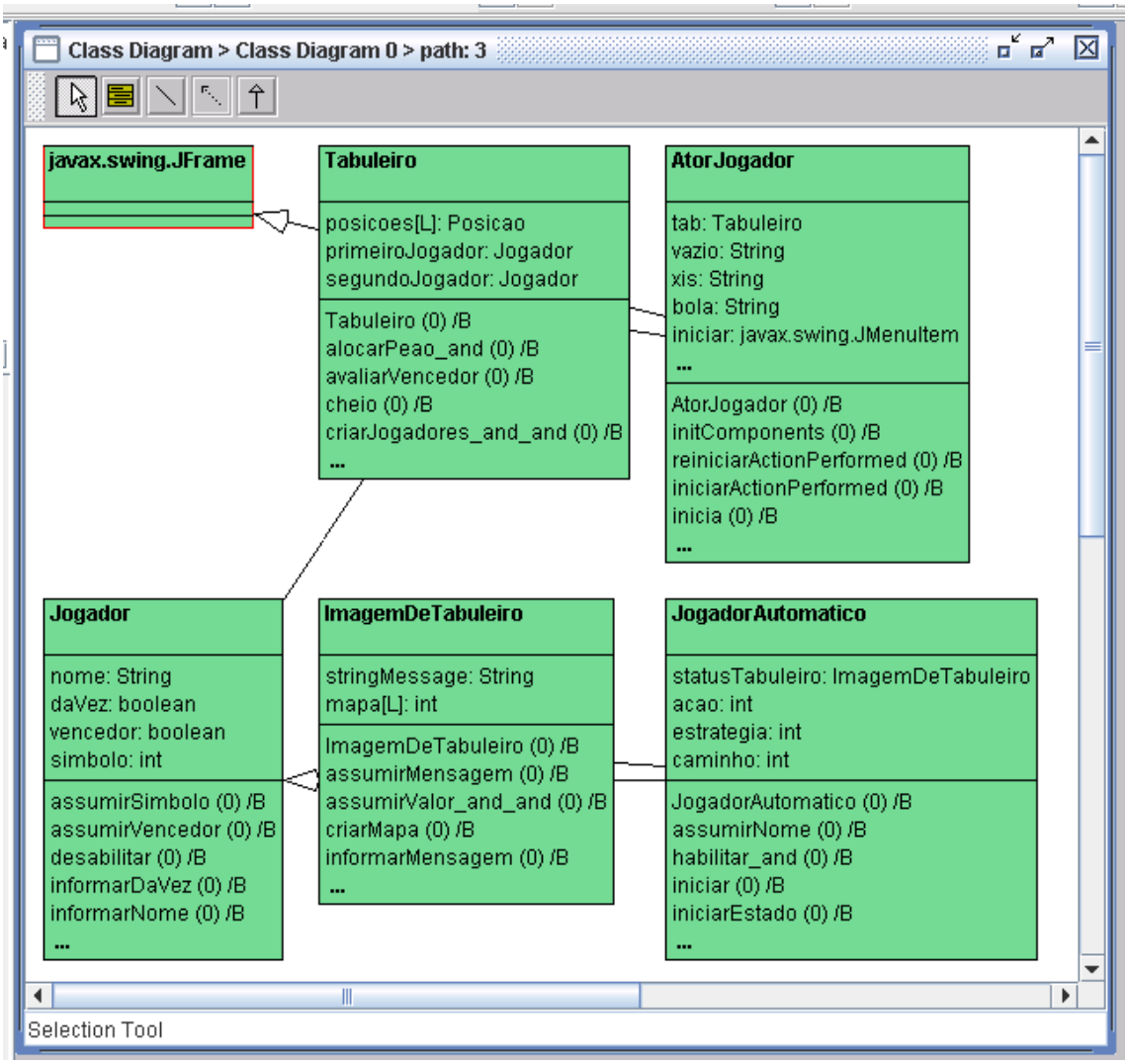

Figura 13: Diagrama de classes do Jogo da Velha - subconjunto 1

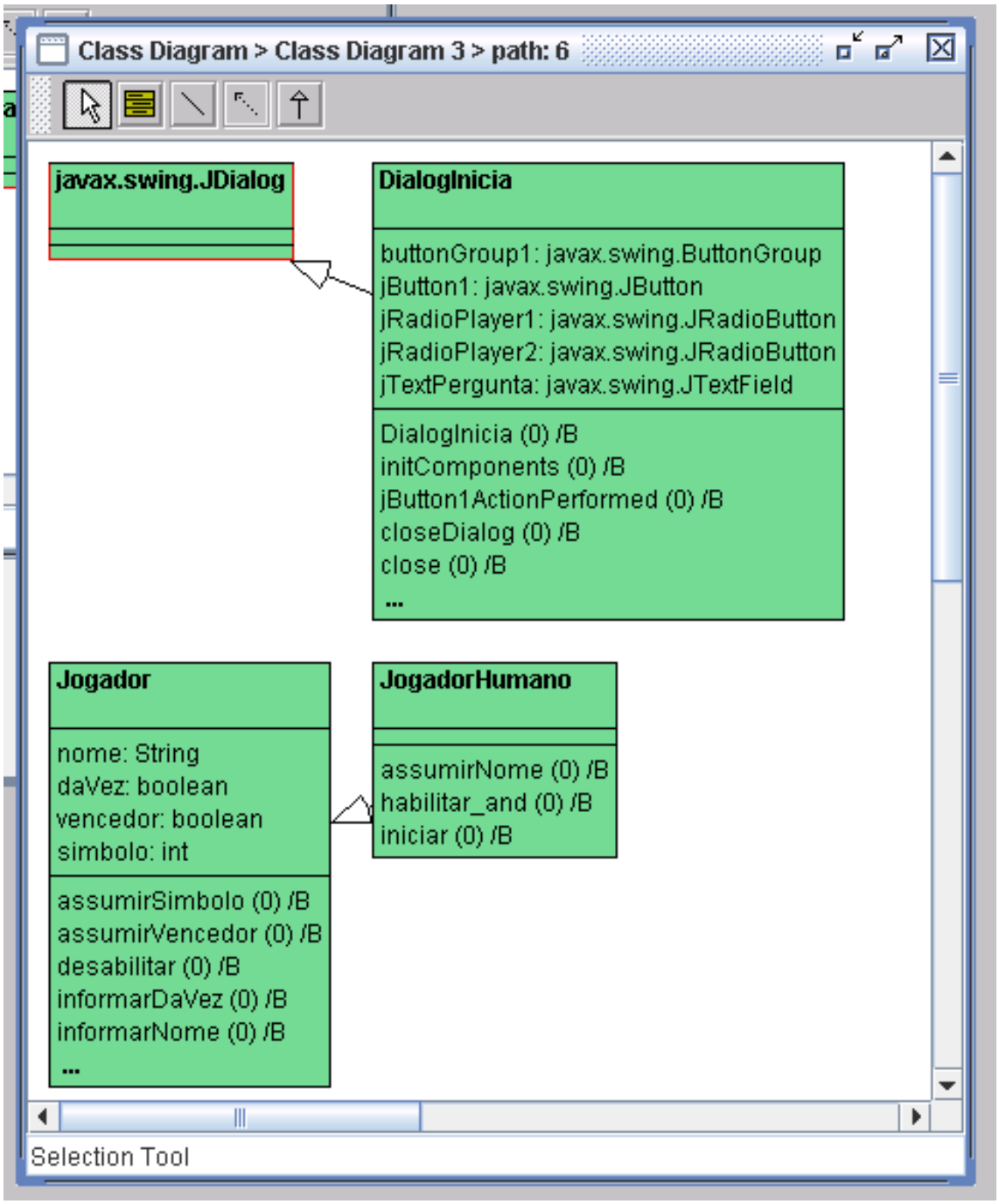

Figura 14: Diagrama de classes do Jogo da Velha - subconjunto 2

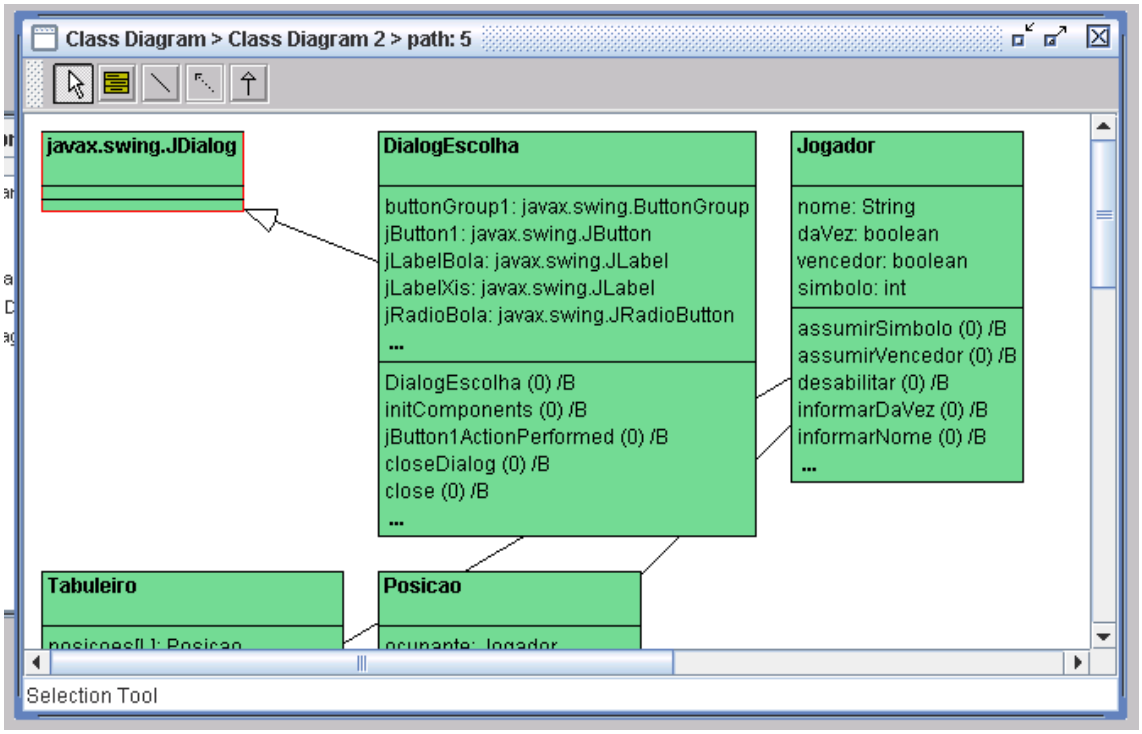

Figura 15: Diagrama de classes do Jogo da Velha - subconjunto 3

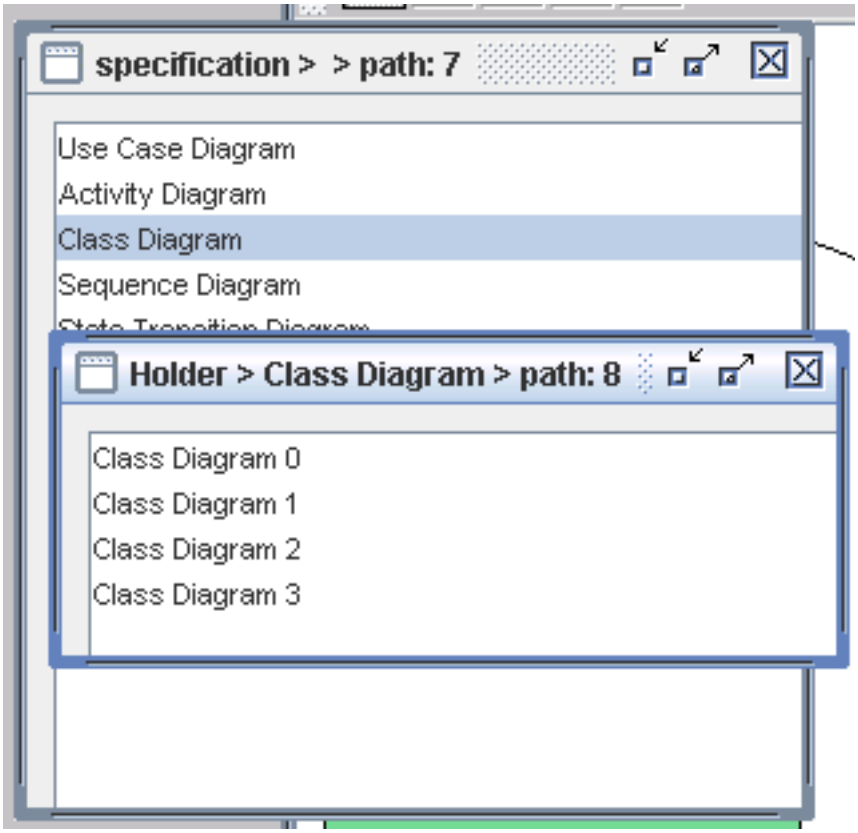

Figura 16: Diagrama de classes do Jogo da Velha - seleção dos diagramas

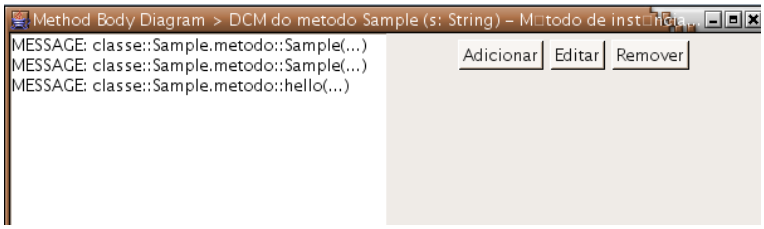

Figura 17: Visualização de um diagram de corpo de método no ambiente SEA

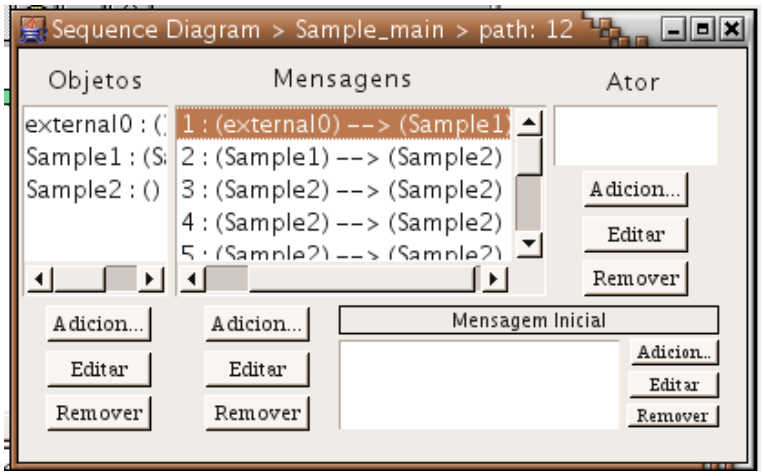

Figura 18: Diagrama de seqüência extraído pela ferramenta

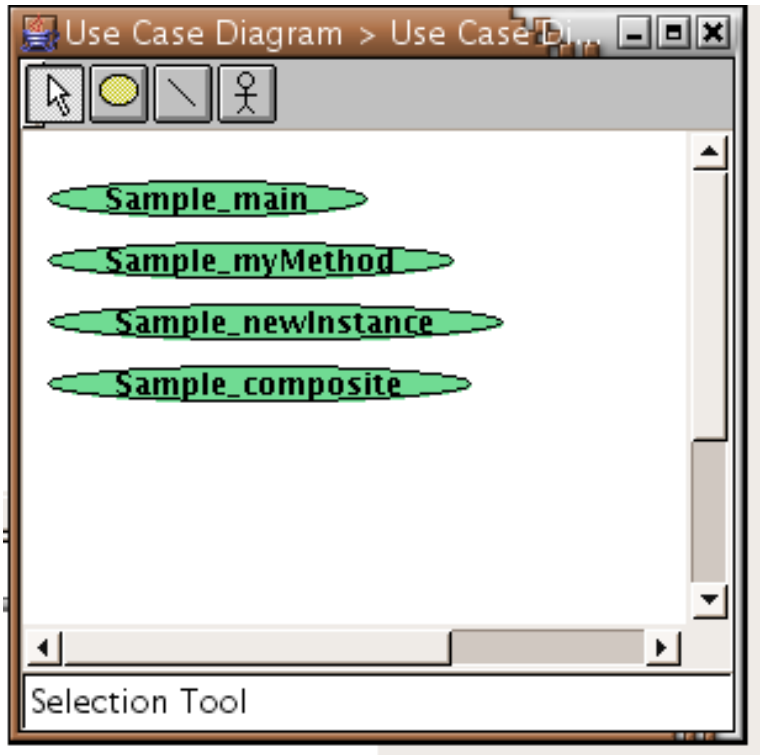

Figura 19: Diagrama de casos de uso extraído pela ferramenta

8 Conclusão e trabalhos futuros

Nos capítulos anteriores apresentamos a especificação e o projeto de uma ferramenta de engenharia reversa de código Java para o ambiente SEA. Essa ferramenta representa um primeiro passo em direção ao suporte de engenharia reversa para o ambiente, tendo utilizado vários recursos disponíveis neste, e se adaptado a algumas limitações.

A seguir são apresentadas as principais contribuições obtidas com este trabalho, suas limitações e futuras extensões, tanto em relação ao protótipo desenvolvido quanto em relação a engenharia reversa em si.

O trabalho gerou um suporte inicial de engenharia reversa no ambiente SEA, sendo capaz de recuperar com sucesso diagramas de classe e recriar os diagramas de corpo de método. Apesar de diagramas de seqüência e casos de uso serem gerados, muito precisa ser melhorado nestes, sendo necessário uma complementação no sentido do correto particionamento do sistema em casos de uso e detecção de possíveis atores.

Apesar do suporte à engenharia reversa ter sido incluída no ambiente, algumas das propostas de melhoria na representação de especificações OO no ambiente SEA foram abandonadas em função de cumprimento de prazos. No processo de recuperação do projeto propriamente dito muito ainda pode ser melhorado, iniciando com a detecção de casos de uso e os diagramas de seqüência que os refinam. Com relação a ferramenta, ainda é necessário fazer com que o projeto recuperado seja mais independente de linguagem.

Deixamos como sugestão de trabalhos futuros a extensão do suporte para outras linguagens de programação OO, recuperação de outros diagramas, como por exemplo, diagramas de estado e extensão do ambiente SEA para representar totalmente as produções das linguagens de programação sem perdas semânticas.

Cada vez mais artefatos de software estão sendo construídos e entrando em fase de manutenção, sendo cada vez mais importante a existência de documentação precisa e atualizada destes. Infelizmente, muitos grupos de desenvolvimento e até mesmo manutenção tendem a negligenciar a atualização da documentação. A engenharia reversa vem a ser

uma ferramenta importantíssima de apoio para reduzir essa deficiência de documentação, mas em geral o processo de entendimento de um sistema complexo não é uma tarefa trivial e pode levar mais tempo do que o disponível. O esforço desprendido nessa tarefa pode ser sensivelmente reduzido com a utilização de ferramentas automatizadas. Algumas das técnicas apresentadas com este trabalho podem vir a aumentar o alcance das ferramentas de engenharia reversa, diminuindo mais o esforço despendido para compreensão de sistemas existentes.

Referências

1 SILVA, R. P. *Suporte ao desenvolvimento e uso de frameworks e componentes*. Dissertação (Tese de Doutorado) — Universidade Federal do Rio Grande do Sul, Instituto de Informï¿½tica, Marï¿½o 2000. 6, 9, 15

2 CHIKOFSKY, E. J.; CROSS, J. H. Reverse engineering and design recovery: A taxonomy. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 7, n. 1, p. 13–17, 1990. ISSN 0740-7459. 8

3 OMG. UML 2.0 superstructure specification. Draft version in September 8, 2003. 9

4 LUCCA, G. A. D.; FASOLINO, A. R.; CARLINI, U. de. Recovering use case models from object-oriented code: A thread-based approach. In: *WCRE '00: Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*. Washington, DC, USA: IEEE Computer Society, 2000. p. 108. ISBN 0-7695-0881-2. 9, 31

5 JAVACC: JavaCC Home. Disponível em: <https://javacc.dev.java.net/>. 16, 17

6 BISON - GNU Project - Free Software Foundation (FSF). Disponível em: <http://www.gnu.org/software/bison/bison% -.html>. 16

7 FLEX - GNU Project - Free Software Foundation (FSF). Disponível em: <www.gnu.org/software/flex/>. 16

8 GALS - Gerador de Analisadores Léxicos e Sintáticos. Disponível em: <http://gals-.sourceforge.net/>. 16

9 VISWANADHA, S. *Java 1.5 Grammar for JavaCC*. Nov 2003. Disponível em: <https://javacc.dev.java.net/files/documents/17/3131/Java1.5.zip>. 19

10 GOSLING, J. et al. *The Java Language Specification*. 5F, No.7, Lane 50, Sec.3 Nan Kang Road Taipei, Taiwan: GOTOP Information Inc., 2000. 21, 23

11 DEITEL, H. M.; DEITEL, P. J. *Java How to Program*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130341517. 23

12 WARD, M. P.; BENNETT, K. H. A practical program transformation system for reverse engineering. In: *WCRE '93: Proceedings of the 1993 Working Conference on Reverse Engineering,* (Baltimore, Maryland; May 21-23, 1993). IEEE Computer Society Press (Order Number 3780-02), May 1993. p. 212–221. Disponível em: <citeseer.ist.psu.edu/ward93practical.html>.

13 TILLEY, S. R. et al. Programmable reverse engineering. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific Publishing Company, v. 4, n. 4, p. 501–520, 1994. Disponível em: <citeseer.ist.psu.edu/tilley94programmable-.html>.

14 SYSTÄ, T. *Static And Dynamic Reverse Engineering Techniques for Java Software Systems*. Disponível em: <citeseer.ist.psu.edu/syst00static.html>.

15 JARZABEK, S.; KEAM, T. P. Design of a generic reverse engineering assistant tool. In: *WCRE '95: Proceedings of the Second Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 1995. p. 61. ISBN 0-8186-7111-4.

16 BAXTER, I. D.; MEHLICH, M. Reverse engineering is reverse forward engineering. In: *WCRE '97: Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE '97)*. Washington, DC, USA: IEEE Computer Society, 1997. p. 104. ISBN 0-8186-8162-4.

17 JARZABEK, S.; WOON, I. Towards a precise description of reverse engineering methods and tools. In: *CSMR '97: Proceedings of the 1st Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '97)*. Washington, DC, USA: IEEE Computer Society, 1997. p. 3. ISBN 0-8186-7892-5.

18 WEST, R. *Reverse Engineering - An Overview*. Londres: CCTA, 1995. 98 p., 18 cm. (Information Systems Engineering Library). ISBN 0-11-330602-4.

ANEXO A – Código fonte

A.1 Parser

```
/*
 * @(#)ReverseEngineeringTool.java
 *
 * Historico de modificacoes:
 * Autor                    Data                    Descricao
 * ------------------------ ----------------------- ---------------------------
 */
package ocean.documents.oo.tools;

import java.io.*;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.util.*;

import com.sun.org.apache.bcel.internal.generic.GETSTATIC;

import ocean.accessories.SingleConnector;
import ocean.documents.oo.models.ModeloDeObjetos;
import ocean.documents.oo.tools.reverseengineering.*;
import ocean.documents.oo.tools.reverseengineering.extractors.*;
import ocean.framework.ConceptualModel;
import ocean.framework.Specification;
import ocean.tools.OceanTool;

/**
```

```
 * ï¿½ï¿½Descrio de uma frase da classe. ï¿½ï¿½Descrio mais detalhada da cla
 * deve fazer, coisas que assume, resultados esperados, enfim, qualquer c
 * ï¿½us-la sem conhecimento ï¿½prvio.
 *
 * @author otavio
 * @since 10/09/2005
 */
public class ReverseEngineeringTool extends OceanTool
{

    /**
     * Logger da classe.
     *
     * @see org.apache.log4j.Logger
     */
    private static final org.apache.log4j.Logger logger = org.apache.log4
        .getLogger(ReverseEngineeringTool.class);

    private HashMap<String, Class< ? extends OceanParser>> parsers;

    /**
     * Parser selecionado.
     */
    private OceanParser parser;

    private UseCaseExtractor useCaseExtractor;

    /**
     *
     */
    public ReverseEngineeringTool()
    {
        super();
        initializeParsers();
    }
```

```
    /**
     * ï¿½ï¿½Descrio de uma frase do ï¿½mtodo. ï¿½ï¿½Descrio mais detalhada d
     * ï¿½mtodo deve fazer, coisas que ele assume, resultados esperados, e
     * facilite quem for ï¿½us-lo sem conhecimento ï¿½prvio.
     */
    private void initializeParsers()
    {
        parsers = new HashMap<String, Class< ? extends OceanParser>>();
        parsers.put("Java", JavaParser.class);
    }

    /*
     * (non-Javadoc)
     *
     * @see ocean.tools.OceanTool#toolName()
     */
    @Override
    public String toolName()
    {
        return "Reverse Engineering";
    }

    /*
     * (non-Javadoc)
     *
     * @see ocean.tools.OceanTool#run()
     */
    @Override
    public void run()
    {
        // logger.debug("Mostrando janela.");
         SingleConnector.managerView().showInternalFrame(
         new ReverseEngineeringLoaderFrame(this));
//          try
```

```
//          {
//              setSourceLanguage("Java");
//              createSpecification("blah");
//          }
//          catch (ParseException e)
//          {
//              logger.debug("Ops.", e);
//          }
    }

    /**
     * ï¿½ï¿½Descrio de uma frase do ï¿½mtodo. ï¿½ï¿½Descrio mais detalhada d
     * ï¿½mtodo deve fazer, coisas que ele assume, resultados esperados, e
     * facilite quem for ï¿½us-lo sem conhecimento ï¿½prvio.
     *
     * @param text
     * @throws ParseException
     */
    public void createSpecification(String specificationName) throws Pars
    {
        logger.debug("Criando ï¿½ï¿½especificao \"" + specificationName + "
        if (specificationName == null || specificationName.equals(""))
        {
            throw new IllegalArgumentException("Specification must have a
        }
        if (parser == null)
        {
            throw new IllegalStateException("No parser available.");
        }


        Specification specification = (Specification) SingleConnector.ma
            "specification");
        parser.setSpecification(specification);
        SingleConnector.manager().path().includeMainDocument_index(specif
```

```
        SingleConnector.manager().pathIndex());
    SingleConnector.manager().pathIndex(SingleConnector.manager().pat

    // TODO obter FileManager pelas configs do usuario...

    List<File> fileList = new ArrayList<File>();

//      fileList.add(new
//      File("/home/otavio/workspace/JVelha/src/java/jvelha/Posicao.ja
    // fileList.add(new
    // File("/home/otavio/workspace/JVelha/src/java/jvelha/Tabuleiro.
//      fileList.add(new File("/home/otavio/workspace/JVelha/src/java
//      fileList.add(new File("D:\\users\\otavio\\jv3"));
//    fileList.add(new File("/home/old_hd/otavio.old/ufsc/04.2/comda
    // fileList = (List<File>) getParser().getProperty("FILE_LIST");
    fileList.add(new File("Sample.java"));

    FileManager fileManager = new FileManager(fileList, true)
    {

        protected FileFilter getFileFilter()
        {
            return new FileFilter()
            {

                public boolean accept(File pathname)
                {
                    // TODO Auto-generated method stub
                    return pathname.getName().endsWith(".java") && pa
                            && pathname.canRead();
                }

            };
        }
    };
```

```
boolean success = false;

try
{
    // extrator de casos de uso e de diagramas de sequencia tem u
    // comportamento especial...
    // o extrator de casos de uso guarda uma lista dos casos de u
    // ele detectou e em
    // quais metodos esses casos de uso iniciaram...
    useCaseExtractor = new UseCaseExtractor();
    parser.addParseTimeExtractor(useCaseExtractor);

    parser.init();
    while (fileManager.hasMoreFiles())
    {
        Reader r = fileManager.getNextReader();
        parser.ReInit(r);
        try
        {
            parser.run();
        }
        catch (RuntimeException e)
        {
            logger.error("Problem parsing code at token " + parse
            throw e;
        }
        catch (Error e)
        {
            logger.error("Error parsing code at token " + parser.
            throw e;
        }
        try
        {
            r.close();
```

```
                }
                catch (IOException e)
                {
                    logger.error("Couldn't close the file reader.", e);
                }
            }

            parser.terminate();
            logger.debug("Repositorio de conceito da spec =" + specifica
            success = true;
        }
        catch (FileNotFoundException e)
        {
            logger.debug("Ops.", e);
        }

        for (ModelExtractor me : getModelExtractors())
        {
            me.specification(specification);
            me.run();
        }

        if (success)
        {
            SingleConnector.manager().path().includeMainDocument_index(sp
                SingleConnector.manager().pathIndex());
            SingleConnector.manager().pathIndex(SingleConnector.manager()
            SingleConnector.manager().updateWindow();
        }
    }

    /**
     * ï¿½ï¿½Descrio de uma frase do ï¿½mtodo. ï¿½ï¿½Descrio mais detalhada d
     * ï¿½mtodo deve fazer, coisas que ele assume, resultados esperados, e
     * facilite quem for ï¿½us-lo sem conhecimento ï¿½prvio.
```

```
 *
 * @return
 */
private List<ModelExtractor> getModelExtractors()
{
    ArrayList<ModelExtractor> arrayList = new ArrayList<ModelExtracto

    arrayList.add(new DCMExtractor());
    arrayList.add(new ClassDiagramExtractor());
    arrayList.add(new SequenceDiagramExtractor(useCaseExtractor));


    return arrayList;
}

/**
 * ï¿½ï¿½Descrio de uma frase do ï¿½mtodo. ï¿½ï¿½Descrio mais detalhada d
 * ï¿½mtodo deve fazer, coisas que ele assume, resultados esperados, d
 * facilite quem for ï¿½us-lo sem conhecimento ï¿½prvio.
 *
 * @return
 */
public List<String> getAvailableLanguages()
{
    ArrayList<String> s = new ArrayList<String>();
    s.addAll(parsers.keySet());
    return Collections.unmodifiableList(s);
}

/**
 * ï¿½ï¿½Descrio de uma frase do ï¿½mtodo. ï¿½ï¿½Descrio mais detalhada d
 * ï¿½mtodo deve fazer, coisas que ele assume, resultados esperados, d
 * facilite quem for ï¿½us-lo sem conhecimento ï¿½prvio.
 *
 * @return
```

```
 */
public OceanParser getParser()
{
    return parser;
}

/**
 * ï¿½ï¿½Descrio de uma frase do ï¿½mtodo. ï¿½ï¿½Descrio mais detalhada d
 * ï¿½mtodo deve fazer, coisas que ele assume, resultados esperados, c
 * facilite quem for ï¿½us-lo sem conhecimento ï¿½prvio.
 *
 * @return
 */
public List<Class< ? extends ConceptualModel>> getAvailableModels()
{
    List<Class< ? extends ConceptualModel>> list = new ArrayList<Clas

    list.add(ModeloDeObjetos.class);

    return list;
}

/**
 * ï¿½ï¿½Descrio de uma frase do ï¿½mtodo. ï¿½ï¿½Descrio mais detalhada d
 * ï¿½mtodo deve fazer, coisas que ele assume, resultados esperados, c
 * facilite quem for ï¿½us-lo sem conhecimento ï¿½prvio.
 *
 * @param string
 */
public void setSourceLanguage(String string)
{
    logger.debug("Trocando para parser da linguagem: " + string);
    Class< ? extends OceanParser> clazz = parsers.get(string);
    if (clazz == null)
    {
```

```
        throw new IllegalArgumentException("Invalid language name: "
    }

    try
    {
        // reader usado para enganar o parser :)
        StringReader reader = new StringReader("");

        Constructor< ? extends OceanParser> constructor = clazz
            .getConstructor(new Class[] { Reader.class });

        this.parser = constructor.newInstance(new Object[] { reader }
    }
    catch (InstantiationException e)
    {
        logger.error("Coudln't get the parser object.", e);
    }
    catch (IllegalAccessException e)
    {
        logger.error("Coudln't get the parser object.", e);
    }
    catch (SecurityException e)
    {
        logger.error("Coudln't get the parser object.", e);
    }
    catch (NoSuchMethodException e)
    {
        logger.error("Coudln't get the parser object.", e);
    }
    catch (IllegalArgumentException e)
    {
        logger.error("Coudln't get the parser object.", e);
    }
    catch (InvocationTargetException e)
    {
```

```
                logger.error("Coudln't get the parser object.", e);
            }
        }
}

/*
 * @(#)ClassDiagramExtractor.java
 *
 * Historico de modificacoes:
 * Autor                    Data                    Descricao
 * ------------------------ ----------------------- ----------------------------
 */
package ocean.documents.oo.tools.reverseengineering.extractors;

import java.util.*;
import java.util.Map.Entry;

import ocean.documents.oo.concepts.*;
import ocean.documents.oo.models.ModeloDeObjetos;
import ocean.documents.oo.tools.reverseengineering.ModelExtractor;
import ocean.documents.oo.tools.reverseengineering.util.Digraph;
import ocean.framework.Concept;
import ocean.jhotdraw.SpecificationDrawing;
import ocean.smalltalk.OceanVector;
import CH.ifa.draw.framework.Figure;
import CH.ifa.draw.framework.FigureEnumeration;
import CH.ifa.draw.standard.AbstractFigure;

/**
 * çãDescrio de uma frase da classe. çãDescrio mais detalhada da classe, o
 * deve fazer, coisas que assume, resultados esperados, enfim, qualquer c
 * áus-la sem conhecimento éprvio.
 *
 * @author otavio
 * @since 07/10/2005
 */
```

```
public class ClassDiagramExtractor extends ModelExtractor
{

    /**
     * Logger da classe.
     *
     * @see org.apache.log4j.Logger
     */
    private static final org.apache.log4j.Logger logger = org.apache.log4
        .getLogger(ClassDiagramExtractor.class);

    private static final int DEFAULT_INSET_X = 20;

    private static final int DEFAULT_INSET_Y = 20;

    /**
     *
     */
    public ClassDiagramExtractor()
    {
        super();
        // TODO Auto-generated constructor stub
    }

    /*
     * (non-Javadoc)
     *
     * @see ocean.documents.oo.tools.reverseengineering.ModelExtractor#ru
     */
    @Override
    public void run()
    {
        logger.info("Criando diagramas de classes");
        long startTime = System.nanoTime();
```

```
        List<List<Classe>> diagrams = new GraphBasedSubsetingStrategy(spe
            .partition(specification().components(Classe.class));

        long parTime = System.nanoTime();
        logger.info("Montando os diagramas. Total = " + diagrams.size());
        int diagramIndex = 0;
        for (List<Classe> list : diagrams)
        {
            createModel(list, "Class Diagram " + diagramIndex++);
        }

        long endTime = System.nanoTime();
        final double scale = 1000 * 1000 * 1000;
        logger.info("Diagram extraction algorithm took: " + ((parTime - s
                + " seconds.");
        logger.info("Diagram generation took: " + ((endTime - parTime) /
        logger.info("Full process took: " + ((endTime - startTime) / scal
    }

    /**
     * çãDescrio de uma frase do émtodo. çãDescrio mais detalhada do émtod
     * émtodo deve fazer, coisas que ele assume, resultados esperados, en
     * facilite quem for áus-lo sem conhecimento éprvio.
     *
     * @param vector
     */
    private void createModel(List<Classe> vector, String name)
    {
        ModeloDeObjetos model = (ModeloDeObjetos) specification().createC
            ModeloDeObjetos.class, name);

        int columns = (int) Math.ceil(Math.sqrt(vector.size()));
        logger.debug("Diagrams will have " + columns + " columns.");
        if (columns * columns < vector.size())
        {
```

```
        columns++;
    }
    int[] columnStarts = new int[columns];
    int[] rowStarts = new int[columns];

    int figureRow = 0;
    int figureColumn = 0;

    for (Classe classe : vector)
    {
        logger.debug("Adicionando classe " + classe.name() + " no dia

        model.absorbComponent(classe);
        // specification().attachMaster_toComponent(model, classe);
        model.drawing().createFigureFor(classe);

        rowStarts[figureRow] = Math.max(((AbstractFigure) model.drawi
            classe)).size().height, rowStarts[figureRow]);
        columnStarts[figureColumn] = Math.max(((AbstractFigure) model
            .getFigureOfConcept(classe)).size().width, columnStarts[f
        logger.debug("rowStarts[" + figureRow + "] = " + rowStarts[fi
                + " and columnStarts[" + figureColumn + "] = " + colu

        figureColumn++;
        if (figureColumn == columns)
        {
            figureColumn = 0;
            figureRow++;
        }
    }

    // obtem o valor real das colunas
    for (int i = 1; i < rowStarts.length; i++)
    {
        rowStarts[i] += rowStarts[i - 1] + DEFAULT_INSET_X;
```

```
}
for (int i = 1; i < columnStarts.length; i++)
{
    columnStarts[i] += columnStarts[i - 1] + DEFAULT_INSET_Y;
}

figureRow = 0;
figureColumn = 0;
FigureEnumeration figureEnumeration = model.drawing().figures();
while (figureEnumeration.hasMoreElements())
{
    Figure figure = figureEnumeration.nextFigure();

    int columnOffset = 0;
    int rowOffset = 0;

    if (figureColumn != 0)
    {
        columnOffset = columnStarts[figureColumn - 1] + DEFAULT_
    }
    if (figureRow != 0)
    {
        rowOffset = rowStarts[figureRow - 1] + DEFAULT_INSET_Y;
    }

    figure.moveBy(columnOffset, rowOffset);

    figureColumn++;
    if (figureColumn == columns)
    {
        figureColumn = 0;
        figureRow++;
    }
}
```

```
        model.redraw();

        addInheritances(vector, model);
        addBinaryRelations(vector, model);
        addAgregations(vector, model);

        model.redraw();
    }

    private void addAgregations(List<Classe> vector, ModeloDeObjetos mod
    {
        logger.debug("Adicionando agregacoes.");
        SpecificationDrawing drawing = model.drawing();
        for (Classe classe : vector)
        {
            List<Agregacao> components = specification().getComponentsOf
                Agregacao.class, classe);

            for (Agregacao associacaoBinaria : components)
            {
                if (vector.contains(associacaoBinaria.classeAgregado())
                        && vector.contains(associacaoBinaria.classeParte(
                {
                    if (!model.includes(associacaoBinaria))
                    {
                        logger.debug("Adicionando agregacao " + associaca
                                + " " + associacaoBinaria.classeAgregado(
                                + associacaoBinaria.classeParte().name())
                        model.absorbComponent(associacaoBinaria);
                    }
                    // FIXME --> tah lancando uma excecao:
                    drawing.createFigureFor(associacaoBinaria);
                }
            }
        }
```

```
}

/**
 * çãDescrio de uma frase do émtodo. çãDescrio mais detalhada do émtod
 * émtodo deve fazer, coisas que ele assume, resultados esperados, en
 * facilite quem for áus-lo sem conhecimento éprvio.
 *
 * @param vector
 * @param model
 */
private void addBinaryRelations(List<Classe> vector, ModeloDeObjetos
{
    logger.debug("Adicionando associacoes.");
    SpecificationDrawing drawing = model.drawing();
    for (Classe classe : vector)
    {
        List<AssociacaoBinaria> components = specification().getComp
            AssociacaoBinaria.class, classe);

        for (AssociacaoBinaria associacaoBinaria : components)
        {
            if (vector.contains(associacaoBinaria.classeTerminal1())
                    && vector.contains(associacaoBinaria.classeTermin
            {
                if (!model.includes(associacaoBinaria))
                {
                    logger.debug("Adicionando associacao " + associac
                            + " " + associacaoBinaria.classeTerminal1
                            + associacaoBinaria.cardinalidade1() + "]
                            + associacaoBinaria.classeTerminal2().nam
                            + associacaoBinaria.cardinalidade2() + "]
                    model.absorbComponent(associacaoBinaria);
                }
                // FIXME --> tah lancando uma excecao:
                drawing.createFigureFor(associacaoBinaria);
```

```
            }
        }
    }
}

/**
 * çãDescrio de uma frase do émtodo. çãDescrio mais detalhada do émtod
 * émtodo deve fazer, coisas que ele assume, resultados esperados, en
 * facilite quem for áus-lo sem conhecimento éprvio.
 *
 * @param vector
 * @param m
 */
private void addInheritances(List<Classe> vector, ModeloDeObjetos mo
{
    logger.debug("Adicionando␣herancas.");
    SpecificationDrawing drawing = model.drawing();
    for (Classe classe : vector)
    {
        List<Heranca> components = specification().getComponentsOf␣re
            classe);

        for (Heranca inheritance : components)
        {
            if (vector.contains(inheritance.superclasse())
                    && vector.contains(inheritance.subclasse()))
            {
                if (!model.includes(inheritance))
                {
                    logger.debug("Adicionando␣heranca:␣sub[" + inheri
                            + "]->" + inheritance.superclasse().name(
                    model.absorbComponent(inheritance);
                    // FIXME --> tah lancando uma excecao:
                    drawing.createFigureFor(inheritance);
                }
```

```
                }
            }
        }

    }
}

/*
 * @(#)DCMExtractor.java
 *
 * Historico de modificacoes:
 * Autor                    Data                    Descricao
 * ------------------------ ----------------------- ----------------------------
 */
package ocean.documents.oo.tools.reverseengineering.extractors;

import java.util.ArrayList;
import java.util.HashMap;

import ocean.concepts.LinkConcept;
import ocean.documents.oo.concepts.*;
import ocean.documents.oo.models.DCM;
import ocean.documents.oo.tools.reverseengineering.ModelExtractor;
import ocean.framework.Concept;
import ocean.smalltalk.OceanVector;

/**
 * ï¿½ï¿½Descrio de uma frase da classe. ï¿½ï¿½Descrio mais detalhada da cla
 * deixando claro o que ela deve fazer, coisas que assume, resultados esp
 * enfim, qualquer coisa que facilite quem for ï¿½us-la sem conhecimento ï
 *
 * @author otavio
 * @since 12/10/2005
 */
public class DCMExtractor extends ModelExtractor
{
```

```
/**
 * Logger da classe.
 *
 * @see org.apache.log4j.Logger
 */
private static final org.apache.log4j.Logger logger = org.apache.log4
    .getLogger(DCMExtractor.class);

private HashMap<Class< ? extends Concept>, Class< ? extends DCMStater

private ArrayList<Class> ignoredConcepts;

/**
 *
 */
public DCMExtractor()
{
    super();
    initializeConceptMap();
}

/**
 * Ã§Ã£Descrio de uma frase do Ã©mtodo. Ã§Ã£Descrio mais detalhada
 * deixando claro o que o Ã©mtodo deve fazer, coisas que ele assume,
 * resultados esperados, enfim, qualquer coisa que facilite quem for
 * sem conhecimento Ã©prvio.
 */
protected void initializeConceptMap()
{
    conceptClasses.put(MethodTemporaryVariable.class, MultiplePointer

    ignoredConcepts = new ArrayList<Class>();
    ignoredConcepts.add(MethodParameter.class);
    ignoredConcepts.add(DCM.class);
```

```
        ignoredConcepts.add(LinkConcept.class);
    }

    /*
     * (non-Javadoc)
     *
     * @see ocean.documents.oo.tools.reverseengineering.ModelExtractor#ru
     */
    @Override
    public void run()
    {
        logger.info("Creating DCMs");
        ArrayList<Metodo> vector = new ArrayList<Metodo>();
        OceanVector<Classe> classes = specification().components(Classe.c
        for (Classe c : classes)
        {
            vector.addAll(specification().getComponentsOf_relatedWith(Me
        }

        for (Metodo m : vector)
        {
            generateDiagram(m);
        }

    }

    /**
     * Ã§Ã£Descrio de uma frase do Ã©mtodo. Ã§Ã£Descrio mais detalhada
     * deixando claro o que o Ã©mtodo deve fazer, coisas que ele assume,
     * resultados esperados, enfim, qualquer coisa que facilite quem for
     * sem conhecimento Ã©prvio.
     *
     * @param m
     */
    private void generateDiagram(Metodo m)
```

```
{
    logger.debug("Generating DCM for method: " + m);
    DCM currentDCM = (DCM) specification().createConceptualModel_with

    OceanVector<Concept> vector = specification().repositorioConceito
        m);

    ArrayList<MethodTemporaryVariable> variables = new ArrayList<Met

    for (Object concept : vector)
    {
        if (!ignoredConcepts.contains(concept.getClass()))
        {
            logger.debug("Including figure for " + concept + " in DCM
                    + currentDCM.metodo());
            DCMStatement stmt = null;

            if (concept instanceof MethodTemporaryVariable)
            {
                // special treatment to variables - they are all gath
                // a single variables node
                variables.add((MethodTemporaryVariable) concept);
            }
            else
            {
                if (conceptClasses.get(concept.getClass()) == null)
                {
                    stmt = (DCMStatement) concept;
                    currentDCM.absorbComponent(stmt);
                }
                else
                {
                    logger.error("UNEXPECTED ENTRANCE ON BRANCH.",new
                    stmt = (DCMStatement) currentDCM.createComponent
                        .get(concept.getClass()), currentDCM.metodo()
```

```
                        stmt.name(((Concept) concept).name());
                        stmt.header("VARIABLES");
                        stmt.container(null);
                        stmt.auxOrder(0);
                        stmt.order(-1);
                    }

                    currentDCM.updateStatementOrderWithInclusionOf(stmt);
                }
            }
        }

        if (variables.size() > 0)
        {
            MultiplePointerStatement variablesConcept = (MultiplePointerS
                .createComponent_with(MultiplePointerStatement.class, cur
            variablesConcept.referedObjectList(new OceanVector(variables)
            variablesConcept.header("VARIABLES");
            variablesConcept.name("VARIABLES");
            variablesConcept.container(null);
            variablesConcept.auxOrder(0);
            variablesConcept.order(-1);

            for (Concept v : variables)
            {
                specification().attachMaster_toComponent(variablesConcept
            }
        }

        currentDCM.redraw();
    }

}

/*
```

```
 * @(#)DiagramPartitionStrategy.java
 *
 * Historico de modificacoes:
 * Autor                Data              Descricao
 * ------------------   ---------------   ------------------------------------------
 * otavio               Oct 23, 2005 çã       descrio
 */
package ocean.documents.oo.tools.reverseengineering.extractors;

import java.util.List;

import ocean.documents.oo.concepts.Classe;
import ocean.framework.Specification;

public abstract class DiagramPartitionStrategy
{

    private Specification specification;

    private int diagramSize;

    public DiagramPartitionStrategy(Specification specification, int diag
    {
        setSpecification(specification);
        setDiagramSize(diagramSize);
    }

    public void setDiagramSize(int diagramSize)
    {
        this.diagramSize = diagramSize;
    }

    public int getDiagramSize()
    {
        return diagramSize;
```

```
    }

    public void setSpecification(Specification specification)
    {
        this.specification = specification;
    }

    public Specification specification()
    {
        return specification;
    }

    public abstract List<List<Classe>> partition(List<Classe> classes);
}

/*
 * @(#)GraphBasedSubsetingStrategy.java
 *
 * Historico de modificacoes:
 * Autor                Data              Descricao
 * ------------------   ---------------   ------------------------------------------
 * otavio               Oct 23, 2005 çã       descrio
 */
package ocean.documents.oo.tools.reverseengineering.extractors;

import java.util.*;

import ocean.documents.oo.concepts.*;
import ocean.documents.oo.tools.reverseengineering.util.Digraph;
import ocean.framework.Concept;
import ocean.framework.Specification;
import ocean.smalltalk.OceanVector;

public class GraphBasedSubsetingStrategy extends DiagramPartitionStrategy
{
```

```
/**
 * Log4J Logger.
 *
 * @see org.apache.log4j.Logger
 */
private static final org.apache.log4j.Logger logger = org.apache.log4
    .getLogger(GraphBasedSubsetingStrategy.class);

public GraphBasedSubsetingStrategy(Specification specification, int d
{
    super(specification,diagramSize);
}

protected Digraph<Classe> createGraph(List<Classe> classes)
{
    Digraph<Classe> graph = new Digraph<Classe>();

    for (Classe c : classes)
    {
        graph.addNode(c);
    }

    addConnectionOfType(graph, Heranca.class);
    addConnectionOfType(graph, AssociacaoBinaria.class);
    addConnectionOfType(graph, Agregacao.class);

    return graph;
}

protected List<List<Classe>> extractSets(Digraph<Classe> graph, List<
{
    List<List<Classe>> sets = new LinkedList<List<Classe>>();

    for (Classe c : classes)
    {
```

```
            List<Classe> adjacentsOf = graph.getAdjacentsOf(c);
            if (!adjacentsOf.contains(c))
            {
                adjacentsOf.add(c);
            }
            boolean shouldAdd = true;
            for (int i = 0; i < sets.size(); ++i)
            {
                List<Classe> list = sets.get(i);
                if (list.containsAll(adjacentsOf))
                {
                    shouldAdd = false;
                    break;
                }
                else if (adjacentsOf.containsAll(list))
                {
                    sets.set(i, adjacentsOf);
                    shouldAdd = false;
                }
            }
            if (shouldAdd)
            {
                sets.add(adjacentsOf);
            }
        }
        return sets;
    }

    @Override
    public List<List<Classe>> partition(List<Classe> classes)
    {
        Digraph<Classe> graph = createGraph(classes);

        List<List<Classe>> sets = extractSets(graph, classes);
```

```
List<List<Classe>> diagrams = new LinkedList<List<Classe>>();
List<Classe> representedClasses = new LinkedList<Classe>();

logger.debug("Selecionando conjuntos com tamanho adequado: " + ge
for (Iterator<List<Classe>> it = sets.iterator(); it.hasNext(); /
{
    List<Classe> list = it.next();

    StringBuffer set = new StringBuffer();
    if (logger.isDebugEnabled())
    {
        for (Classe c : list)
        {
            set.append(", " + c.name());
        }
        set.delete(0, 2);
        logger.debug("Conjunto [" + set.toString() + "]");
    }

    if (list.size() == getDiagramSize())
    {
        logger.debug("Marcando conjunto [" + set + "] como um dia
        diagrams.add(list);
        representedClasses.addAll(list);
        it.remove();
    }
    else if (representedClasses.containsAll(list))
    {
        it.remove();
    }
}

logger.debug("Removendo conjuntos jah completamente representados
for (Iterator<List<Classe>> it = sets.iterator(); it.hasNext(); /
{
```

```
        List<Classe> list = it.next();

        if (representedClasses.containsAll(list))
        {
            it.remove();
        }
    }

    if (logger.isDebugEnabled())
    {
        for (Iterator<List<Classe>> it = sets.iterator(); it.hasNext(
        {
            List<Classe> list = it.next();
            StringBuffer set = new StringBuffer();
            for (Classe c : list)
            {
                set.append(", " + c.name());
            }
            set.delete(0, 2);
            logger.debug("Conjunto [" + set.toString() + "] esta na l
        }
    }

    int diagramSize = getDiagramSize();
    while (sets.size() > 0)
    {
        logger.debug("Procurando conjuntos com tamanho " + diagramSiz
        boolean changed = true;
        if (sets.size() == 1)
        {
            diagrams.add(sets.get(0));
            if (logger.isDebugEnabled())
            {
                StringBuffer resultString = new StringBuffer();
                for (Classe c : sets.get(0))
```

```
            {
                resultString.append(", " + c.name());
            }
            resultString.delete(0, 2);
            logger.debug("Adicionando conjunto (ultimo da lista):
            changed = true;
        }
        sets.clear();
        break;
    }

    while (sets.size() > 1 && changed)
    {
        changed = false;
        joins: for (int i = 0; i < sets.size(); ++i)
        {
            List<Classe> list1 = sets.get(i);
            for (int j = i; j < sets.size(); ++j)
            {
                List<Classe> result = new ArrayList<Classe>();

                result.addAll(list1);

                for (Classe c : sets.get(j))
                {
                    if (!result.contains(c))
                    {
                        result.add(c);
                    }
                }

                if (result.size() == diagramSize)
                {
                    if (logger.isDebugEnabled())
                    {
```

```
                        StringBuffer resultString = new StringBu
                        for (Classe c : result)
                        {
                            resultString.append(", " + c.name());
                        }
                        resultString.delete(0, 2);
                        logger.debug("Adicionando join (i,j)=(" +
                                + resultString + "]");
                    }
                    changed = true;
                    if (i != j)
                    {
                        sets.remove(j);
                    }
                    diagrams.add(result);
                    sets.remove(i);
                    i--;
                    j--;
                    continue joins;
                }
            }
        }
    }
    diagramSize--;
}

if (logger.isDebugEnabled())
{
    for (Iterator<List<Classe>> it = sets.iterator(); it.hasNext(
    {
        List<Classe> list = it.next();
        StringBuffer set = new StringBuffer();
        for (Classe c : list)
        {
            set.append(", " + c.name());
```

```
                }
                set.delete(0, 2);
                logger.debug("Conjunto [" + set.toString()
                        + "] ficou sem representacao de diagrama");
            }
        }
        return diagrams;
    }

    /**
     * çãDescrio de uma frase do émtodo. çãDescrio mais detalhada do émtod
     * deixando claro o que o émtodo deve fazer, coisas que ele assume,
     * resultados esperados, enfim, qualquer coisa que facilite quem for
     * sem conhecimento éprvio.
     *
     * @param graph
     * @param name
     */
    protected void addConnectionOfType(Digraph<Classe> graph, Class type)
    {
        OceanVector<Concept> vector = specification().components(type);

        for (Concept t : vector)
        {
            Classe start = null;
            Classe end = null;
            if (t instanceof Heranca)
            {
                Heranca her = (Heranca) t;
                start = her.subclasse();
                end = her.superclasse();
            }
            else if (t instanceof Agregacao)
            {
                Agregacao agr = (Agregacao) t;
```

```
                start = agr.classeAgregado();
                end = agr.classeParte();
            }
            else if (t instanceof AssociacaoBinaria)
            {
                AssociacaoBinaria ab = (AssociacaoBinaria) t;
                start = ab.classeTerminal1();
                end = ab.classeTerminal2();
            }

            graph.connect(start, end);
        }
    }

}

/*
 * @(#)SequenceDiagramExtractor.java
 *
 * Historico de modificacoes:
 * Autor                    Data                    Descricao
 * ------------------------ ----------------------- ----------------------------
 */
package ocean.documents.oo.tools.reverseengineering.extractors;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import CH.ifa.draw.framework.Figure;
import CH.ifa.draw.standard.CompositeFigure;

import ocean.documents.oo.concepts.*;
import ocean.documents.oo.graphical.scenarioDiagram.*;
import ocean.documents.oo.models.Cenario;
import ocean.documents.oo.tools.reverseengineering.ModelExtractor;
```

```
import ocean.framework.Concept;
import ocean.framework.SpecificationElement;
import ocean.jhotdraw.SpecificationDrawing;
import ocean.smalltalk.OceanVector;

/**
 * ï¿½ï¿½Descrio de uma frase da classe. ï¿½ï¿½Descrio mais detalhada da cla
 * deixando claro o que ela deve fazer, coisas que assume, resultados esp
 * enfim, qualquer coisa que facilite quem for ï¿½us-la sem conhecimento ï
 *
 * @author otavio
 * @since 12/10/2005
 */
public class SequenceDiagramExtractor extends ModelExtractor
{

    /**
     * Logger da classe.
     *
     * @see org.apache.log4j.Logger
     */
    private static final org.apache.log4j.Logger logger = org.apache.log4
        .getLogger(SequenceDiagramExtractor.class);

    private static final int OBJ_WIDTH = 100;

    private UseCaseExtractor useCaseExtractor;

    private ScenarioEditor editor;

    private int objects, msgOrder;

    private Cenario dseq;

    private HashMap<Concept, Objeto> objetos;
```

```
private ArrayList<Concept> visitedMessages;

/**
 * @param useCaseExtractor
 */
public SequenceDiagramExtractor(UseCaseExtractor useCaseExtractor)
{
    super();
    this.useCaseExtractor = useCaseExtractor;
    objetos = new HashMap<Concept, Objeto>();
    visitedMessages = new ArrayList<Concept>();
}

/*
 * (non-Javadoc)
 *
 * @see ocean.documents.oo.tools.reverseengineering.ModelExtractor#ru
 */
@Override
public void run()
{
    // makes the useCaseExtractor calculate the usecases
    useCaseExtractor.run();

    // gets each use case
    List<UseCase> useCases = specification().components(UseCase.class

    for (UseCase uc : useCases)
    {
        createDiagram(uc, useCaseExtractor.getSequenceStarter(uc));
    }
}

/**
```

```
 * Ã§Ã£Descrio de uma frase do Ã©mtodo. Ã§Ã£Descrio mais detalhada
 * deixando claro o que o Ã©mtodo deve fazer, coisas que ele assume,
 * resultados esperados, enfim, qualquer coisa que facilite quem for
 * sem conhecimento Ã©prvio.
 *
 * @param uc
 * @param sequenceStarter
 */
private void createDiagram(UseCase uc, Metodo sequenceStarter)
{
    msgOrder = 1;
    logger.debug("Creating SD for: " + uc);

    dseq = (Cenario) specification().createConceptualModel_withName(C

    // creating first object and the initial message
    Objeto external = createObject(dseq, objects++, "external");
    Objeto starter = createObject(dseq, objects++, sequenceStarter.cl
    starter.classe(sequenceStarter.classe());

    expandMethod(external, starter, sequenceStarter);

    dseq.redraw();
}

/**
 * Ã§Ã£Descrio de uma frase do Ã©mtodo. Ã§Ã£Descrio mais detalhada
 * deixando claro o que o Ã©mtodo deve fazer, coisas que ele assume,
 * resultados esperados, enfim, qualquer coisa que facilite quem for
 * sem conhecimento Ã©prvio.
 *
 * @param caller
 * @param calledMethod
 */
private void expandMethod(Objeto caller, Objeto target, Metodo calledM
```

```
{
    logger.info("Expanding message: " + calledMethod);
    // insert message from "caller -> target"
    createMessage(caller, target, calledMethod);

    // propagate the call
    List vector = specification().repositorioConceito().tabelaMaster(

    for (Object member : vector)
    {
        logger.debug("Processing member: " + member);

        if (member instanceof MessageStatement)
        {
            MessageStatement msg = (MessageStatement) member;

            if (!visitedMessages.contains(msg)){
            visitedMessages.add(msg);

            if (msg.secondReferedObject() instanceof MethodTemporary
            {
                logger.debug("Ignoring message as it's to a temp vari
            }
            else
            {
                Objeto msgTarget = getObject((Concept) msg.secondRefe
                expandMethod(target, msgTarget, msg.calledMethod());
            }                   }
        }
    }
}

/**
 * Ã§Ã£Descrio de uma frase do Ã©mtodo. Ã§Ã£Descrio mais detalhada
 * deixando claro o que o Ã©mtodo deve fazer, coisas que ele assume,
```

```
     * resultados esperados, enfim, qualquer coisa que facilite quem for
     * sem conhecimento Ã©prvio.
     *
     * @param object
     * @return
     */
    private Objeto getObject(Concept object)
    {
        Objeto obj = objetos.get(object);
        if (obj == null)
        {
            Classe targetType = null;

            if (object instanceof TypedObject)
            {
                TypedObject type = (TypedObject) object;
                targetType = type.type().relatedClass();
            }
            else if (object instanceof Classe)
            {
                targetType = (Classe) object;
            }

            obj = createObject(dseq, objects, targetType.name());
            objetos.put(object, obj);
        }
        return obj;
    }

    /**
     * Ã§Ã£Descrio de uma frase do Ã©mtodo. Ã§Ã£Descrio mais detalhada
     * deixando claro o que o Ã©mtodo deve fazer, coisas que ele assume,
     * resultados esperados, enfim, qualquer coisa que facilite quem for
     * sem conhecimento Ã©prvio.
     *
```

```
     * @param caller
     * @param target
     * @param calledMethod
     */
    private void createMessage(Objeto caller, Objeto target, Metodo called
    {
        Mensagem msg = (Mensagem) dseq.createComponent_with_and(Mensagem.
        msg.atorOrigem(caller);
        msg.atorDestino(target);
        msg.metodoDestino(calledMethod);
        msg.ordem(msgOrder++);
        // TODO ver procedimento para criacao da figura de mensagem
        ScenarioMessageFigure msgFig = new ScenarioMessageFigure();
        msgFig.concept(msg);
        dseq.drawing().add(msgFig);
    }

    private Objeto createObject(Cenario dseq, int objIndex, String classT
    {
        Objeto obj = (Objeto) dseq.createComponent_named(Objeto.class, cl

        if (obj == null)
        {
            throw new RuntimeException("Couldn't create object: " + class
        }

        ScenarioObjectFigure sof = new ScenarioObjectFigure();

        dseq.drawing().add(sof);
        sof.setMaxH(2048);
        sof.concept(obj);
        obj.getObservable().addObserver(sof);
        Figure objName = ((ScenarioObjectFigure) sof).getObjectName();
        dseq.drawing().add(objName);
        obj.redraw();
```

```
            sof.moveBy(OBJ_WIDTH * objIndex, 0);
            return obj;

        }
}

/*
 * @(#)UseCaseExtractor.java
 *
 * Historico de modificacoes:
 * Autor                    Data                    Descricao
 * ------------------------ ----------------------- ----------------------------
 */
package ocean.documents.oo.tools.reverseengineering.extractors;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import ocean.documents.oo.concepts.MessageStatement;
import ocean.documents.oo.concepts.Metodo;
import ocean.documents.oo.concepts.UseCase;
import ocean.documents.oo.models.UseCaseDiagram;
import ocean.framework.Concept;
import ocean.jhotdraw.ConceptualFigure;

/**
 * ï¿½ï¿½Descrio de uma frase da classe. ï¿½ï¿½Descrio mais detalhada da cla
 * deixando claro o que ela deve fazer, coisas que assume, resultados esp
 * enfim, qualquer coisa que facilite quem for ï¿½us-la sem conhecimento ï
 *
 * @author otavio
 * @since 12/10/2005
 */
public class UseCaseExtractor extends ParseTimeExtractor
{
```

```
/**
 * Logger da classe.
 *
 * @see org.apache.log4j.Logger
 */
private static final org.apache.log4j.Logger logger = org.apache.log4
    .getLogger(UseCaseExtractor.class);

private ArrayList<Metodo> metodos;

private ArrayList<Metodo> calledMethods;

private HashMap<UseCase, Metodo> useCasesExtracted;

/**
 *
 */
public UseCaseExtractor()
{
    super();
    metodos = new ArrayList<Metodo>();
    calledMethods = new ArrayList<Metodo>();
    useCasesExtracted = new HashMap<UseCase, Metodo>();
}

/*
 * (non-Javadoc)
 *
 * @see ocean.documents.oo.tools.reverseengineering.extractors.Parse
 */
private void conceptPopped(Concept concept)
{

}
```

```
/*
 * (non-Javadoc)
 *
 * @see ocean.documents.oo.tools.reverseengineering.extractors.Parse
 */
private void conceptPushed(Concept concept)
{
    logger.debug("Pushed: " + concept.conceptName() + " " + concept.n
    if (concept instanceof Metodo)
    {
        metodos.add((Metodo) concept);
    }
    else
    {
        throw new IllegalArgumentException("Unexpected concept class:
    }
}

/*
 * (non-Javadoc)
 *
 * @see ocean.documents.oo.tools.reverseengineering.extractors.Parse
 */
@Override
public List<Class< ? extends Concept>> getConcepts()
{
    List<Class< ? extends Concept>> concepts = new ArrayList<Class< ?

    concepts.add(Metodo.class);
    concepts.add(MessageStatement.class);

    return concepts;
}
```

```
/*
 * (non-Javadoc)
 *
 * @see ocean.documents.oo.tools.reverseengineering.extractors.ParseT
 */
@Override
public void parseTimeEvent(ParseTimeEvent event)
{
    switch (event.getEventType())
    {
        case PUSHED:
            conceptPushed(event.getConcept());
            break;
        case POPED:
            conceptPopped(event.getConcept());
            break;
        case INSERTED:
            conceptInserted(event.getConcept());
            break;
        default:
            logger.warn("Unregonized event type: " + event.getEventTy
    }
}

/**
 * Ã§Ã£Descrio de uma frase do Ã©mtodo. Ã§Ã£Descrio mais detalhada
 * deixando claro o que o Ã©mtodo deve fazer, coisas que ele assume,
 * resultados esperados, enfim, qualquer coisa que facilite quem for
 * sem conhecimento Ã©prvio.
 *
 * @param concept
 */
private void conceptInserted(Concept concept)
{
    logger.debug("Concept inserted: " + concept);
```

```
        MessageStatement msg = (MessageStatement) concept;

        logger.debug("Adding " + msg.calledMethod() + " to the list of ca
        calledMethods.add(msg.calledMethod());
    }

    @Override
    public void run()
    {
        this.metodos.removeAll(calledMethods);

        UseCaseDiagram useCaseDiagram = (UseCaseDiagram) specification()
            .createConceptualModel_withName(UseCaseDiagram.class, "Use ca

        logger.debug("Methods that start a use case: " + metodos);

        int positionY = 10;

        for (Metodo m : metodos)
        {
            logger.debug("Creating UC: " + m.classe().name() + "_" + m.na
            UseCase uc = (UseCase) useCaseDiagram.createComponent_named(
                UseCase.class, m.classe().name() + "_" + m.name());
            insertUseCaseDescription(m, uc);
            useCaseDiagram.drawing().createFigureFor(uc);

            ConceptualFigure figure = useCaseDiagram.drawing().getFigureC
            figure.moveBy(0,positionY);
            positionY += figure.size().getHeight() + 10;
        }
    }

    /**
     * Ã§Ã£Descrio de uma frase do Ã©mtodo. Ã§Ã£Descrio mais detalhada
     * deixando claro o que o Ã©mtodo deve fazer, coisas que ele assume,
```

```
     * resultados esperados, enfim, qualquer coisa que facilite quem for
     * sem conhecimento Ã©prvio.
     *
     * @param m
     * @param concept
     */
    private void insertUseCaseDescription(Metodo m, UseCase concept)
    {
        useCasesExtracted.put(concept, m);
    }

    public Metodo getSequenceStarter(UseCase uc)
    {
        return useCasesExtracted.get(uc);
    }
}


/*
 * Copyright Â© 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa
 * California 95054, U.S.A. All rights reserved.  Sun Microsystems, Inc.
 * intellectual property rights relating to technology embodied in the pr
 * that is described in this document. In particular, and without limitat
 * these intellectual property rights may include one or more of the U.S.
 * patents listed at http://www.sun.com/patents and one or more additiona
 * patents or pending patent applications in the U.S. and in other countr
 * U.S. Government Rights - Commercial software. Government users are sub
 * to the Sun Microsystems, Inc. standard license agreement and applicabl
 * provisions of the FAR and its supplements.  Use is subject to license
 * Sun,  Sun Microsystems,  the Sun logo and  Java are trademarks or regi
 * trademarks of Sun Microsystems, Inc. in the U.S. and other countries.
This
 * product is covered and controlled by U.S. Export Control laws and may
 * subject to the export or import laws in other countries.  Nuclear, mi
 * chemical biological weapons or nuclear maritime end uses or end users,
 * whether direct or indirect, are strictly prohibited.  Export or reexpo
```

```
 * to countries subject to U.S. embargo or to entities identified on U.S.
 * export exclusion lists, including, but not limited to, the denied pers
 * and specially designated nationals lists is strictly prohibited.
 */

/** EDIT by Otavio
  * --> Problemas encontrados:
  *   -> <EOF> precisa estar numa linha soh dele aparentemente,
  *       pelo menos, nao pode vir depois de um comentario...
  */
options {
  JAVA_UNICODE_ESCAPE = true;
  ERROR_REPORTING = false;
  STATIC = false;
  DEBUG_PARSER = true;
}

PARSER_BEGIN(JavaParser)

package ocean.documents.oo.tools.reverseengineering;

import java.util.*;
import ocean.documents.oo.tools.reverseengineering.util.expression.*;

/**
 * ï¿½Gramtica para engenharia reversa de ï¿½cdigo Java 1.5, para o ambien
 * Baseado na ï¿½verso ï¿½disponvel na pagina do JavaCC.
 * Grammar to parse Java version 1.5
 * @author Sreenivasa Viswanadha - Simplified and enhanced for 1.5
 */
public class JavaParser extends OceanParser
{
        public String getLanguageName() { return "Java"; }

        public Token lastToken()
```

```
        {
        return token;
        }

        public void run()
        {
                try
                {
                        CompilationUnit();
                }
                catch (ParseException e)
                {
                        throw new RuntimeException(e);
                }
        }
}

PARSER_END(JavaParser)

/* ï¿½espao em branco */

SKIP :
{
  " "
| "\t"
| "\n"
| "\r"
| "\f"
}

/* ï¿½comentrios - atualmente, ignorados pelo parser */

/* para manter compatibilidade estrita com o padrao, os comentarios dever
como:
MORE :
```

```
{
  "//" : IN_SINGLE_LINE_COMMENT
|
  <"/**" ~["/"]> { input_stream.backup(1); } : IN_FORMAL_COMMENT
|
  "/*" : IN_MULTI_LINE_COMMENT
}
mas fazendo isso, um arquivo terminando com um SINGLE_LINE_COMMENT sem um
arquivo vai ser considerado invalido. O compilador padrao do java aceita
*/


SPECIAL_TOKEN : {

    < SINGLE_LINE_COMMENT: "//" (~["\n","\r"])* ("\n"|"\r"|"\r\n")? > }

MORE :
{
  <"/**" ~["/"]> { input_stream.backup(1); } : IN_FORMAL_COMMENT
|
  "/*" : IN_MULTI_LINE_COMMENT
}

//<IN_SINGLE_LINE_COMMENT>
//SPECIAL_TOKEN :
//{
//  <SINGLE_LINE_COMMENT: "\n" | "\r" | "\r\n" > : DEFAULT
//}

<IN_FORMAL_COMMENT>
SPECIAL_TOKEN :
{
  <FORMAL_COMMENT: "*/" > : DEFAULT
}
```

```
<IN_MULTI_LINE_COMMENT>
SPECIAL_TOKEN :
{
  <MULTI_LINE_COMMENT: "*/" > : DEFAULT
}

<IN_SINGLE_LINE_COMMENT,IN_FORMAL_COMMENT,IN_MULTI_LINE_COMMENT>
MORE :
{
  < ~[] >
}

/* palavras reservadas e literais */

TOKEN :
{
  < ABSTRACT: "abstract" >
| < ASSERT: "assert" >
| < BOOLEAN: "boolean" >
| < BREAK: "break" >
| < BYTE: "byte" >
| < CASE: "case" >
| < CATCH: "catch" >
| < CHAR: "char" >
| < CLASS: "class" >
| < CONST: "const" >
| < CONTINUE: "continue" >
| < _DEFAULT: "default" >
| < DO: "do" >
| < DOUBLE: "double" >
| < ELSE: "else" >
| < ENUM: "enum" >
| < EXTENDS: "extends" >
| < FALSE: "false" >
| < FINAL: "final" >
```

```
| < FINALLY: "finally" >
| < FLOAT: "float" >
| < FOR: "for" >
| < GOTO: "goto" >
| < IF: "if" >
| < IMPLEMENTS: "implements" >
| < IMPORT: "import" >
| < INSTANCEOF: "instanceof" >
| < INT: "int" >
| < INTERFACE: "interface" >
| < LONG: "long" >
| < NATIVE: "native" >
| < NEW: "new" >
| < NULL: "null" >
| < PACKAGE: "package">
| < PRIVATE: "private" >
| < PROTECTED: "protected" >
| < PUBLIC: "public" >
| < RETURN: "return" >
| < SHORT: "short" >
| < STATIC: "static" >
| < STRICTFP: "strictfp" >
| < SUPER: "super" >
| < SWITCH: "switch" >
| < SYNCHRONIZED: "synchronized" >
| < THIS: "this" >
| < THROW: "throw" >
| < THROWS: "throws" >
| < TRANSIENT: "transient" >
| < TRUE: "true" >
| < TRY: "try" >
| < VOID: "void" >
| < VOLATILE: "volatile" >
| < WHILE: "while" >
}
```

```
/* literais */

TOKEN :
{
  < INTEGER_LITERAL:
        <DECIMAL_LITERAL> (["l","L"])?
      | <HEX_LITERAL> (["l","L"])?
      | <OCTAL_LITERAL> (["l","L"])?
  >
|
  < #DECIMAL_LITERAL: ["1"-"9"] (["0"-"9"])* >
|
  < #HEX_LITERAL: "0" ["x","X"] (["0"-"9","a"-"f","A"-"F"])+ >
|
  < #OCTAL_LITERAL: "0" (["0"-"7"])* >
|
  < FLOATING_POINT_LITERAL:
        (["0"-"9"])+ "." (["0"-"9"])* (<EXPONENT>)? (["f","F","d","D"])?
      | "." (["0"-"9"])+ (<EXPONENT>)? (["f","F","d","D"])?
      | (["0"-"9"])+ <EXPONENT> (["f","F","d","D"])?
      | (["0"-"9"])+ (<EXPONENT>)? ["f","F","d","D"]
  >
|
  < #EXPONENT: ["e","E"] (["+","-"])? (["0"-"9"])+ >
|
  < CHARACTER_LITERAL:
      "'"
      (     (~["'","\\","\n","\r"])
        | ("\\"
            ( ["n","t","b","r","f","\\","'","\""]
            | ["0"-"7"] ( ["0"-"7"] )?
            | ["0"-"3"] ["0"-"7"] ["0"-"7"]
            )
          )
```

```
        )
        "'"
  >
|
  < STRING_LITERAL:
        "\""
        (     (~["\"","\\","\n","\r"])
          | ("\\"
              ( ["n","t","b","r","f","\\","'","\""]
              | ["0"-"7"] ( ["0"-"7"] )?
              | ["0"-"3"] ["0"-"7"] ["0"-"7"]
              )
            )
        )*
        "\""
  >
}

/* identificadores */

TOKEN :
{
  < IDENTIFIER: <LETTER> (<LETTER>|<DIGIT>)* >
|
  < #LETTER:
        [
         "\u0024",
         "\u0041"-"\u005a",
         "\u005f",
         "\u0061"-"\u007a",
         "\u00c0"-"\u00d6",
         "\u00d8"-"\u00f6",
         "\u00f8"-"\u00ff",
         "\u0100"-"\u1fff",
         "\u3040"-"\u318f",
```

```
        "\u3300"-"\u337f",
        "\u3400"-"\u3d2d",
        "\u4e00"-"\u9fff",
        "\uf900"-"\ufaff"
      ]
  >
|
  < #DIGIT:
      [
        "\u0030"-"\u0039",
        "\u0660"-"\u0669",
        "\u06f0"-"\u06f9",
        "\u0966"-"\u096f",
        "\u09e6"-"\u09ef",
        "\u0a66"-"\u0a6f",
        "\u0ae6"-"\u0aef",
        "\u0b66"-"\u0b6f",
        "\u0be7"-"\u0bef",
        "\u0c66"-"\u0c6f",
        "\u0ce6"-"\u0cef",
        "\u0d66"-"\u0d6f",
        "\u0e50"-"\u0e59",
        "\u0ed0"-"\u0ed9",
        "\u1040"-"\u1049"
      ]
  >
}

/* separadores */

TOKEN :
{
  < LPAREN: "(" >
| < RPAREN: ")" >
| < LBRACE: "{" >
```

```
| < RBRACE: "}" >
| < LBRACKET: "[" >
| < RBRACKET: "]" >
| < SEMICOLON: ";" >
| < COMMA: "," >
| < DOT: "." >
| < AT: "@" >
}

/* operadores */

TOKEN :
{
  < ASSIGN: "=" >
| < LT: "<" >
| < BANG: "!" >
| < TILDE: "~" >
| < HOOK: "?" >
| < COLON: ":" >
| < EQ: "==" >
| < LE: "<=" >
| < GE: ">=" >
| < NE: "!=" >
| < SC_OR: "||" >
| < SC_AND: "&&" >
| < INCR: "++" >
| < DECR: "--" >
| < PLUS: "+" >
| < MINUS: "-" >
| < STAR: "*" >
| < SLASH: "/" >
| < BIT_AND: "&" >
| < BIT_OR: "|" >
| < XOR: "^" >
| < REM: "%" >
```

```
| < LSHIFT: "<<" >
| < PLUSASSIGN: "+=" >
| < MINUSASSIGN: "-=" >
| < STARASSIGN: "*=" >
| < SLASHASSIGN: "/=" >
| < ANDASSIGN: "&=" >
| < ORASSIGN: "|=" >
| < XORASSIGN: "^=" >
| < REMASSIGN: "%=" >
| < LSHIFTASSIGN: "<<=" >
| < RSIGNEDSHIFTASSIGN: ">>=" >
| < RUNSIGNEDSHIFTASSIGN: ">>>=" >
| < ELLIPSIS: "..." >
}

/* devido a ï¿½sintxe de ï¿½ï¿½programao ï¿½genrica (generics), os >'s preci
TOKEN :
{
  < RUNSIGNEDSHIFT: ">>>" >
  {
     matchedToken.kind = GT;
     ((Token.GTToken)matchedToken).realKind = RUNSIGNEDSHIFT;
     input_stream.backup(2);
  }
| < RSIGNEDSHIFT: ">>" >
  {
     matchedToken.kind = GT;
     ((Token.GTToken)matchedToken).realKind = RSIGNEDSHIFT;
     input_stream.backup(1);
  }
| < GT: ">" >
}


/*****************************************
```

```
 * A partir daqui ï¿½ a ï¿½ sintxe da linguagem *
 ****************************************/

void CompilationUnit():
{}
{
  { disable_tracing(); }
  [ PackageDeclaration() ]
  ( { addImport(false,"java.lang.*"); } ImportDeclaration() )*
  ( TypeDeclaration() )*
  <EOF>
}

void PackageDeclaration():
{
        NameNode pacote = null;
}
{
  "package"
  pacote = Name() { setNamespace(pacote.getName()); }
  ";"
}

void ImportDeclaration():
{
        boolean fieldImport = false;
        String name = null;
        NameNode nameNode;
}
{
  "import"
  [ "static" {fieldImport = true; } ]
  nameNode = Name()
  [ "." "*" { name = nameNode.getName()+".*"; } ]
  ";" { addImport(fieldImport,name); }
```

```
}

/*
 * Modifiers. We match all modifiers in a single rule to reduce the chanc
 * syntax errors for simple modifier mistakes. It will also enable us to
 * better error messages.
 */

ModifierSet Modifiers():
{
   ModifierSet modifiers = new ModifierSet();
}
{
 (
  LOOKAHEAD(2)
   (
    "public" { modifiers.addModifier(Modifier.PUBLIC); }
   |
    "static" { modifiers.addModifier(Modifier.STATIC); }
   |
    "protected" { modifiers.addModifier(Modifier.PROTECTED); }
   |
    "private" { modifiers.addModifier(Modifier.PRIVATE); }
   |
    "final" { modifiers.addModifier(Modifier.FINAL); }
   |
    "abstract" { modifiers.addModifier(Modifier.ABSTRACT); }
   |
    "synchronized" { modifiers.addModifier(Modifier.SYNCHRONIZED); }
   |
    "native" { modifiers.addModifier(Modifier.NATIVE); }
   |
    "transient" { modifiers.addModifier(Modifier.TRANSIENT); }
   |
    "volatile" { modifiers.addModifier(Modifier.VOLATILE); }
```

```
  |
   "strictfp" { modifiers.addModifier(Modifier.STRICTFP); }
  |
   Annotation()
  )
 )*

 {
    return modifiers;
 }
}

/*
 * ï¿½ï¿½Declarao de tipos.
 */
void TypeDeclaration():
{
   ModifierSet modifiers;
}
{
  ";"
|
  modifiers = Modifiers()
  (
     ClassOrInterfaceDeclaration(modifiers)
   |
     EnumDeclaration(modifiers)
   |
     AnnotationTypeDeclaration(modifiers)
  )
}


void ClassOrInterfaceDeclaration(ModifierSet modifiers):
{
```

```
    boolean isInterface = false;
    Token tk = null;
}
{
  ( "class" | "interface" { isInterface = true; } )
  tk = <IDENTIFIER> { if (isInterface) { pushInterface(tk.image); } else
  [ TypeParameters() ]
  [ ExtendsList(isInterface) ]
  [ ImplementsList(isInterface) ]
  ClassOrInterfaceBody(isInterface)
  { if (isInterface) { popInterface(); } else { popClass(); } }
}

void ExtendsList(boolean isInterface):
{
    String superName = null;
}
{
    "extends"
    superName = ClassOrInterfaceType() { addInheritance(superName); }
    ( ","
          superName = ClassOrInterfaceType()
    {
                if (isInterface)
          throw new ParseException("A class cannot extend more than one ot
             addInheritance(superName);
    }
    )*
}

void ImplementsList(boolean isInterface):
{
        String ifaceName = null;
}
{
```

```
    "implements"
    ifaceName = ClassOrInterfaceType() { addInheritance(ifaceName); }
    ( ","
          ifaceName = ClassOrInterfaceType()
    )*
    {
       if (isInterface)
          throw new ParseException("An interface cannot implement other in
       addInheritance(ifaceName);
    }
}

void EnumDeclaration(ModifierSet modifiers):
{
        Token tk = null;
}
{
  "enum"
  tk = <IDENTIFIER> { pushClass(tk.image); addStereotype(Stereotype.ENUM)
  [ ImplementsList(false) ]
  EnumBody()
}

void EnumBody():
{}
{
    "{"
    EnumConstant() ( "," EnumConstant() )*
    [ ";" ( ClassOrInterfaceBodyDeclaration(false) )* ]
    "}"
}

void EnumConstant():
{
        Token tk = null;
```

```
}
{
  tk = <IDENTIFIER>
  {
      pushVariable(tk.image,getCurrentTypeName());
      ModifierSet ms = new ModifierSet();
      ms.addModifier(Modifier.STATIC);
      ms.addModifier(Modifier.PUBLIC);
      ms.addModifier(Modifier.FINAL);
      setModifiers(ms);
  }
  [ Arguments() ] [ ClassOrInterfaceBody(false) ]
}

void TypeParameters():
{}
{
   "<" TypeParameter() ( "," TypeParameter() )* ">"
}

void TypeParameter():
{}
{
   <IDENTIFIER> [ TypeBound() ]
}

void TypeBound():
{}
{
   "extends" ClassOrInterfaceType() ( "&" ClassOrInterfaceType() )*
}

void ClassOrInterfaceBody(boolean isInterface):
{}
{
```

```
  "{" ( ClassOrInterfaceBodyDeclaration(isInterface) )* "}"
}

void ClassOrInterfaceBodyDeclaration(boolean isInterface):
{
   boolean isNestedInterface = false;
   ModifierSet modifiers;
}
{
  LOOKAHEAD(2)
   Initializer()
   {
      if (isInterface)
         throw new ParseException("An interface cannot have initializers")
   }
|
   modifiers = Modifiers() // Just get all the modifiers out of the way.
                           // more checks, pass the modifiers down to the member
   (
       ClassOrInterfaceDeclaration(modifiers)
     |
       EnumDeclaration(modifiers)
     |
       LOOKAHEAD( [ TypeParameters() ] <IDENTIFIER> "(" )
       ConstructorDeclaration()
     |
       LOOKAHEAD( Type() <IDENTIFIER> ( "[" "]" )* ( "," | "=" | ";" ) )
       FieldDeclaration(modifiers)
     |
       MethodDeclaration(modifiers)
   )
|
   ";"
}
```

```
void FieldDeclaration(ModifierSet modifiers):
{
        String type = null;
}
{
  // Modifiers are already matched in the caller
  type = Type()
  VariableDeclarator(modifiers,type) ( "," VariableDeclarator(modifiers,t
}

void VariableDeclarator(ModifierSet modifiers, String type):
{
        StringBuffer realType = new StringBuffer(type);
        String name = null;
}
{
  name = VariableDeclaratorId(realType) { pushVariable(name,realType.toSt
  [ "=" VariableInitializer() ]
  { popVariable(); }
}

String VariableDeclaratorId(StringBuffer type):
{
        Token tk = null;
}
{
  tk = <IDENTIFIER>
  ( "[" "]" { type.append("[]"); } )*
  { return tk.image; }
}

String VariableInitializer():
{
        String str;
        ExpressionNode en;
```

```
}
{
{ }
(
  str = ArrayInitializer()
|
  en = Expression() { str = en.toString(); }
 )
{ return str; }
}

String ArrayInitializer():
{
        String init= "", tmp;
}
{
  "{" [ init = VariableInitializer() ( LOOKAHEAD(2) "," tmp = VariableIn
  { return "{" + init + "}"; }
}

void MethodDeclaration(ModifierSet modifiers):
{
        String type = null;
        StringBuffer realType = new StringBuffer();
}
{
  // Modifiers already matched in the caller!
  [ TypeParameters() ]
  type = ResultType() { realType = new StringBuffer(type); }
  MethodDeclarator(realType) [ "throws" NameList() ]
  ( Block() | ";"  ) { popMethod(); }
}

void MethodDeclarator(StringBuffer type):
{
```

```
        Token tk = null;
}
{
  tk = <IDENTIFIER> { pushMethod(tk.image); }
  FormalParameters() ( "[" "]" { type.append("[]"); } )*
  { setType(type.toString()); }
}

void FormalParameters():
{}
{
  "(" [ FormalParameter() ( "," FormalParameter() )* ] ")"
}

void FormalParameter():
{
String name,type;
StringBuffer realType;
ModifierSet modifiers = new ModifierSet();
boolean varargs = false;
}
{
  [ "final" { modifiers.addModifier(Modifier.FINAL); } ]
  type = Type() { realType = new StringBuffer(type); }
  [ "..." { varargs = true; } ]
  name = VariableDeclaratorId(realType)
  { insertParameter(modifiers,realType.toString(),name,varargs); }
}

void ConstructorDeclaration():
{
        Token tk = null;
}
{
  [ TypeParameters() ]
```

```
  // Modifiers matched in the caller
  tk = <IDENTIFIER> { pushMethod(tk.image); addStereotype(Stereotype.CONS
  FormalParameters() [ "throws" NameList() ]
  "{"
    [ LOOKAHEAD(ExplicitConstructorInvocation()) {  } ExplicitConstructor
} ]
    ( BlockStatement() )*
  "}" { popMethod(); }
}

void ExplicitConstructorInvocation():
{  List node;
}
{
  LOOKAHEAD("this" Arguments() ";")
  "this" node = Arguments() ";" { insertExpression(new MethodCallNode(new
|
  [ LOOKAHEAD(2) PrimaryExpression() "." ] "super" {  } Arguments() {
} ";"
}

void Initializer():
{}
{
  [ "static" ] Block()
}



/*
 * Type, name and expression syntax follows.
 */

String Type():
{
        String type = null;
```

```
}
{
 (
   LOOKAHEAD(2)
   type = ReferenceType()
 |
   type = PrimitiveType()
   )
  { return type; }
}

String ReferenceType():
{
        StringBuffer dimensions = new StringBuffer();
        String typeName = null;
}
{
   typeName = PrimitiveType()
   ( LOOKAHEAD(2) "[" "]" { dimensions.append("[]"); } )+
  |
   ( typeName = ClassOrInterfaceType() )
   ( LOOKAHEAD(2) "[" "]" { dimensions.append("[]"); } )*
   { return typeName + dimensions; }
}

String ClassOrInterfaceType():
{
        StringBuffer name = new StringBuffer();
        Token tk = null;
}
{
  tk = <IDENTIFIER> { name.append(tk.image); }
  [ LOOKAHEAD(2) TypeArguments() ]
  ( LOOKAHEAD(2) "."
  tk = <IDENTIFIER>  { name.append("." + tk.image); }
```

```
  [ LOOKAHEAD(2) TypeArguments() ] )*
  { return name.toString(); }
}

String TypeArguments():
{
        String type, tmp;
}
{
   "<" type = TypeArgument() ( "," tmp = TypeArgument() { type += ","+tmp
   { return "<"+type+">"; }
}

String TypeArgument():
{
        String type;
}
{
(
   type = ReferenceType()
 |
   "?" { type = "?"; } [ type = WildcardBounds() { type = "?"+type; }]
)
{ return type; }
}

String WildcardBounds():
{
String type;
}
{
   "extends" type = ReferenceType() { return "extends" + type; }
 |
   "super" type = ReferenceType() { return "super" + type; }
}
```

```
String PrimitiveType():
{
        Token tk = null;
}
{
 (
  tk = "boolean"
|
  tk = "char"
|
  tk = "byte"
|
  tk = "short"
|
  tk = "int"
|
  tk = "long"
|
  tk = "float"
|
  tk = "double"
  )
  { return tk.image; }
}

String ResultType():
{
        String type = null;
}
{
(
  "void" { type = "void"; }
|
```

```
  type = Type()
)
{ return type; }
}

NameNode Name():
/*
 * Um LOOKAHED de 2 ï¿½ï¿½posies ï¿½ï¿½ necessria para remover a ambiguidade
 * import id.id.*;
 * e
 * import id.id;
 * por exemplo.
 */
{
        Token tk = null;
        NameNode name = null;
}
{
  tk = <IDENTIFIER>  { name = new NameNode(tk.image); }
  ( LOOKAHEAD(2) "."
    tk = <IDENTIFIER> { name = new NameNode(tk.image,name); }
  )*
  { return name; }
}

void NameList():
{}
{
  Name() ( "," Name() )*
}



/*
 * Expression syntax follows.
 */
```

```
ExpressionNode Expression():
/*
 * This expansion has been written this way instead of:
 *   Assignment() | ConditionalExpression()
 * for performance reasons.
 * However, it is a weakening of the grammar for it allows the LHS of
 * assignments to be any conditional expression whereas it can only be
 * a primary expression.  Consider adding a semantic predicate to work
 * around this.
 */
{
        ExpressionNode expNode, rhs;
}
{
  expNode = ConditionalExpression()
  [
    LOOKAHEAD(2)
    { expNode = new AssignmentNode(expNode); } ((AssignmentNode)expNode).
    rhs = Expression() { ((AssignmentNode)expNode).rhs = rhs; }
  ]
  { expNode.update(); return expNode; }
}

String AssignmentOperator():
{
}
{
  ( "=" | "*=" | "/=" | "%=" | "+=" | "-=" | "<<=" | ">>=" | ">>>=" | "&=
        { return token.image; }
}

ExpressionNode ConditionalExpression():
{
        ExpressionNode node, condTrue, condFalse;
```

```
}
{
  node = ConditionalOrExpression()
  [ "?" condTrue = Expression() ":"
    condFalse = Expression() { node = new IfNode(node,condTrue,condFalse)
  ]
  {  return node; }
}

ExpressionNode ConditionalOrExpression():
{
ExpressionNode node, node1;
}
{
  node = ConditionalAndExpression() ( "||" node1 = ConditionalAndExpressi
  { return node; }
}

ExpressionNode ConditionalAndExpression():
{
ExpressionNode node, node1;
}
{
  node = InclusiveOrExpression() ( "&&" node1 = InclusiveOrExpression() {
  { return node; }
}

ExpressionNode InclusiveOrExpression():
{
ExpressionNode node, node1;
}
{
  node = ExclusiveOrExpression() ( "|" node1 = ExclusiveOrExpression() {
  { return node; }
 }
```

```
ExpressionNode ExclusiveOrExpression():
{
ExpressionNode node, node1;
}
{
node = AndExpression() ( "^" node1 = AndExpression() { node = new Arithme
  { return node; }
}

ExpressionNode AndExpression():
{
ExpressionNode node, node1;
}
{
  node = EqualityExpression() ( "&" node1 = EqualityExpression() { node =
  { return node; }
}

ExpressionNode EqualityExpression():
{
ExpressionNode node, node1;
Token tk;
}
{
  node = InstanceOfExpression() ( ( tk = "==" | tk = "!=" ) node1 = Insta
    { return node; }

}

ExpressionNode InstanceOfExpression():
{
ExpressionNode en;
String tmp;
}
```

```
{
  en = RelationalExpression() [ "instanceof" tmp = Type() { en = new Type
    { return en; }

}

ExpressionNode RelationalExpression():
{
ExpressionNode node, node1;
Token tk;
}
{
  node = ShiftExpression() ( ( tk = "<" | tk = ">" | tk = "<=" | tk = ">=
    { return node; }

}

ExpressionNode ShiftExpression():
{
String tmp;
ExpressionNode node, node1;
}
{
  node = AdditiveExpression()
   (
      ( "<<" { tmp = "<<"; }  | tmp = RSIGNEDSHIFT() | tmp = RUNSIGNEDSHI
      node1 = AdditiveExpression() { node = new ArithmeticNode(node,node1
   )*
   { return node; }
}

ExpressionNode AdditiveExpression():
{
Token tk;
ExpressionNode node, node1;
```

```
}
{
  node = MultiplicativeExpression() ( ( tk = "+" | tk = "-" ) node1 = Mu
 { return node; }
}

ExpressionNode MultiplicativeExpression():
{
ExpressionNode node, tmpNode;
Token tk;
}
{
  node = UnaryExpression()
     ( ( tk = "*" | tk = "/" | tk = "%" ) tmpNode = UnaryExpression() { n
 { return node; }
}

ExpressionNode UnaryExpression():
{
Token tk;
ExpressionNode exp = null;
}
{
(
  ( tk = "+" | tk = "-" ) exp = UnaryExpression() { exp = new UnaryExpres
|
  exp = PreIncrementExpression()
|
  exp = PreDecrementExpression()
|
  exp = UnaryExpressionNotPlusMinus()
)
{ return exp; }
}
```

```
ExpressionNode PreIncrementExpression():
{
ExpressionNode exp;
}
{
  "++" exp = PrimaryExpression()  { return new PreUpdateNode(exp,true); }
}

ExpressionNode PreDecrementExpression():
{
ExpressionNode exp;
}
{
  "--" exp = PrimaryExpression() { return new PreUpdateNode(exp,false); }
}

ExpressionNode UnaryExpressionNotPlusMinus():
{
String tmp;
Token tk;
ExpressionNode exp;
}
{
(
  ( tk = "~" | tk = "!" ) exp = UnaryExpression() { exp = new UnaryExpres
|
  LOOKAHEAD( CastLookahead() )
  exp = CastExpression()
|
  exp = PostfixExpression()
)
  { return exp;}
}

// This production is to determine lookahead only.  The LOOKAHEAD specifi
```

```
// below are not used, but they are there just to indicate that we know a
// this.
void CastLookahead():
{}
{
  LOOKAHEAD(2)
  "(" PrimitiveType()
|
  LOOKAHEAD("(" Type() "[")
  "(" Type() "[" "]"
|
  "(" Type() ")" ( "~" | "!" | "(" | <IDENTIFIER> | "this" | "super" | "n
}

ExpressionNode PostfixExpression():
{
        ExpressionNode exp;
}
{
  exp = PrimaryExpression() [
    "++" { exp = new PostUpdateNode(exp,true); }
    |
    "--" { exp = new PostUpdateNode(exp,false); }
  ] { return exp;}
}

ExpressionNode CastExpression():
{
String str;
ExpressionNode en = null;
}
{
(
  LOOKAHEAD("(" PrimitiveType())
  "(" str = Type() ")" en = UnaryExpression() { en = new CastNode(str,en)
```

```
|
  "(" str = Type() ")" en = UnaryExpressionNotPlusMinus() { en = new Cast
)
{ return en; }
}

ExpressionNode PrimaryExpression():
{
        ExpressionNode exp = null;
        ExpressionNode tmp;
}
{
  exp = PrimaryPrefix() { exp = new PrimaryExpressionNode(exp); }
  ( LOOKAHEAD(2)
    tmp = PrimarySuffix() { ((PrimaryExpressionNode)exp).addSuffix(tmp);
  )*
  {
    return exp;
  }
}

String MemberSelector():
{
        String args = "";
        Token tk;
}
{
  "." args = TypeArguments() tk = <IDENTIFIER> { return "."+args+tk.image
}

ExpressionNode PrimaryPrefix():
{
   String prefix = "";
   Token tk;
   ExpressionNode en;
```

```
}
{
(
  en = Literal()
|
  "this" { en = new NameNode("this"); }
|
  "super" "." tk = <IDENTIFIER> { en = new ExpressionNode("@@SUPER@@"+tk.
|
  "(" en = Expression() ")"
|
  en = AllocationExpression()
|
  LOOKAHEAD( ResultType() "." "class" )
  prefix = ResultType() "." "class" { en = new ExpressionNode("@@CLASS@@"
|
  en = Name()
)
{
   return en;
}
}

ExpressionNode PrimarySuffix():
{
        ExpressionNode exp = null;
        Token tk = null;
        String tmp = null;
        List args = null;
}
{
(
  LOOKAHEAD(2)
  "." "this" { exp = new NameNode("this"); }
|
```

```
  LOOKAHEAD(2)
  "." exp = AllocationExpression()
|
  LOOKAHEAD(3)
  tmp = MemberSelector() { exp = new ExpressionNode("@@MEMBER_SEL@@"+tmp)
|
  "[" exp = Expression()  "]" { }
|
  "." tk = <IDENTIFIER> { exp = new NameNode(tk.image); }
|
  args = Arguments() { exp = new MethodCallNode(null,args); }
)
{ return exp; }
}

ExpressionNode Literal():
{
  ExpressionNode en;
  Token tk = null;
}
{
(
  tk = <INTEGER_LITERAL> { en = new LiteralNode("int",tk.image); }
|
  tk = <FLOATING_POINT_LITERAL> { en = new LiteralNode("float",tk.image);
|
  tk = <CHARACTER_LITERAL> { en = new LiteralNode("char",tk.image); }
|
  tk = <STRING_LITERAL> { en = new LiteralNode("String",tk.image); }
|
  en = BooleanLiteral()
|
  en = NullLiteral()
 )
 { return en; }
```

```
}

ExpressionNode BooleanLiteral():
{}
{
  "true" { return new LiteralNode("boolean","true"); }
|
  "false" { return new LiteralNode("boolean","false"); }
}

ExpressionNode NullLiteral():
{}
{
  "null" { return new LiteralNode("null","null"); }
}

List Arguments():
{
        List args = new ArrayList();
}
{
  "(" [ args = ArgumentList() ] ")" { return args; }
}

List ArgumentList():
{
        ExpressionNode tmp;
        List argList = new ArrayList();
}
{
  tmp = Expression() { argList.add(tmp); } ( "," tmp = Expression() { arg
  { return argList; }
}

ExpressionNode AllocationExpression():
```

```
{
        ExpressionNode node;
        String tmp, str;
        List args = null;
}
{
{  }
(
  LOOKAHEAD(2)
  "new" tmp = PrimitiveType() str = ArrayDimsAndInits() { node = new Arra
|
  "new" tmp = ClassOrInterfaceType() [ TypeArguments() ] { node = new Nar
    (
      str = ArrayDimsAndInits() { node = new ArrayNode(node,str); }
    |
      args = Arguments() { node = new MethodCallNode(node,args); } [ Clas
    )
)
{ return node; }
}

/*
 * The third LOOKAHEAD specification below is to parse to PrimarySuffix
 * if there is an expression between the "[...]".
 */
String ArrayDimsAndInits():
{
        String str = "", tmpStr = "";
        ExpressionNode tmp;
}
{
(
  LOOKAHEAD(2)
  ( LOOKAHEAD(2) "[" tmp = Expression() "]" { str += "["+tmp.toString()+"
|
```

```
  ( "[" "]" { str += "[]"; } )+ tmpStr = ArrayInitializer() { str += tmpS
)
{return str;}
}


/*
 * Statement syntax follows.
 */

void Statement():
{
}
{
{  }
(
  LOOKAHEAD(2)
  LabeledStatement()
|
  AssertStatement()
|
  Block()
|
  EmptyStatement()
|
  StatementExpression() ";"
|
  SwitchStatement()
|
  IfStatement()
|
  WhileStatement()
|
  DoStatement()
|
```

```
  ForStatement()
|
  BreakStatement()
|
  ContinueStatement()
|
  ReturnStatement()
|
  ThrowStatement()
|
  SynchronizedStatement()
|
  TryStatement()
)
{   }
}

void AssertStatement():
{}
{
  "assert" Expression() [ ":" Expression() ] ";"
}

void LabeledStatement():
{}
{
  <IDENTIFIER> ":" Statement()
}

void Block():
{}
{
  "{" { pushBlock(); } ( BlockStatement() )* { popBlock(); } "}"
}
```

```
void BlockStatement():
{
}
{
  LOOKAHEAD([ "final" ] Type() <IDENTIFIER>)
  LocalVariableDeclaration() ";"
|
  { } Statement() {  }
|
  ClassOrInterfaceDeclaration(null)
}

void LocalVariableDeclaration():
{
        ModifierSet modifiers = new ModifierSet();
        String type = null;
}
{
  [ "final" { modifiers.addModifier(Modifier.FINAL); } ]
  type = Type()
  VariableDeclarator(modifiers,type) ( "," VariableDeclarator(modifiers,t
}

void EmptyStatement():
{}
{
  ";"
}

void StatementExpression():
/*
 * The last expansion of this production accepts more than the legal
 * Java expansions for StatementExpression.  This expansion does not
 * use PostfixExpression for performance reasons.
 */
```

```
{
        ExpressionNode lhs = null;
        String oper = "";
        boolean isExpression = false;
        boolean assignment = false;
        ExpressionNode expression;
}
{
(
  expression = PreIncrementExpression()
|
  expression = PreDecrementExpression()
|
  (
  expression = PrimaryExpression() { isExpression = true; }
  [
    "++" { expression = new PostUpdateNode(expression,true); }
  |
     "--" { expression = new PostUpdateNode(expression,false); }
  |
    { expression = new AssignmentNode(expression); }
    ((AssignmentNode)expression).operator = AssignmentOperator()
    lhs = Expression() { ((AssignmentNode)expression).rhs = lhs; }
  ]
  )
)
  {
        insertExpression(expression);
  }
}

void SwitchStatement():
{}
{
  "switch" "(" Expression() ")" "{"
```

```
    ( SwitchLabel() ( BlockStatement() )* )*
  "}"
}

void SwitchLabel():
{}
{
  "case" Expression() ":"
|
  "default" ":"
}

void IfStatement():
/*
 * The disambiguating algorithm of JavaCC automatically binds dangling
 * else's to the innermost if statement.  The LOOKAHEAD specification
 * is to tell JavaCC that we know what we are doing.
 */
{
}
{
  "if" "(" Expression() ")"
    Statement()
   [ LOOKAHEAD(1) "else" Statement() ]
}

void WhileStatement():
{}
{
  "while" "(" Expression() ")" Statement()
}

void DoStatement():
{}
{
```

```
  "do" Statement() "while" "(" Expression() ")" ";"
}

void ForStatement():
{}
{
  "for" "("

  (
      LOOKAHEAD(Type() <IDENTIFIER> ":")
      Type() <IDENTIFIER> ":" Expression()
    |
      [ ForInit() ] ";" [ Expression() ] ";" [ ForUpdate() ]
  )

  ")" Statement()
}

void ForInit():
{}
{
  LOOKAHEAD( [ "final" ] Type() <IDENTIFIER> )
  LocalVariableDeclaration()
|
  StatementExpressionList()
}

void StatementExpressionList():
{}
{
  StatementExpression() ( "," StatementExpression() )*
}

void ForUpdate():
{}
```

```
{
  StatementExpressionList()
}

void BreakStatement():
{}
{
  "break" [ <IDENTIFIER> ] ";"
}

void ContinueStatement():
{}
{
  "continue" [ <IDENTIFIER> ] ";"
}

void ReturnStatement():
{}
{
  "return" [ Expression() ] ";"
}

void ThrowStatement():
{}
{
  "throw" Expression() ";"
}

void SynchronizedStatement():
{}
{
  "synchronized" "(" Expression() ")" Block()
}

void TryStatement():
```

```
/*
 * Semantic check required here to make sure that at least one
 * finally/catch is present.
 */
{}
{
  "try" Block()
  ( "catch" "(" FormalParameter() ")" Block() )*
  [ "finally" Block() ]
}

/* We use productions to match >>>, >> and > so that we can keep the
 * type declaration syntax with generics clean
 */

String RUNSIGNEDSHIFT():
{}
{
  ( LOOKAHEAD({ getToken(1).kind == GT &&
                ((Token.GTToken)getToken(1)).realKind == RUNSIGNEDSHIFT}
   ">" ">" ">" { return ">>>"; }
  )
}

String RSIGNEDSHIFT():
{}
{
  ( LOOKAHEAD({ getToken(1).kind == GT &&
                ((Token.GTToken)getToken(1)).realKind == RSIGNEDSHIFT} )
  ">" ">" { return ">>"; }
  )
}

/* Annotation syntax follows. */
```

```
void Annotation():
{}
{
    LOOKAHEAD( "@" Name() "(" ( <IDENTIFIER> "=" | ")" ))
    NormalAnnotation()
  |
    LOOKAHEAD( "@" Name() "(" )
    SingleMemberAnnotation()
  |
    MarkerAnnotation()
}

void NormalAnnotation():
{}
{
    "@" Name() "(" [ MemberValuePairs() ] ")"
}

void MarkerAnnotation():
{}
{
  "@" Name()
}

void SingleMemberAnnotation():
{}
{
  "@" Name() "(" MemberValue() ")"
}

void MemberValuePairs():
{}
{
    MemberValuePair() ( "," MemberValuePair() )*
}
```

```
void MemberValuePair():
{}
{
    <IDENTIFIER> "=" MemberValue()
}

void MemberValue():
{}
{
    Annotation()
  |
    MemberValueArrayInitializer()
  |
    ConditionalExpression()
}

void  MemberValueArrayInitializer():
{}
{
  "{" MemberValue() ( LOOKAHEAD(2) "," MemberValue() )* [ "," ] "}"
}


/* Annotation Types. */

void AnnotationTypeDeclaration(ModifierSet modifiers):
{}
{
  "@" "interface" <IDENTIFIER> AnnotationTypeBody()
}

void AnnotationTypeBody():
{}
{
```

```
  "{" ( AnnotationTypeMemberDeclaration() )* "}"
}

void AnnotationTypeMemberDeclaration():
{
   ModifierSet modifiers;
}
{
 modifiers = Modifiers()
 (
   LOOKAHEAD(Type() <IDENTIFIER> "(")
   Type() <IDENTIFIER> "(" ")" [ DefaultValue() ] ";"
  |
   ClassOrInterfaceDeclaration(modifiers)
  |
   EnumDeclaration(modifiers)
  |
   AnnotationTypeDeclaration(modifiers)
  |
   FieldDeclaration(modifiers)
 )
 |
   ( ";" )
}

void DefaultValue():
{}
{
  "default" MemberValue()
}

/* Generated By:JavaCC: Do not edit this line. Token.java Version 3.0 */
package ocean.documents.oo.tools.reverseengineering;

/**
 * Describes the input token stream.
```

```
 */

public class Token {

  /**
   * An integer that describes the kind of this token.  This numbering
   * system is determined by JavaCCParser, and a table of these numbers
   * stored in the file ...Constants.java.
   */
  public int kind;

  /**
   * beginLine and beginColumn describe the position of the first charac
   * of this token; endLine and endColumn describe the position of the
   * last character of this token.
   */
  public int beginLine, beginColumn, endLine, endColumn;

  /**
   * The string image of the token.
   */
  public String image;

  /**
   * A reference to the next regular (non-special) token from the input
   * stream.  If this is the last token from the input stream, or if the
   * token manager has not read tokens beyond this one, this field is
   * set to null.  This is true only if this token is also a regular
   * token.  Otherwise, see below for a description of the contents of
   * this field.
   */
  public Token next;

  /**
   * This field is used to access special tokens that occur prior to this
```

```
   * token, but after the immediately preceding regular (non-special) tok
   * If there are no such special tokens, this field is set to null.
   * When there are more than one such special token, this field refers
   * to the last of these special tokens, which in turn refers to the nex
   * previous special token through its specialToken field, and so on
   * until the first special token (whose specialToken field is null).
   * The next fields of special tokens refer to other special tokens that
   * immediately follow it (without an intervening regular token).
If there
   * is no such token, this field is null.
   */
  public Token specialToken;

  /**
   * Returns the image.
   */
  public String toString()
  {
     return image;
  }

  /**
   * Returns a new Token object, by default. However, if you want, you
   * can create and return subclass objects based on the value of ofKind.
   * Simply add the cases to the switch for all those special cases.
   * For example, if you have a subclass of Token called IDToken that
   * you want to create if ofKind is ID, simlpy add something like :
   *
   *    case MyParserConstants.ID : return new IDToken();
   *
   * to the following switch statement. Then you can cast matchedToken
   * variable to the appropriate type and use it in your lexical actions.
   */
  public static final Token newToken(int ofKind)
  {
```

```
      switch(ofKind)
      {
        default : return new Token();
        case JavaParserConstants.RUNSIGNEDSHIFT:
        case JavaParserConstants.RSIGNEDSHIFT:
        case JavaParserConstants.GT:
           return new GTToken();
      }
    }

    public static class GTToken extends Token
    {
      int realKind = JavaParserConstants.GT;
    }
}
```

ANEXO B – Artigo