MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
SECRETARIA DE CIÊNCIA E TECNOLOGIA
CURSO DE MESTRADO EM SISTEMAS DE COMPUTAÇÃO

SÉRGIO AUGUSTO FREITAS FILHO

DETECÇÃO DE PADRÕES DE CÓDIGO EM WMLSCRIPT

Rio de Janeiro
2006

INSTITUTO MILITAR DE ENGENHARIA

SÉRGIO AUGUSTO FREITAS FILHO

DETECÇÃO DE PADRÕES DE CÓDIGO EM WMLSCRIPT

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.

Orientador: Alex de Vasconcellos Garcia – D.C.

Rio de Janeiro
2006

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro – RJ – CEP 22290-270

F866d Freitas Filho, Sérgio Augusto
Detecção de padrões de código em WMLScript/
Sérgio Augusto Freitas Filho.io de Janeiro: Instituto Militar de Engenharia, 2006.
151 p.:il, tab.
Dissertação: (mestrado) – Instituto Militar de Engenharia, Rio de Janeiro, 2006.
1. Engenharia de Sistemas e Computação. 2. Linguagens de Programação. 4. WMLScript (linguagem de programação de computador). I. Título. II. Instituto Militar de Engenharia.

CDD 005.133

INSTITUTO MILITAR DE ENGENHARIA

SÉRGIO AUGUSTO FREITAS FILHO

DETECÇÃO DE PADRÕES DE CÓDIGO EM WMLSCRIPT

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.

Orientador: Alex de Vasconcellos Garcia – D.C.

Aprovada em 28 de abril de 2006 pela seguinte Banca Examinadora:

Prof. Alex de Vasconcellos Garcia – D.C. do IME – Presidente

Prof. Fernando Náufel do Amaral – D.C. da UERJ

Prof. Luiz Carlos de Castro Guedes – D.C. da UFF

Rio de Janeiro
2006

AGRADECIMENTOS

Gostaria de agradecer às pessoas que tiveram uma importante participação na realização deste trabalho a quem gostaria de expressar minha imensa gratidão.

Ao meu querido pai Sérgio Augusto Freitas, pelo amor, carinho, paciência e compreensão principalmente durante esses dois anos, marcados pela distância e ausência por uma infelicidade do destino.

À minha irmã Joanina Pereira Freitas, avó Joanina Capello Freitas e ao meu avô Wilson Freitas que teria grande orgulho deste trabalho, mas infelizmente não vive mais entre nós.

Aos meus companheiros de república, Diogo, Ricardo, Pablo, Flávio, Alemão, Maurício, entre outros, pelo companheirismo durante esses anos de convivência.

Aos amigos que deixei em Juiz de Fora.

Ao Prof. Alex pela amizade, confiança, dedicação e competência com que conduziu a orientação deste trabalho.

Ao instituto Militar de Engenharia, professores e funcionários do Departamento de Engenharia de Sistemas, e a todas as pessoas que, direta ou indiretamente, contribuíram para o desenvolvimento deste trabalho.

A universidade Federal de Juiz de Fora pela minha formação acadêmica consistente.

A CAPES, pelo suporte financeiro proporcionado durante estes dois anos, sem o qual seria impossível a realização desse sonho.

E finalmente, gostaria de agradecer a minha amada namorada Cristiane Rivelli, pelo amor, carinho e infinita paciência para suportar os momentos de ausência durante todo esse tempo.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

API	APPLICATION PROGRAM INTERFACE
AST	ABSTRACT SYNTAX TREE
DOM	DOCUMENT OBJECT MODEL
JDT	JAVA DEVELOPMENT TOOLING
PDL	PATTERN DESCRIPTION LANGUAGE
PDM	PATTERN DETECT MACHINE
XML	EXTENSIBLE MARKUP LANGUAGE
WAP	WIRELESS APPLICATION PROTOCOL
WDT	WMLSCRIPT DEVELOPMENT TOOLING
WPD	WMLSCRIPT PATTERN DETECTOR

RESUMO

Este trabalho apresenta uma ferramenta de detecção de padrões de codificação para a linguagem *WMLScript*. Em nossa pesquisa não encontramos outra ferramenta equivalente para esta linguagem. Em comparação com ferramentas similares para outras linguagens a ferramenta é mais flexível e permite a criação de novos padrões de forma fácil e intuitiva. A ferramenta baseou-se em um trabalho análogo desenvolvido neste instituto para a linguagem Java.

Ilustramos o uso da ferramenta através da implementação de um conjunto de padrões de codificação para a linguagem *WMLScript*. Não temos conhecimento de um padrão de codificação publicado para a linguagem. Assim, apresentamos como uma contribuição adicional um conjunto de padrões de código para *WMLScript*. Finalmente, para validarmos a funcionalidade da ferramenta e a utilidade dos padrões propostos, foi feito um estudo de caso aplicando-se os padrões a dezesseis programas de domínio público.

ABSTRACT

This work presents a code pattern detection tool for the WMLScript language. In our research we did not find another similar tool for this language. In comparison with similar tools for other languages, this tool is more flexible and allows the creation of new patterns in an easy and intuitive way. The tool was based on an analogous work developed in this institute for the Java language.

We illustrate the use of this tool by implementing a set of code patterns for the WMLScript language. We are not aware of any code pattern published for the language. Thus, we present as an additional contribution a set of code patterns for WMLScript. Finally, to validate the functionality of the tool and the utility of the considered patterns, a case study was made applying the patterns in sixteen public domain applications.

1 INTRODUÇÃO

Em 2005, Albuquerque apresentou neste instituto a dissertação: “Detecção de Padrões de Código Java na Plataforma Eclipse” [Albuquerque, 2005]. Seu trabalho apresentou o conceito de padrão como sendo uma árvore de símbolos terminais e não-terminais da linguagem Java. A ferramenta consiste em um algoritmo que encontra ocorrências dos padrões especificados na AST (árvore sintática abstrata) que representa os programas. Albuquerque introduziu ainda a descrição destes padrões numa linguagem chamada PDL (*Pattern Description Language*) com sintaxe XML [XML]. Esta abordagem revelou-se muito útil e flexível, sendo fácil criar um novo padrão e incorporá-lo à ferramenta.

No presente trabalho apresentamos o desenvolvimento de uma ferramenta análoga para a linguagem *WMLScript* [WMLSCRIPT, 2000]. No nosso conhecimento não há outra ferramenta similar disponível para esta linguagem, nem tampouco um conjunto de padrões indicado para disciplinar a codificação na linguagem. Para desenvolver esta ferramenta, criamos um analisador sintático para *WMLScript* e o integramos ao algoritmo desenvolvido por Albuquerque.

No intuito de apresentar um caso de utilização da ferramenta, foi necessário criar um conjunto de padrões de codificação *WMLScript*. Nesta dissertação apresentamos estes padrões, justificando-os e codificando-os em PDL. Os padrões de código definidos nesta dissertação foram resultado de um levantamento de práticas de codificação em outras linguagens e um estudo específico de *WMLScript*. Os padrões têm como objetivos a padronização da nomenclatura, legibilidade, desempenho, diminuição dos custos de manutenção e prevenção de erros nas aplicações. Apresentamos também os resultados do caso de utilização.

1.1 MOTIVAÇÃO

Constatamos um interesse no desenvolvimento de aplicações utilizando a linguagem *WMLScript*. Por exemplo, a empresa UPT [UPT] utiliza o *WMLScript* na implementação de

jogos, a Divas *Software* [DIVAS] no desenvolvimento de aplicações empresariais e a AKMOSOFT [AKMOSOFT] em sistemas de informação customizados cliente/servidor para *web*, todas utilizam ou já utilizaram o *WMLScript*, na maioria das vezes conjugado a outras linguagens e tecnologias. Entretanto não encontramos ferramentas para apoiar o desenvolvimento dos programas nesta linguagem. Neste cenário propomos a implementação da ferramenta de detecção de padrões para auxiliar na etapa de codificação do processo de desenvolvimento de *software* com a linguagem *WMLScript*.

A inspeção de código é um processo inerentemente humano, pois lida com a correção do código que é um processo não computável. No entanto, acreditamos que o uso da ferramenta de detecção de padrões pode complementar uma inspeção humana, sendo mais eficaz na busca de padrões repetitivos.

1.2 OBJETIVOS DA DISSERTAÇÃO

Nosso primeiro objetivo foi desenvolver uma ferramenta de detecção de padrões de código para a linguagem *WMLScript*. A ferramenta baseou-se em uma ferramenta análoga, para linguagem Java, descrita em [Albuquerque, 2005]. Reutilizamos o algoritmo de detecção de padrões e desenvolvemos um *parser* integrado à plataforma Eclipse [ECLIPSE, 2003]. O *parser* foi desenvolvido utilizando o gerador JavaCC.

Para validarmos a ferramenta, buscou-se um conjunto de padrões de codificação *WMLScript* para formalizá-los em PDL. Uma vez que não encontramos tal padronização estabelecemos como objetivo definir um conjunto de padrões para o *WMLScript.* Na dissertação apresentamos os padrões e suas respectivas implementações em PDL. Os padrões foram divididos em categorias como convenção de nomenclatura, estruturas de controle, legibilidade e desempenho. Vamos justificar a utilização de cada padrão e apontar os benefícios que podem ser alcançados.

Para verificação da funcionalidade da ferramenta e o impacto da utilização dos padrões nos programas, realizamos um estudo de caso e esperamos com os resultados obtidos comprovar a eficácia da aplicação dos padrões na melhoria dos sistemas desenvolvidos na linguagem.

1.3 TRABALHOS RELACIONADOS

Constatamos a existência de ferramentas que trabalham no processo de inspeção de código mas não encontramos nenhuma que trabalhasse com a linguagem *WMLScript*; ainda assim vamos descrever algumas das ferramentas que encontramos na literatura.

Podemos entender a *Lint* [JOHNSON, 1978] como uma ferramenta precursora na identificação de padrões. A ferramenta gera *warnings* para o usuário ao encontrar certos padrões. A *Lint* busca por padrões nos programas escritos em C e não permite a inclusão de novos padrões.

Mais recentemente surgiram as ferramentas para inspeção de código Java, a PMD [PMD] é uma ferramenta *open source* que possui *plug-ins* para diversas plataformas de desenvolvimento como JDeveloper, Eclipse, JEdit, NetBeans, TextPad, etc. A PMD possui um conjunto de padrões de código pré-definidos, contudo apresenta uma complexidade na criação de novos padrões. A ferramenta *FindBugs* [FINDBUGS] também *open source* tem como diferencial em relação às outras a procura por *bugs* em código compilado, ou seja, em *bytecode.* Encontramos ainda uma ferramenta de uso comercial a *CodePro Advisor* [CODEPRO], que pereceu ser a mais completa, além da detecção de padrões, ela possui funcionalidades para o cálculo de métricas, análise de dependências em nível de acoplamento entre projetos, pacotes e classes, reparação de JavaDoc [JAVA], entre outros.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

Organizamos a dissertação no seguinte formato: no capítulo 2 apresentamos a ferramenta *Pattern Detect Machine* (PDM) implementada por [Albuquerque, 2005] para detecção de padrões de código Java na plataforma Eclipse, explicamos o funcionamento da ferramenta, a arquitetura técnica e o algoritmo de detecção de padrões, sempre referenciando as questões que serão tratadas para realizar a detecção dos padrões de código no *WMLScript.* No capítulo 3 mostramos como funciona o JavaCC um gerador de analisadores léxicos e sintáticos em Java utilizado para gerar o analisador sintático do *WMLScript*. No capítulo 4 construímos o analisador sintático do *WMLScript* no JavaCC e implementamos um modelo de classes para

representação da estrutura da AST dos programas. No capítulo 5 descrevemos a ferramenta de detecção de padrões de código implementada, explicamos os aspectos da integração entre a ferramenta PDM, o analisador sintático e a plataforma Eclipse. O capítulo 6 apresenta a definição do conjunto de padrões de código que serão utilizados na análise dos programas. No capítulo 7 realizamos um estudo de caso em programas implementados na linguagem *WMLScript*, utilizando os padrões definidos. Finalmente, no capítulo 8, apresentamos as conclusões do trabalho, as contribuições realizadas e sugestões de trabalhos futuros.

2 DETECÇÃO DE PADRÕES DE CÓDIGO COM A FERRAMENTA PATTERN DETECT MACHINE (PDM)

A *Pattern Detect Machine* (PDM) [ALBUQUERQUE, 2005] é uma ferramenta de detecção de padrões de código Java integrada ao ambiente Eclipse. O algoritmo de detecção de padrões de código da PDM realiza a busca dos padrões na AST dos programas. A PDM utiliza a infraestrutura disponível na plataforma Eclipse [ECLIPSE, 2003] para realizar a análise sintática nos programas Java e obter uma representação da AST dos programas. Neste trabalho reutilizamos o algoritmo de detecção de padrões da PDM, entretanto foi necessário implementar um analisador sintático para o *WMLScript* e realizar a integração deste com o algoritmo de detecção.

A PDM disponibiliza uma linguagem para descrição dos padrões de código (PDL), que permite definir os padrões em arquivos estruturados com XML [XML], a PDL é formada por elementos correspondentes aos nós que implementam a AST dos programas. Consideramos a PDL um diferencial da PDM para criação de novos padrões de código, a utilização do padrão XML possibilita uma estruturação simples e apropriada para descrição dos padrões. Os requisitos para criação dos padrões de código com a PDL são: conhecer a estrutura da AST da linguagem, compreender como funcionam os atributos da PDL e estar familiarizado com o XML. Na próxima seção vamos explicar em detalhes como a PDL deve ser utilizada, ilustramos com exemplos em *WMLScript* como proceder na criação dos padrões de código.

Neste capítulo vamos abordar como utilizar a PDL para implementação dos padrões de codificação. Apresentamos ainda características da arquitetura da PDM e do algoritmo de detecção de padrões.

2.1 LINGUAGEM PARA DESCRIÇÃO DOS PADRÕES (PDL)

A linguagem PDL permite a criação de novos padrões de forma muito mais simples que as ferramentas existentes. Por exemplo, na ferramenta *CodePro Advisor* [CODEPRO], para definir um novo padrão de código é preciso estender o *framework* do Eclipse, conhecer

conceitos de desenvolvimento de *plugins*, programar em Java para implementar as regras, além de conhecer o modelo da AST da linguagem. Na ferramenta PMD [PMD], novos padrões podem ser definidos em *XPath* [XPATH] ou Java, os padrões mais complexos são difíceis de serem descritos.

Albuquerque [ALBUQUERQUE, 2005] avaliou outras ferramentas além das citadas no parágrafo anterior e frente às dificuldades recorrentes em todas para criação de novos padrões de código, surgiu a idéia da PDL, uma linguagem desenvolvida no padrão XML e composta basicamente por elementos que correspondem aos nós da AST e atributos que auxiliam na definição dos padrões. A PDL permite definir novos padrões de código sem que seja necessário ao usuário possuir habilidades em programação.

2.1.1 DESCRIÇÃO DOS PADRÕES SINTÁTICOS

Os elementos e atributos da PDL são particulares para cada linguagem de programação e são definidos de acordo com a estrutura da AST utilizada para representar os programas. Albuquerque definiu a PDL para Java de acordo com a estrutura da AST implementada na plataforma Eclipse, nesta dissertação a PDL para *WMLScript* foi definida de acordo com a estrutura da AST gerada pelo analisador sintático implementado para linguagem (capítulo 4). A seguir ilustramos a criação de um padrão de código em PDL, utilizamos um padrão didático que busca por mais de uma variável declarada em uma mesma instrução. Vamos analisar uma função em *WMLScript* que possui uma ocorrência do padrão.

```
extern function teste() {
  var x,y;
}
```

FIG. 2.1 Função em *WMLScript*.

O analisador sintático desenvolvido nesta dissertação gera a AST apresentada na FIG. 2.2. Os mesmos nomes dos nós serão usados na criação do padrão.

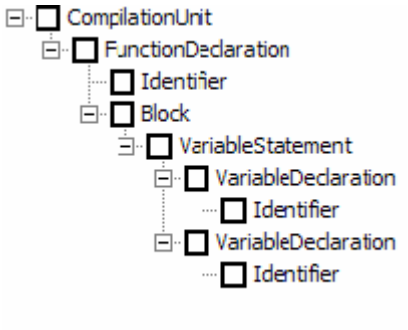

FIG. 2.2 AST gerada pelo analisador sintático do *WMLScript*.

Primeiramente, vamos ilustrar uma árvore que representa o padrão de código que estamos procurando. Na FIG. 2.3 interpretamos o padrão como duas variáveis declaradas em uma mesma instrução.

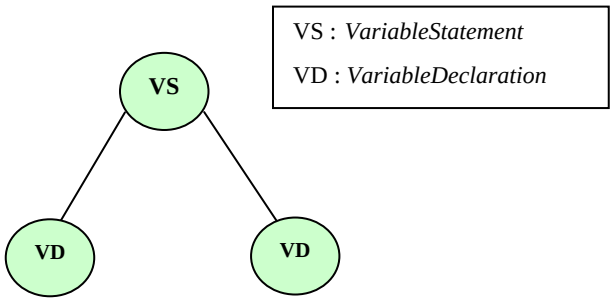

FIG. 2.3 Mais de uma variável declarada em uma mesma instrução.

O padrão de código é escrito em PDL com dois elementos *VariableDeclaration* filhos do elemento *VariableStatement*. Haverá uma ocorrência do padrão sempre que o algoritmo detectar na AST duas ou mais variáveis declaradas em uma mesma instrução. Na FIG. 2.4 mostramos uma implementação do padrão de código na PDL.

```
<VariableStatement>
    <VariableDeclaration/>
    <VariableDeclaration/>
</VariableStatement>
```

FIG. 2.4 Padrão definido em PDL.

Finalmente, mostramos a associação do padrão com sua ocorrência na árvore sintática, o leitor pode perceber que há apenas uma ocorrência do padrão na AST, a associação existente é obtida automaticamente pela ferramenta de detecção de padrões.

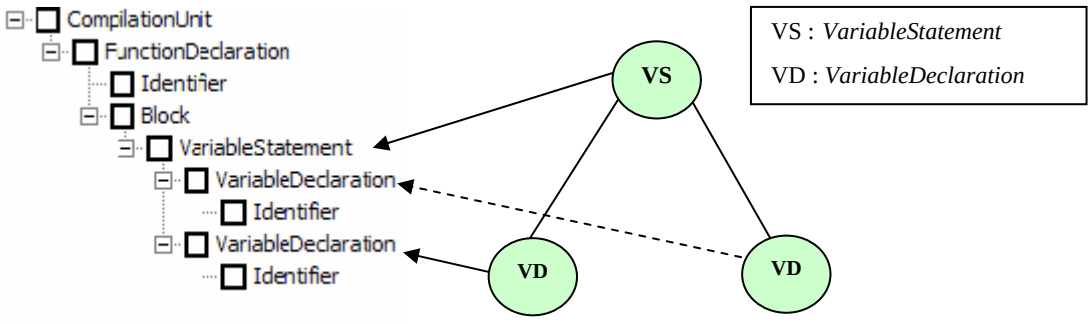

FIG. 2.5 Casamento do padrão na AST do programa *WMLScript*.

2.1.2 ATRIBUTOS DA PDL

A PDL é formada por um conjunto de atributos especiais e um conjunto com os atributos específicos de cada linguagem de programação (ou atributos simples). Nesta dissertação os atributos específicos do *WMLScript* serão descritos em detalhes no capítulo 5. Para exemplificar o uso de um atributo simples apresentamos o padrão na FIG. 2.6, onde acessamos o atributo "nome" de um elemento *Identifier* em uma *FunctionDeclaration*, isto significa que estamos acessando o nome de uma função declarada no *WMLScript*. O padrão

utilizado neste exemplo simula a busca por declarações de funções com o nome iniciado por letra maiúscula, valendo-se do casamento com uma expressão regular.

```
<FunctionDeclaration>
    <Identifier nome="[A-Z].*"/>
</FunctionDeclaration>
```

FIG. 2.6 Padrão em PDL utilizando o atributo “nome” do elemento *Identifier*.

Os atributos especiais definem comportamentos interessantes para definição dos padrões de código. Descreveremos, a seguir, cada um desses atributos apresentando como eles devem ser utilizados.

2.1.2.1 ATRIBUTO _MARK

A fim de melhorar a usabilidade da ferramenta de detecção de padrões, o atributo *_Mark* permite incluir marcações no editor do Eclipse para os padrões encontrados e inserir um texto informativo para o usuário. Para exemplificar o atributo *_Mark* consideramos novamente o padrão de código que busca ocorrências de duas ou mais variáveis declaradas em uma mesma instrução.

```
<VariableStatement _Mark=”Evite declarar mais de uma variável em um
mesmo comando.”>
    <VariableDeclaration/>
    <VariableDeclaration/>
</VariableStatement>
```

FIG. 2.7 Padrão utilizando o atributo *_Mark*.

O atributo *_Mark* localiza os padrões no código fonte dos programas e informa ao usuário as ocorrências encontrados. Para tanto, é utilizada a interface ilustrada a seguir no ambiente Eclipse.

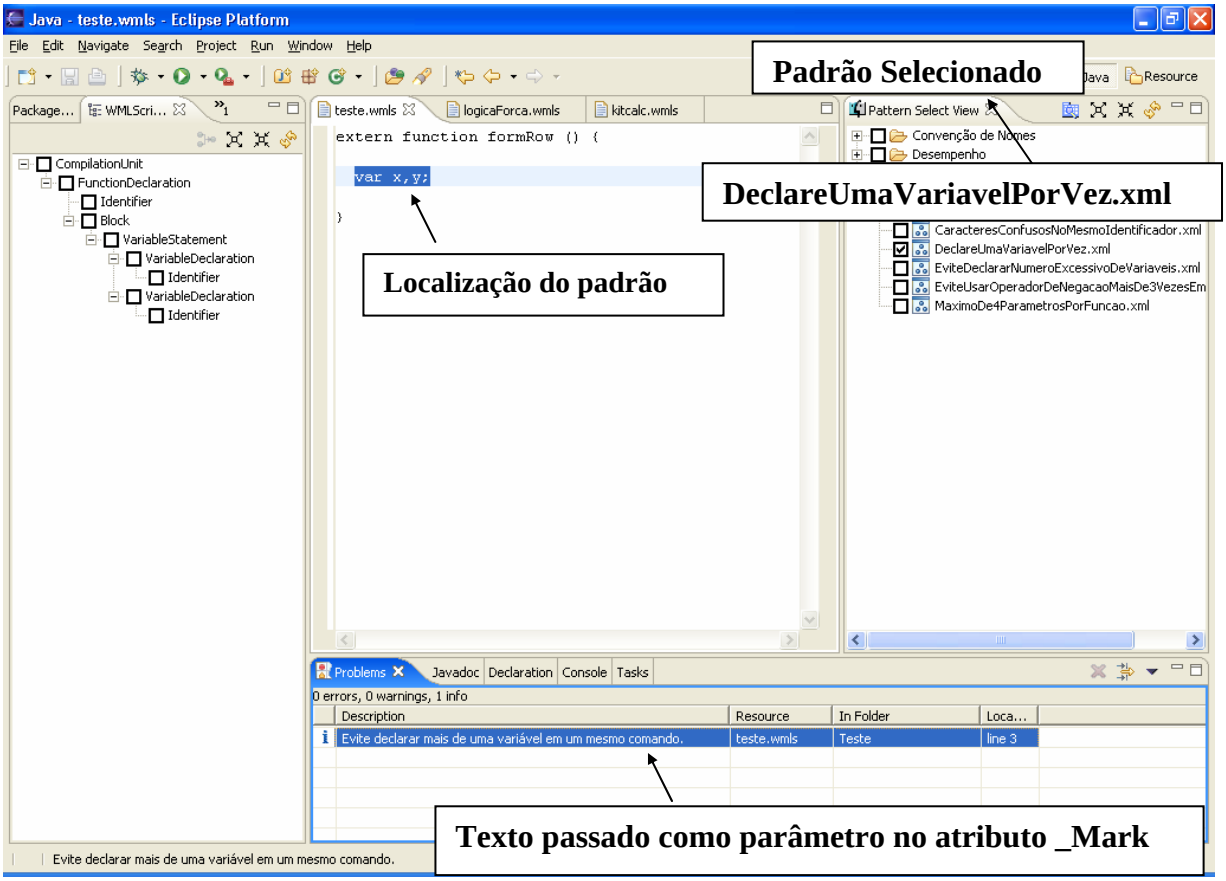

FIG. 2.8 Exemplo da utilização do atributo *_Mark* no ambiente Eclipse.

2.1.2.2 ATRIBUTO *_NOT*

Ao marcar um elemento da PDL com o atributo *_Not* definimos que o mesmo não deve ocorrer na posição definida pelo padrão. Para caracterizar uma ocorrência de um padrão que utiliza o atributo *_Not* o algoritmo deve localizar uma subárvore na AST dos programas que corresponda à árvore do padrão sem a presença do elemento marcado com *_Not*. Vamos

mostrar como o atributo *_Not* funciona em um padrão ilustrativo implementado para o *WMLScript* na categoria estruturas de controle que busca *while* sem bloco de chaves.

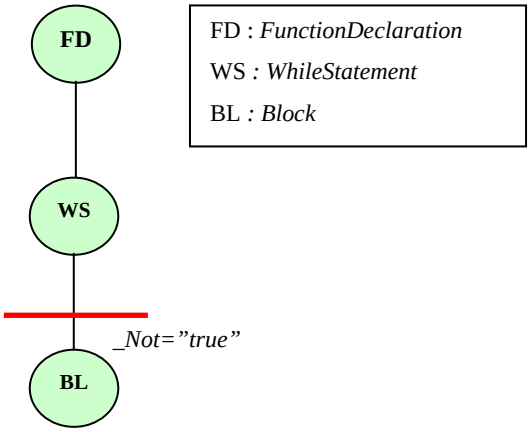

FIG. 2.9 Ilustração de padrão utilizando o atributo *_Not*.

A seguir definimos em PDL o mesmo padrão e mostramos como o atributo *_Not* deve ser instanciado em um elemento. O atributo indica que o nó não deverá ocorrer na posição indicada pelo padrão.

```
<FunctionDeclaration>
    <WhileStatement _Mark="While sem bloco com chaves.">
        <Block _Not="true"/>
    </WhileStatement>
</FunctionDeclaration>
```

FIG. 2.10 Padrão em PDL utilizando o atributo *_Not*.

Na FIG. 2.11 separamos uma função do *WMLScript* que possui uma estrutura de *while* sem bloco de chaves, ao executar o algoritmo de detecção de padrões, ele deverá encontrar uma ocorrência do padrão.

```
//FunctionDeclaration
extern function teste(){

  //VariableStatement
  var x = 0;

  //WhileStatement
  while (x == 0);
}
```

FIG. 2.11 Função *WMLScript* com uma ocorrência do padrão.

Utilizamos o *parser* implementado nesta dissertação para gerar a AST do código *WMLScript*. Na FIG. 2.12 mostramos a associação do padrão com sua ocorrência na árvore sintática, o leitor pode observar que o nó *WhileStatement* não possui nenhum filho *Block*. Neste exemplo encontramos uma ocorrência do padrão.

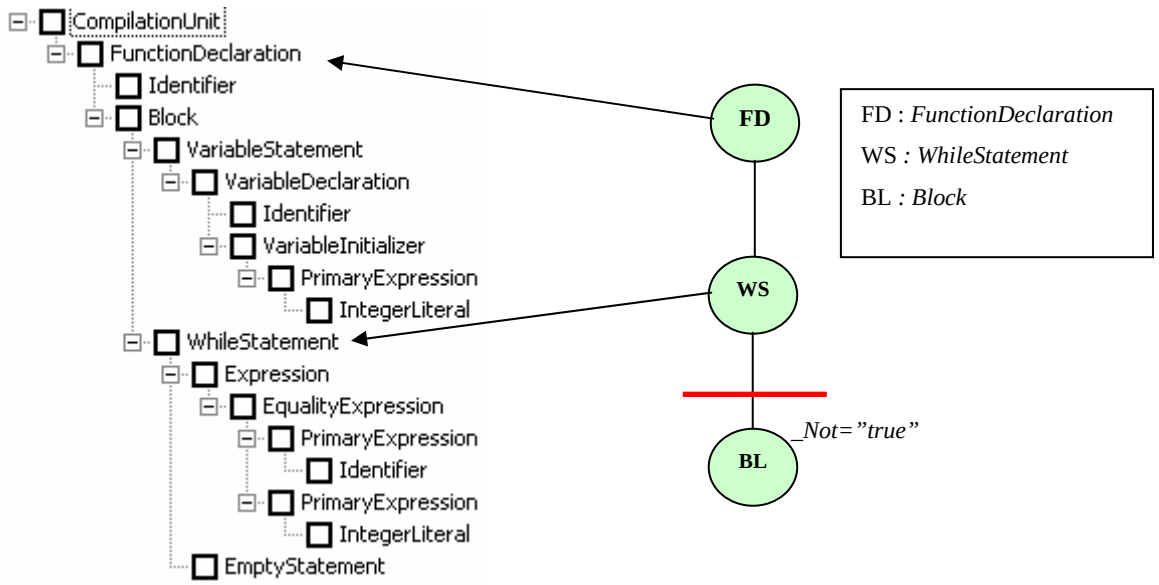

FIG. 2.12 Casamento do padrão na AST do programa *WMLScript*.

2.1.2.3 ATRIBUTO *_INPARENTSUBTREE*

O atributo *_InParentSubTree* é utilizado para identificar em qual subárvore do elemento pai o nó marcado deverá ocorrer. Utilizamos como exemplo um padrão didático implementado para o *WMLScript* na categoria desempenho, o mesmo sugere ao desenvolvedor evitar testes comparativos explícitos envolvendo valores booleanos. No exemplo, o nó *EqualityExpression* que representa a instrução comparativa deve ocorrer na subárvore *Expression* de *IfStatement*.

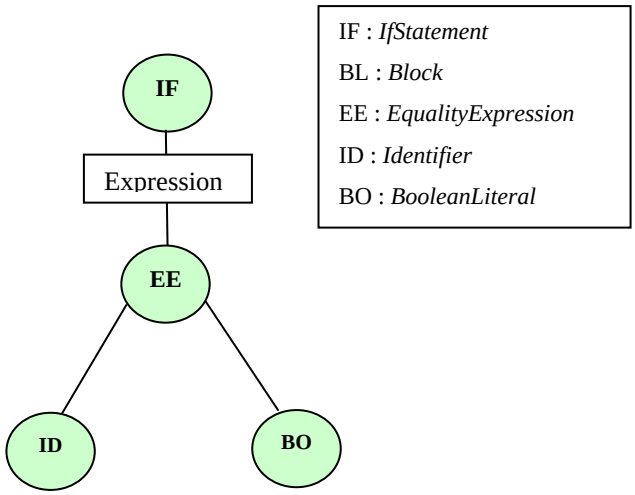

FIG. 2.13 Ilustração de padrão utilizando o atributo *_ InParentSubTree*.

Definimos então em PDL o padrão de código proposto utilizando o atributo *_InParentSubTree*, observe que instanciamos o atributo com o valor *Expression* indicando que o elemento marcado deverá ocorrer na subárvore *Expression* do nó *IfStatement.*

```
<IfStatement _Mark="Evite testes comparativos explícitos">
    <EqualityExpression _InParentSubTree="Expression">
           <Identifier/>
           <BooleanLiteral/>
    </EqualityExpression>
</IfStatement>
```

FIG. 2.14 Padrão em PDL utilizando o atributo *_InParentSubTree*.

A seguir escrevemos em *WMLScript* uma função com uma ocorrência do padrão na condição do *if* . Ao realizar um teste comparativo envolvendo valor booleano, se omitirmos o valor booleano o compilador do *WMLScript* atinge um desempenho melhor em sua execução.

```
//FunctionDeclaration
extern function teste(){
    //IfStatement
    if ( a != true)  //evite
    {}
}
```

FIG. 2.15 Função com teste explícito na condição do *loop*.

Utilizamos o analisador sintático implementado para o *WMLScript* para gerar a AST que representa o código fonte da função. Ao executar o algoritmo de detecção de padrões ele realiza a busca na AST gerada. Na FIG. 2.16 ilustramos as associações que são obtidas automaticamente pelo algoritmo. Neste exemplo detectamos uma única ocorrência do padrão.

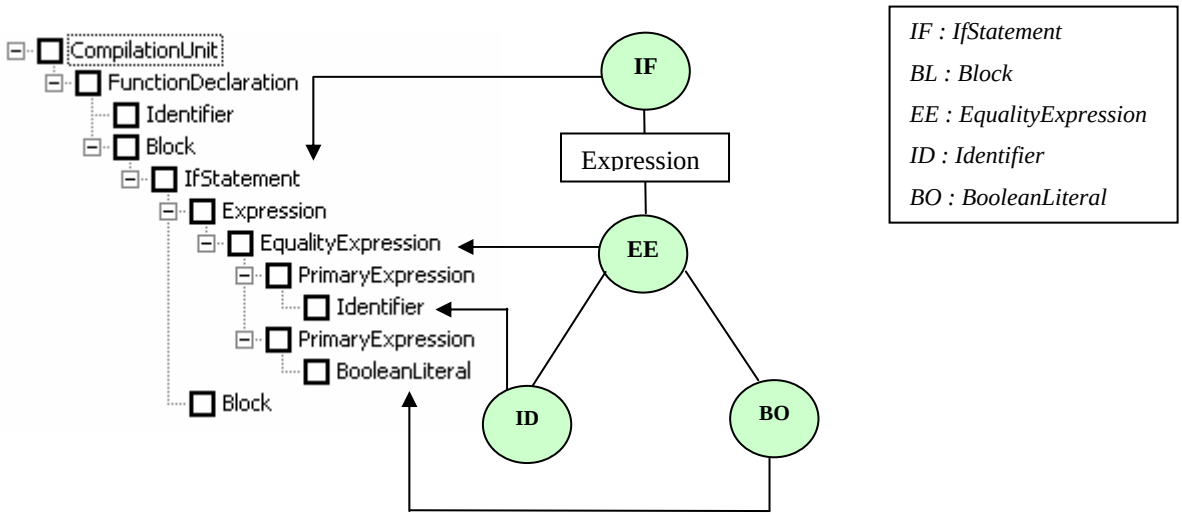

FIG. 2.16 Casamento do padrão na AST do programa *WMLScript*.

2.1.2.4 ATRIBUTO *_MAXSUBTREELEVEL*

Define um nível máximo de profundidade para ocorrência de determinado nó, tomando como referência o pai do nó marcado. Para ilustrar o funcionamento deste atributo utilizamos o padrão do *WMLScript* que busca por funções sem conteúdo, ou seja, funções com o corpo vazio. No exemplo um nó *Block* só pode aparecer como filho imediato de *FunctionDeclaration*. Conseguimos impor a restrição proposta instanciando o atributo *_MaxSubTreeLevel* do elemento *Block* com o valor um, indicando que o elemento *Block* deve ocorrer em um nível imediatamente abaixo do elemento *FunctionDeclaration*.

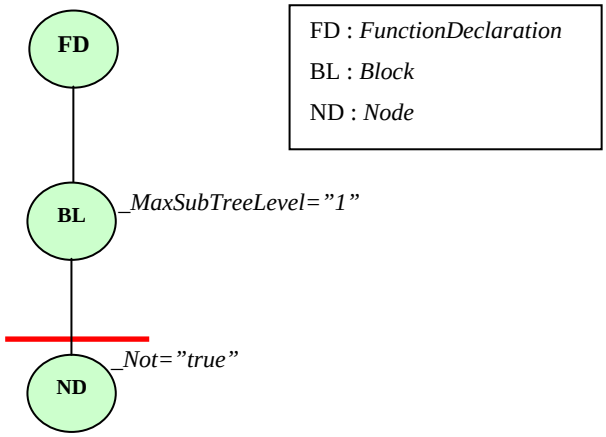

FIG. 2.17 Ilustração do padrão utilizando o atributo _ *MaxSubTreeLevel*.

O mesmo padrão foi definido em PDL na FIG. 2.18, este padrão utiliza o nó *Node* representando qualquer elemento da PDL, já o atributo *_MaxSubTreeLevel* foi definido no elemento *Block* indicando que este deve ocorrer como filho imediato de *FunctionDeclaration*.

```
<FunctionDeclaration _Mark="Função com Corpo Vazio.">
    <Block _MaxSubTreeLevel="1">
        <Node _Not="true"/>
    </Block>
</FunctionDeclaration>
```

FIG. 2.18 Padrão em PDL utilizando o atributo _ *MaxSubTreeLevel*.

Existe uma relação de herança entre os tipos de nós da AST. Por exemplo, todos os tipos de nós são subclasses da classe *Node.* Esse recurso é usado, por exemplo, no padrão da FIG. 2.18. O modelo de classes com todas as relações de herança é apresentado no capítulo 4.

Para ilustrar a utilização do atributo _ *MaxSubTreeLevel* analisamos o código da função *WMLScript* na FIG. 2.19.

```
//FunctionDeclaration
extern function teste(){}
```

FIG. 2.19 Função em *WMLScript* sem conteúdo.

Finalmente mostramos as associações obtidas automaticamente pela ferramenta de detecção de padrões para a ocorrência do padrão na AST da função.

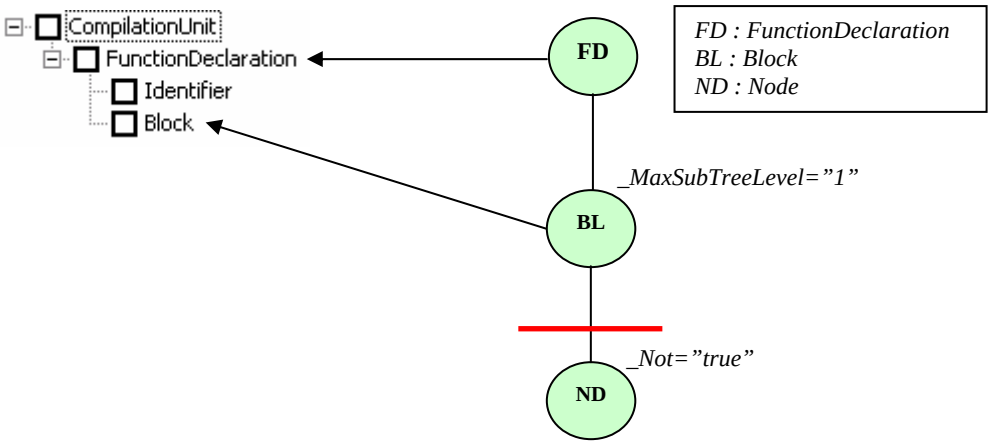

FIG. 2.20 Casamento do padrão na AST do programa *WMLScript*.

2.1.2.5 ATRIBUTO *_BIND*

Correlaciona atributos de dois ou mais elementos. Possui dois parâmetros a serem preenchidos, um funciona como marcador e o outro o atributo a ser ligado. No exemplo a seguir do *WMLScript*, buscamos por alguma ocorrência da variável de controle do *for* sendo atualizada no corpo da própria estrutura. Para definir este padrão temos que identificar a variável de controle do *for* no corpo da estrutura de controle. Na FIG. 2.21 ilustramos uma árvore que representa o padrão de código utilizando o atributo *_Bind.*

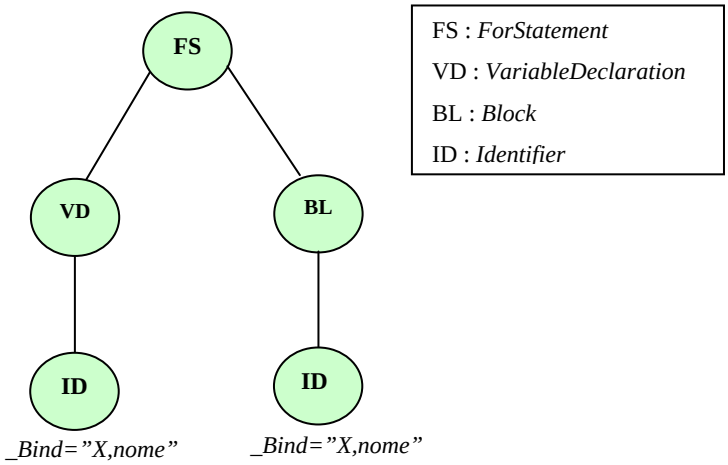

FIG. 2.21 Ilustração do padrão utilizando o atributo _ *Bind*.

O atributo *_Bind* deve ser utilizado nos elementos que representam a variável de controle do *for* na declaração e no corpo da estrutura de controle. Na FIG. 2.22 definimos o padrão em PDL e ilustramos a utilização do atributo *_Bind*, observe que o primeiro elemento *Identifier* marcado no padrão representa a variável de controle sendo declarada e o segundo elemento *Identifier* marcado está no corpo do *for* dentro do elemento *Block*.

```
<ForStatement _Mark="For com variavel de controle atualizada no corpo">
    <VariableDeclaration _MaxSubTreeLevel="1">
        <Identifier _Bind="X,nome" />
    </VariableDeclaration>
    <Block _MaxSubTreeLevel="1">
        <Identifier _Bind="X,nome" />
    </Block>
</ForStatement>
```

FIG. 2.22 Padrão em PDL utilizando o atributo _ *Bind*.

A seguir definimos uma função em *WMLScript* que possui uma ocorrência do padrão, ou seja, declaramos um *for* e alteramos a variável de controle dentro da própria estrutura.

```
extern function formRow () {
   for (var i = 0; i < 100 ; ++i ){
       i++;
   }
}
```

FIG. 2.23 Função com variável de controle sendo atualizada dentro do *for*.

Utilizamos o analisador sintático para gerar a AST da função definida em *WMLScript* na FIG. 2.23, então utilizamos o algoritmo de detecção de padrões para encontrar as ocorrências do padrão na estrutura da AST. Neste exemplo, encontramos uma ocorrência do padrão de código, na FIG. 2.24 mostramos a associação do padrão com sua ocorrência na estrutura da AST gerada para o programa.

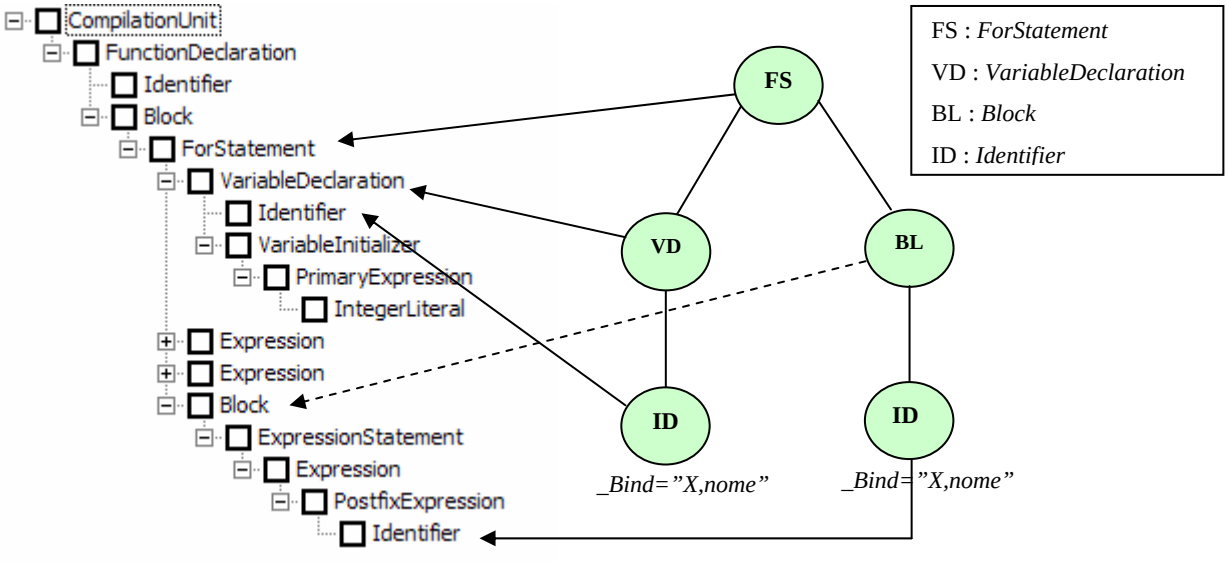

FIG. 2.24 Casamento do padrão na AST do programa *WMLScript*.

2.1.2.6 ATRIBUTO *_DISTINCT*

O atributo *_Distinct* foi uma melhoria que realizamos na PDM. Ao encontrar mais de uma ocorrência de um padrão em uma mesma instrução é possível utilizar o atributo *_Distinct* para inserir uma única marcação. No exemplo abaixo, utilizamos novamente o padrão que busca por ocorrências de mais de uma variável declarada em uma mesma instrução. Neste caso mesmo que haja, por exemplo, nove variáveis declaradas na mesma instrução a ferramenta de detecção de padrões irá inserir somente uma marcação na linha onde as variáveis foram declaradas. Na FIG. 2.25, ilustramos o padrão utilizando o atributo *_Distinct*.

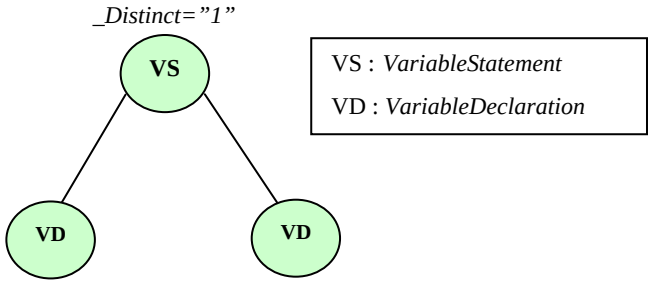

FIG. 2.25 Ilustração do padrão utilizando o atributo *_ Distinct*.

Neste caso foi interessante posicionar o atributo *_Distinct* no nó *VariableStatement* pois ele será o pai das variáveis declaradas na mesma instrução. Na FIG. 2.26 representamos o padrão em PDL, declaramos dois elementos *VariableDeclaration*, neste caso o algoritmo vai encontrar somente uma ocorrência do padrão sempre que houver duas ou mais variáveis declaradas na mesma instrução.

```
<VariableStatement _Distinct="1" _Mark="Evite declarar mais de uma
variável em um mesmo comando.">
    <VariableDeclaration/>
    <VariableDeclaration/>
</VariableStatement>
```

FIG. 2.26 Padrão em PDL utilizando o atributo _ *Distinct*.

Para ficar ainda mais claro, vamos ilustrar a utilização do atributo *_Distinct* em uma função com três variáveis declaradas em uma mesma instrução, neste caso sem utilizar o atributo o algoritmo foi implementado para encontra três ocorrências do padrão, no caso as combinações: (x, y), (x, z), e (y, z). Com o atributo a ferramenta insere somente uma marcação independente do número de variáveis declaradas na mesma instrução.

```
extern function teste() {
  var x, y, z;
}
```

FIG. 2.27 Função com duas variáveis declaradas em uma mesma instrução.

Então a ferramenta de detecção de padrões obtém somente as associações ilustradas na FIG. 2.28 e determinamos a ocorrência de uma única instância do padrão na AST analisada referente à função codificada em *WMLScript* na FIG. 2.27.

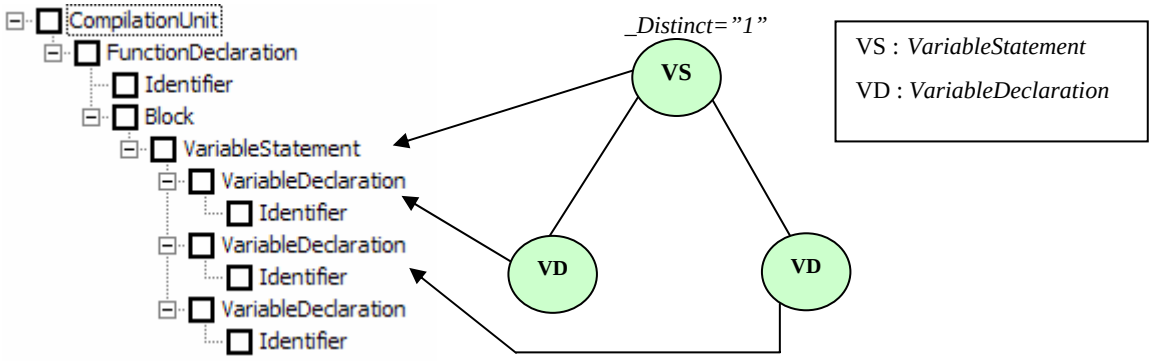

FIG. 2.28 Casamento do padrão na AST do programa *WMLScript*.

2.1.3 ELEMENTO ESPECIAL OR

Além dos atributos especiais, Albuquerque implementou o elemento *OR* que realiza a função do "ou" entre os elementos utilizados na composição de um padrão. Na FIG. 2.29 o padrão busca por ocorrências de *for* "ou" *while* sem bloco de chaves.

```
<FunctionDeclaration>
<OR>
    <WhileStatement _Mark="While sem bloco com chaves">
        <Block _Not="true"/>
    </WhileStatement>
    <ForStatement _Mark="While sem bloco com chaves">
        <Block _Not="true"/>
    </ForStatement>
</OR>
</FunctionDeclaration>
```

FIG. 2.29 Padrão em PDL ilustrando a utilização do elemento *OR*.

2.2 ARQUITETURA DA FERRAMENTA

A ferramenta de detecção de padrões de código foi implementada como um projeto de *plugin* na plataforma Eclipse. Vamos mostrar a interface para detecção de padrões proposta pela PDM no ambiente Eclipse, e então vamos explicar a arquitetura da ferramenta. A interface proposta pela PDM é composta por um editor para o desenvolvimento do código dos programas, e duas *views*, uma para visualizar a árvore sintática dos programas e outra para disponibilizar a biblioteca com os padrões de código implementados em PDL. A PDM ainda utiliza a *Problems View* do Eclipse para mostrar as ocorrências dos padrões encontrados. Na FIG. 2.30 mostramos um instantâneo do ambiente Eclipse configurado com a perspectiva da ferramenta de detecção de padrões.

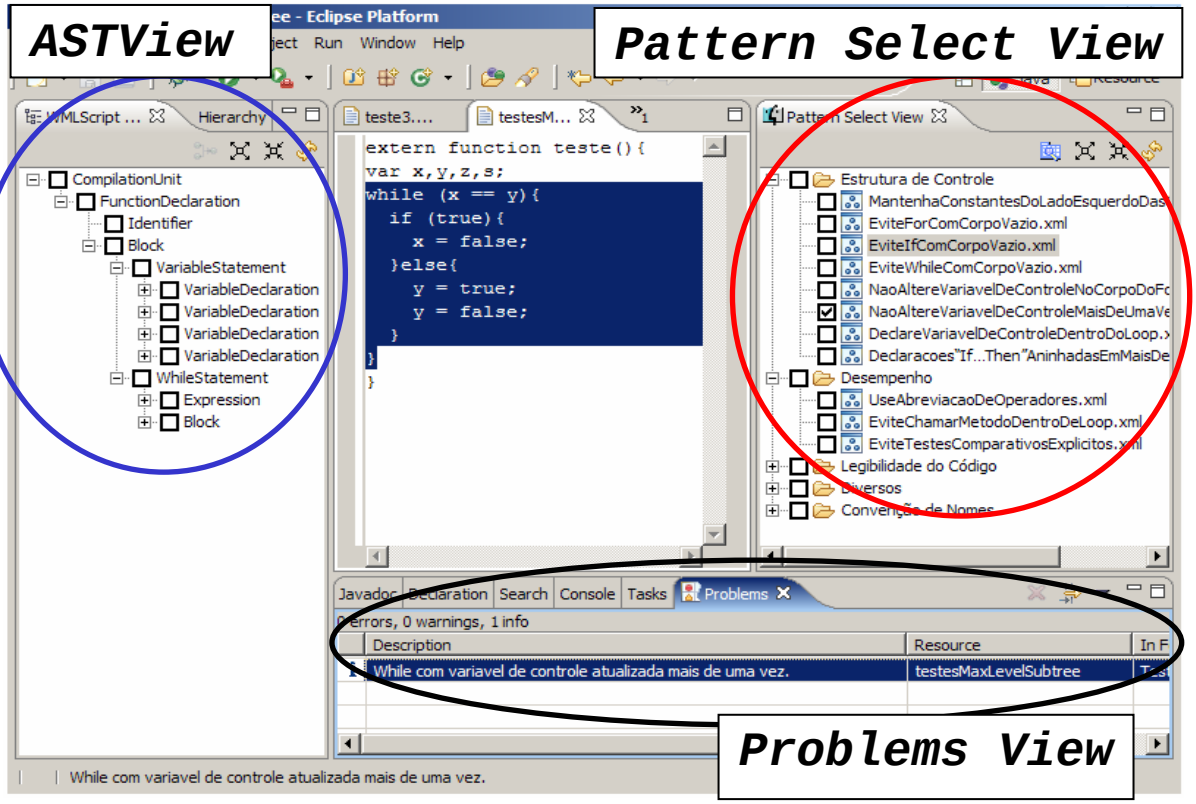

FIG. 2.30 Ferramenta de detecção de padrões integrada na plataforma Eclipse.

A arquitetura técnica demonstra o processo de detecção de padrões implementado pela ferramenta PDM. Adaptamos a ilustração da arquitetura [ALBUQUERQUE, 2005] na FIG. 2.31, para refletir as alterações que foram realizadas para a detecção dos padrões de código no *WMLScript*. O processo de detecção dos padrões é análogo ao implementado por Albuquerque para detecção dos padrões de código Java. Primeiro são definidos os padrões em PDL, então utilizamos a API DOM[1] para ler os padrões e guardar uma representação destes em memória no formato de árvore, em seguida chamamos o analisador sintático para construção da AST dos programas, e finalmente chamamos o algoritmo de detecção de padrões para realizar a busca dos padrões na AST gerando as marcações para os padrões encontrados.

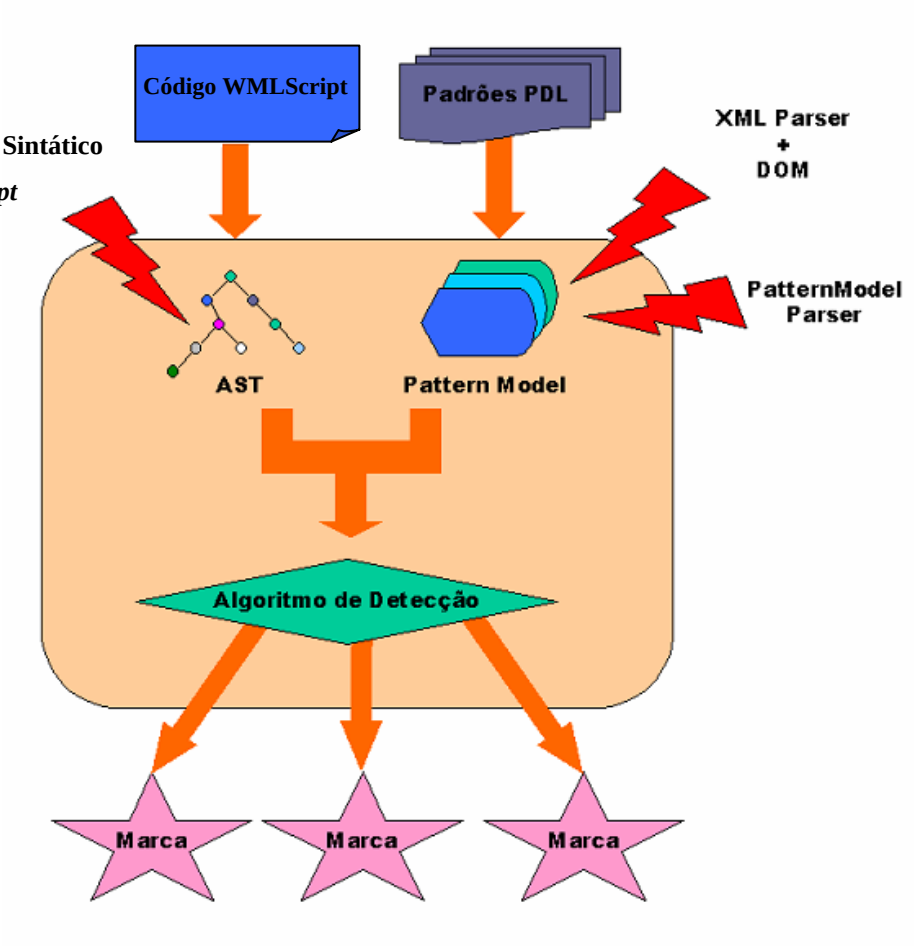

FIG. 2.31 Arquitetura da ferramenta de detecção de padrões [ALBUQUERQUE, 2005]

[1] DOM é a abreviatura de *Document Object Model* que é uma API em Java para processamento de documentos XML.

A fim de migrar a ferramenta PDM para *WMLScript* os seguintes passos foram cumpridos:

- Definição da linguagem PDL, i. e, dos tipos de nós, seus atributos e seus relacionamentos de herança.
- Implementar um analisador sintático para linguagem de programação.
- Adaptar o algoritmo para buscar os padrões na AST.
- Implementar os atributos específicos da linguagem.
- Integração com a plataforma Eclipse .

2.3 ALGORITMO PARA DETECÇÃO DOS PADRÕES DE CÓDIGO

A idéia geral do algoritmo [ALBUQUERQUE, 2005] de detecção de padrões é simplesmente encontrar ocorrências dos padrões de codificação na AST de um programa. O algoritmo recebe como entrada um conjunto de padrões e a AST do programa a ser analisado, então o mesmo percorre a AST em busca de ocorrências dos padrões. O algoritmo pode ser considerado eficiente no sentido que permite encontrar todas as ocorrências de todos os padrões, independentemente do número de padrões selecionados, percorrendo a AST do programa uma única vez.

Para alcançar o objetivo de encontrar todas as ocorrências de cada padrão percorrendo a AST uma única vez, foi utilizado o recurso de clonagem, sempre que ocorrer um casamento o algoritmo gera uma cópia do padrão que será utilizada posteriormente para continuar a busca das outras possíveis ocorrências do mesmo padrão na AST. Este é um ponto crítico do algoritmo em relação ao desempenho, entretanto em análise realizada por Albuquerque, em situações reais, a clonagem não deve implicar em uma degradação significativa no desempenho, pois o tamanho dos padrões é pequeno em relação à AST dos programas.

2.4 CONSIDERAÇÕES FINAIS

Neste capítulo apresentamos a ferramenta de detecção de padrões de código proposta por [ALBUQUERQUE, 2005]. Mostramos como definir novos padrões utilizando a PDL e seus atributos. Ainda apresentamos características da arquitetura e o algoritmo de detecção. Nos próximos capítulos vamos apresentar a ferramenta JavaCC e então como geramos o analisador sintático do *WMLScript*.

3 JAVA COMPILER COMPILER (JAVACC)

O *Java Compiler Compiler* (JavaCC) [JAVACC] é um gerador de analisadores léxicos e sintáticos desenvolvido pela *Sun Microsystems* [SUN] e mantido atualmente pela *java.net* [JAVA.NET], ele gera código Java a partir da especificação de uma gramática. A sintaxe utilizada pelo JavaCC para descrição das gramáticas é semelhante à do Java [JAVA]. No JavaCC podemos utilizar diversos recursos comuns ao Java, como a implementação de classes, métodos e o tratamento de exceções, a ferramenta ainda implementa facilidades como a geração automática de mensagens de erros para análise léxica e sintática.

O analisador sintático gerado com o auxílio do JavaCC é descendente recursivo [AHO, 1995] e aceita gramáticas do tipo LL(1). O JavaCC permite inserir ações sintáticas durante a especificação da gramática para realizar a construção da AST. Neste capítulo vamos mostrar como especificar a gramática de uma linguagem no JavaCC, as opções disponíveis para configuração, as características dos analisadores sintáticos e léxicos gerados, e finalmente uma análise das classes em Java geradas e suas responsabilidades.

3.1 ESPECIFICAÇÃO DA GRAMÁTICA

No JavaCC a especificação léxica e sintática fica em um único arquivo. A configuração das opções para customização do comportamento do *parser* e da ferramenta JavaCC também são realizadas neste mesmo local. A seguir definimos em tópicos a organização do arquivo de descrição e elucidamos cada seção que deve ser implementada antes da geração do *parser*.

- *Options:* utilizada na configuração das preferências para customização do *parser*. Nesta seção definimos o modo de depuração, checagens de ambigüidade, modo para reportar erros, número de *tokens* lidos por vez, etc.
- *Parser_Begin/Parser_End:* unidade de compilação Java tradicional, nela implementamos uma classe com o mesmo nome da classe gerada para o *parser*. Nesta

seção podemos definir métodos que ficarão disponíveis na classe que realiza a análise sintática.

- *Skip:* utilizada para definir as seqüências de caracteres que serão ignoradas pelo analisador léxico. Nesta seção definimos as estruturas de comentários, espaços em branco, tabulações, etc.
- *Tokens:* os *tokens* da gramática, as palavras reservadas e operadores, além das construções permitidas para itens como identificadores e literais.
- *Productions:* esta seção é utilizada para definir a gramática da linguagem.

O JavaCC permite inserir código Java em ações léxicas e sintáticas definidas durante a especificação da gramática. As ações léxicas são executadas sempre que ocorrer um casamento com sucesso de um *token*, já as ações sintáticas estão junto com as produções e são executadas somente quando o analisador sintático reconhecer determinada produção na entrada [DELEMARO, 2004]. A construção da AST pode ser realizada na implementação das ações sintáticas.

3.2 OPÇÕES DE CONFIGURAÇÃO

As opções de configuração são utilizadas para customizar o comportamento do *parser*, e são definidas em uma área no arquivo de especificação rotulada com a palavra *options.* Uma mesma opção não pode ser configurada mais de uma vez. O JavaCC possui um conjunto de opções padrão para o caso de omissão de uma ou mais opções. As configurações também podem ser realizadas pela linha de comando durante a execução do JavaCC, neste caso as opções definidas na linha de comando sobrescrevem as opções do arquivo de especificação. Na TAB. 3.1 listamos algumas das opções mais utilizadas e suas descrições.

TAB. 3.1 Opções de configuração do JavaCC.

Opção	Descrição
LOOKAHEAD	Define o número de *tokens* lidos nos pontos indicados. Auxilia na tomada de decisão durante a análise sintática.
STATIC	Permite definir todos os métodos e variáveis como estáticos no gerente de *tokens* e no *parser*. Isto implica que apenas um objeto *parser* estará presente por vez. Sem esta opção seria possível utilizar o operador "*new*" para construir mais de um *parser*. Estes poderiam ser utilizados simultaneamente em *threads* diferentes.
DEBUG_PARSER	Quando ativada esta opção permite obter informações de *debug* do *parser*. É possível acompanhar os passos ou ações executadas durante a análise sintática.
ERROR_REPORTING	Esta opção permite melhorar de forma pouco significativa o desempenho do *parser*, ela reduz o número de informações detalhadas nos relatórios de erros da análise sintática.
OPTIMIZE_TOKEN_MANAGER	Opção para otimização do analisador léxico.
BUILD_PARSER	Permite definir se deseja ou não gerar o *parser*, esta opção pode ser utilizada para gerar somente o analisador léxico.

Além das opções apresentadas, o JavaCC possui ainda algumas outras detalhadas em sua documentação. Para implementação do *parser* do *WMLScript*, utilizamos as configurações padrão do JavaCC, alterando somente a opção de *debug*, para acompanhar a execução da análise léxica e sintática durante a fase de desenvolvimento.

3.3 ESPECIFICAÇÃO DO ANALISADOR LÉXICO

O JavaCC permite a geração de analisadores léxicos implementados em Java. O analisador léxico gerado pela ferramenta recebe um programa como entrada e reconhece o código fonte como uma seqüência de caracteres, dividindo a seqüência em pequenas partes, identificadas como *tokens*, e então categoriza os *tokens* em quatro possíveis grupos com comportamentos distintos:

- *Skip*: ignora as *strings* casadas (utilizado para definir estruturas de comentários, tabulações, etc.).
- *More*: continua o fluxo normal após o casamento, a *string* reconhecida vai compor o prefixo da *string* subseqüente.
- *Token*: cria um *token* com a *string* casada e envia para o *parser*.
- *Special_Token* : *token* especial que não participa do analisador sintático.

Na especificação do analisador léxico, os *tokens* reconhecidos pela linguagem, devem ser declarados nas categorias apropriadas por meio de expressões regulares. Na FIG 3.1 mostramos algumas declarações de *tokens* nos grupos *Token* e *Skip* retiradas da gramática do *WMLScript*. O conjunto de *tokens* apresentado, será utilizado para ilustrar em um exemplo posterior o funcionamento do analisador léxico gerado com o auxílio do JavaCC.

```
TOKEN : //Os tokens apresentados são ilustrativos.
   {
           < EXTERN: "extern" >
       |   < FUNCTION: "function" >
       |   < VAR: "var">
       |   < IDENTIFIER: <LETTER> (<LETTER>|<DIGIT>)* >
     }

 SKIP :
   {
       | <"//" (~["\n","\r"])* ("\n"|"\r"|"\r\n")>
       | <"/*" (~["*"])* "*" (~["/"] (~["*"])* "*")* "/">
   }
```

FIG. 3.1 Subconjunto de tokens declarados para linguagem *WMLScript.*

Considerando a seguinte função em *WMLScript* com uma variável “x” declarada em seu escopo, o analisador léxico procede segundo os passos apresentados a seguir:

```
extern function principal(){
      var x;
}
```

FIG. 3.2 Função em *WMLScript.*

Primeiro o código fonte é dividido na seguinte seqüência de *tokens*:

"extern", " ", "function", " ", "principal", "(",")","{","\n","\t","var", " ", "x", ";", "\n", "}"

FIG. 3.3 Seqüência de *tokens* referente à função.

Então são identificados os tipos de cada *token*, neste caso o analisador categoriza ordenadamente os *tokens* nos seguintes grupos:

EXTERN, SKIP, FUNCTION, SKIP, IDENTIFIER, PE, PD, CE, SKIP, SKIP, VAR, SKIP, IDENTIFIER, PV, SKIP, CD, EOF

FIG. 3.4 Categoria dos *tokens* no JavaCC.

Toda vez que um final de arquivo é encontrado o analisador cria o *token* "EOF". A seqüência de *tokens* lida à exceção dos identificados como SKIP são passados para o *parser*,

que analisa a seqüência e realiza a análise sintática nos programa, possivelmente retornando uma representação da AST.

3.4 ESPECIFICAÇÃO DO ANALISADOR SINTÁTICO

Antes de mostrar como utilizar o JavaCC para especificar e gerar o analisador sintático, vamos ilustrar por um exemplo a diferença entre os erros sintáticos e os erros de checagens estáticas, para facilitar o entendimento das responsabilidades no analisador sintático. Utilizamos uma função em *WMLScript* para diferenciar o que um compilador interpreta como erro sintático e o que ele considera erro de checagem estática.

```
extern function exemplo1() {
      var x;
      x = "5";
}
```

FIG. 3.5 Função em *WMLScript.*

A função acima, está sintaticamente correta e o compilador do *WMLScript* gera o *bytecode* sem apresentar nenhum erro de compilação. Se substituirmos a variável “x” por “y” no momento de atribuir o valor “5” à variável, temos a seguinte situação:

```
extern function exemplo1() {
      var x;
      y = "5";
}
```

FIG. 3.6 Função em *WMLScript* com erro de checagem estática.

O compilador não irá mais gerar o *bytecode* para a função, apesar da mesma estar sintaticamente correta, e alerta que a variável “y” não foi declarada, neste caso temos um erro de checagem estática. Tipicamente os analisadores sintáticos capturam erros que podem ser descritos por uma gramática em uma linguagem, o tipo de checagem apresentado não faz parte do escopo deste trabalho, podendo ser implementado como uma melhoria em trabalhos futuros.

Enquanto os analisadores sintáticos gerados com as ferramentas tradicionais *Yacc* [YACC] e *Bison* [BISON] realizam uma análise *bottom-up*, o *parser* gerado pelo JavaCC realiza um tipo de análise sintática *top-down* ou descendente. A análise descendente parte do símbolo inicial e busca identificar na entrada as construções que correspondem às produções da gramática. O *parser* gerado pelo JavaCC realiza uma análise descendente recursiva, onde cada símbolo não terminal corresponde a um método recursivo, responsável por verificar se a entrada “casa” ou não com a estrutura apresentada nos métodos.

Os métodos implementados para o analisador sintático possuem uma sintaxe semelhante aos métodos da linguagem de programação Java. A partir destes métodos é possível definir a gramática sintática de uma linguagem com declarações que assemelham-se à notação *Backus-Naur Form* (BNF). Os métodos implementados para definir o analisador têm uma forma semelhante à ilustrada na FIG. 3.7.

```
/* Método ilustrativo sem equivalente na gramática do WMLScript.
 * Equivale à regra definida em notação BNF:
 * FunctionDeclaration ::= extern_opt function Identifier pe pd
 */
FunctionDeclaration functionDeclaration() :
{
    Token pe,pd;
}
    {
        ( <EXTERN> )? <FUNCTION> <IDENTIFIER> pe=<PE> pd=<PD>
        {
            FunctionDeclaration f = new FunctionDeclaration();
            return f;
        }
    }
```

FIG. 3.7 Modelo de função recursiva utilizada nas especificações com o JavaCC.

O método apresentado é definido por uma seqüência iniciada pelo *token* opcional **extern**, seguido pelos *tokens* **function**, **identifier**, **pe** e **pd** respectivamente. O parser gerado simplesmente verifica se a seqüência de *tokens* da entrada está de acordo com as produções definidas. O JavaCC permite acrescentar ações sintáticas na implementação dos métodos para construção da AST, as ações são executadas somente quando o casamento com a produção definida no método ocorrer com sucesso. Na FIG. 3.7 após a seqüência de *tokens* que define a produção, instanciamos um objeto da classe *FunctionDeclaration*. Estas ações podem ser utilizadas para instanciar classes de um modelo definido para construção da AST dos programas.

3.5 GERAÇÃO DO ANALISADOR LÉXICO E DO ANALISADOR SINTÁTICO

Após construir o arquivo de especificação no JavaCC com as definições do analisador léxico e sintático, faz-se necessária uma chamada ao JavaCC onde são geradas sete classes independentes cada uma em um arquivo separado com funções bem definidas na análise léxica e sintática. As classes geradas são:

- *TokenMgrError*: é uma subclasse de *Throwable* no Java, define uma classe simples para tratamento de erros. Ela á utilizada durante a análise léxica.
- *ParseException*: é uma subclasse de *Exception*, consequentemente de *Throwable*. É utilizada pelo analisador sintático.
- *Token*: classe utilizada para representação dos *tokens*. Possui alguns atributos como *image* que representa a seqüência de caracteres que define o *token*.
- *SimpleCharStream*: classe responsável por fornecer os caracteres ao analisador léxico.
- *ExampleConstants*: interface utilizada tanto pelo analisador léxico quanto pelo analisador sintático.
- *ExampleTokenManager*: classe que implementa o analisador léxico.
- *Example*: classe que implementa o analisador sintático.

As classes devem ser compiladas com um compilador Java, e então podem ser utilizadas em qualquer aplicação. Caso a finalidade seja a construção da AST como é o objetivo deste trabalho, o *parser* deve importar um pacote com uma estrutura de dados implementada para construção da AST.

3.6 CONSIDERAÇÕES FINAIS

Neste capítulo detalhamos aspectos importantes da ferramenta JavaCC, utilizada nesta dissertação para geração do analisador léxico e sintático do *WMLScript.* A implementação do *parser* foi fundamental para o funcionamento da ferramenta de detecção de padrões de código. No próximo capítulo vamos mostrar como definimos o modelo de classes para AST, e como procedemos no JavaCC para construir a árvore dos programas *WMLScript* a partir da instanciação das classes definidas no modelo.

4 ANALISADOR SINTÁTICO DO WMLSCRIPT

A fim de desenvolver a ferramenta de detecção de padrões de código para o *WMLScript,* implementamos um analisador léxico e um analisador sintático para linguagem. Utilizamos a ferramenta JavaCC para gerar o analisador sintático a partir da especificação da gramática. A escolha de uma ferramenta em Java possibilitou a integração do analisador sintático com o ambiente Eclipse e com a ferramenta de detecção de padrões de código *Pattern Detect Machine* (PDM).

O analisador sintático gerado pelo JavaCC é descendente recursivo e aceita as gramáticas do tipo LL(1). O gerador oferece facilidades para resolver conflitos na análise sintática utilizando a cláusula *lookahead* que define o número de *tokens* lidos nos pontos indicados. Ao especificar a gramática do *WMLScript* definimos como o analisador sintático deve proceder na construção da AST, o algoritmo de detecção de padrões utiliza a AST dos programas *WMLScript* para realizar a busca dos padrões de código.

Para a construção da AST dos programas *WMLScript,* projetamos um modelo de classes em Java. Definimos classes para os terminais e não terminais da gramática e construímos a AST instanciando objetos para representação dos nós. O mesmo modelo de classes foi utilizado na definição da nomenclatura da linguagem de descrição de padrões (PDL). Os elementos e atributos da linguagem foram definidos com os mesmos nomes das classes e atributos do modelo. Dessa forma a hierarquia de classes pôde ser usada na PDL para se casar padrões com diversos não terminais, que pertencem à mesma classe.

Neste capítulo vamos mostrar como procedemos na implementação do analisador léxico e sintático do *WMLScript* utilizando o JavaCC. Propomos ainda um modelo de classes em Java para representar a AST dos programas *WMLScript,* elucidamos a competência do modelo de classes para representação da AST e mostramos como algumas propriedades da orientação a objetos (OO) puderam auxiliar no processo de detecção dos padrões de código para o *WMLScript*.

4.1 MODELO DE CLASSES PARA ÁRVORE SINTÁTICA

Implementamos um modelo de classes em Java para construção da AST dos programas *WMLScript*, a estrutura de classes é uma alternativa para representação da AST. Primeiramente desenvolvemos classes para os símbolos terminais e não terminais da gramática, então definimos uma relação de herança entre as classes implementadas. A herança permite referenciar univocamente um conjunto de não terminais da gramática que apresentam comportamentos ou características em comum, ela facilitou o processo de escrita dos padrões de código. Na próxima seção vamos exemplificar a utilização da herança para definição dos padrões. No modelo de classes definimos uma classe *Node* ancestral de todas as outras classes do modelo, além de ser utilizada para definição de padrões em PDL, a classe *Node* facilitou a integração com a ferramenta de detecção de padrões PDM. A *Node* implementa métodos para visitar a AST dos programas e guarda informações para localização dos padrões no código fonte dos programas.

4.1.1 IMPLEMENTAÇÃO DO MODELO DE CLASSES

No modelo de classes do *WMLScript* definimos uma classe *Node* como superclasse abstrata de todas as outras classes. Em *Node* implementamos os métodos utilizados pelo padrão de projeto *visitor* [GAMMA, 1994] para visitar os nós da AST, durante a visita realizamos a busca dos padrões de código, sendo este um ponto de integração com a PDM. Em *Node* definimos os atributos responsáveis pela localização de cada *token*, esta informação de natureza léxica foi utilizada para determinar onde inserir as marcações dos padrões encontrados nos programas analisados. Na FIG 4.1 destacamos a classe *Node* em uma versão resumida com a assinatura dos seus principais métodos e atributos.

```
public abstract class Node
{

  //referência para o pai do nó na AST
  public Node parent;

    //atributos responsáveis por marcar a posição do nó no código fonte do
    //programa analisado
    private int beginLine;
    private int beginColumn;
    private int endLine;
    private int endColumn;

    //métodos utilizados por cada nó durante a visita da AST
     final void acceptChildren(Visitor visitor, Collection children);
     abstract void accept0(Visitor visitor);
     final void acceptChild(Visitor visitor, Node child);
}
```

FIG. 4.1 Classe *Node*

Implementamos as classes abstratas descritas a seguir para generalizar as classes que apresentavam algum comportamento ou responsabilidade em comum, as generalizações propostas tiveram importância para facilitar a definição dos padrões de código.

- *AbstractLiteral:* utilizada para representar univocamente o conjunto dos literais inteiros, ponto flutuante, *string* e booleano.
- *AbstractCommandStatement:* define o conjunto de classes utilizadas para representação das estruturas de fluxo de controle do *WMLScript.*
- *AbstractExpression:*define um conjunto eclético denominado expressões da linguagem *WMLScript*, neste grupo definimos expressões como chamada de funções, expressões aditivas, multiplicativas, pós-fixas, unárias entre outras.
- *AbstractStatement:* Esta classe generaliza *AbstractCommandStatement* e ainda referencia as classes *VariableStatement* e *VariableDeclaration*. Separamos as estruturas de fluxo de controle em *AbstractCommandStatement*, da declaração de variável para suprir uma necessidade de referenciar separadamente as estruturas do fluxo de controle da declaração de variáveis em determinados padrões de código.

As generalizações foram úteis na definição de alguns padrões de codificação, por exemplo, no capítulo 6 desta dissertação o padrão 5.3 diz:

"Funções vazias devem ser removidas. Exclua as funções com corpo vazio. Primeiramente remova as referências à função e depois remova a função."

Ao definir este padrão em PDL como mostrado na FIG. 4.2, o nó *Node* aparece dentro de um Bloco (*Block*) e marcado com o atributo *_Not,* esta situação significa que não é permitida a ocorrência de qualquer um dos nós que estendem *Node* dentro do bloco (*Block*) de chaves de uma função (*FunctionDeclaration*), caracterizando a ocorrência do padrão. A facilidade para referenciar um conjunto de nós através de um só nome pode simplificar consideravelmente a definição de alguns padrões de código.

```
<FunctionDeclaration _Mark="Função com Corpo Vazio.">
    <Block _MaxSubTreeLevel="1">
        <Node _Not="true"/>
    </Block>
</FunctionDeclaration>
```

FIG. 4.2 Padrão definido em PDL, busca por funções vazias.

Na FIG. 4.3 delineamos o modelo implementado para dar suporte à construção da AST, na representação ilustramos a superclasse *Node* e as classes abstratas *AbstractStatement, AbstractExpression, AbstractCommandStatement* e *AbstractLiteral*, definimos a relação de herança entre as classes do modelo. As folhas do diagrama correspondem aos terminais e não terminais. O diagrama não possui informação sintática, para tanto vamos disponibilizar nos anexos uma versão completa do diagrama de classes.

O diagrama apresentado permite ilustrar as classes que foram utilizadas para representação da AST dos programas, no exemplo da seção subseqüente mostramos como representar a AST dos programas desenvolvidos em *WMLScript* utilizando as classes implementadas no modelo.

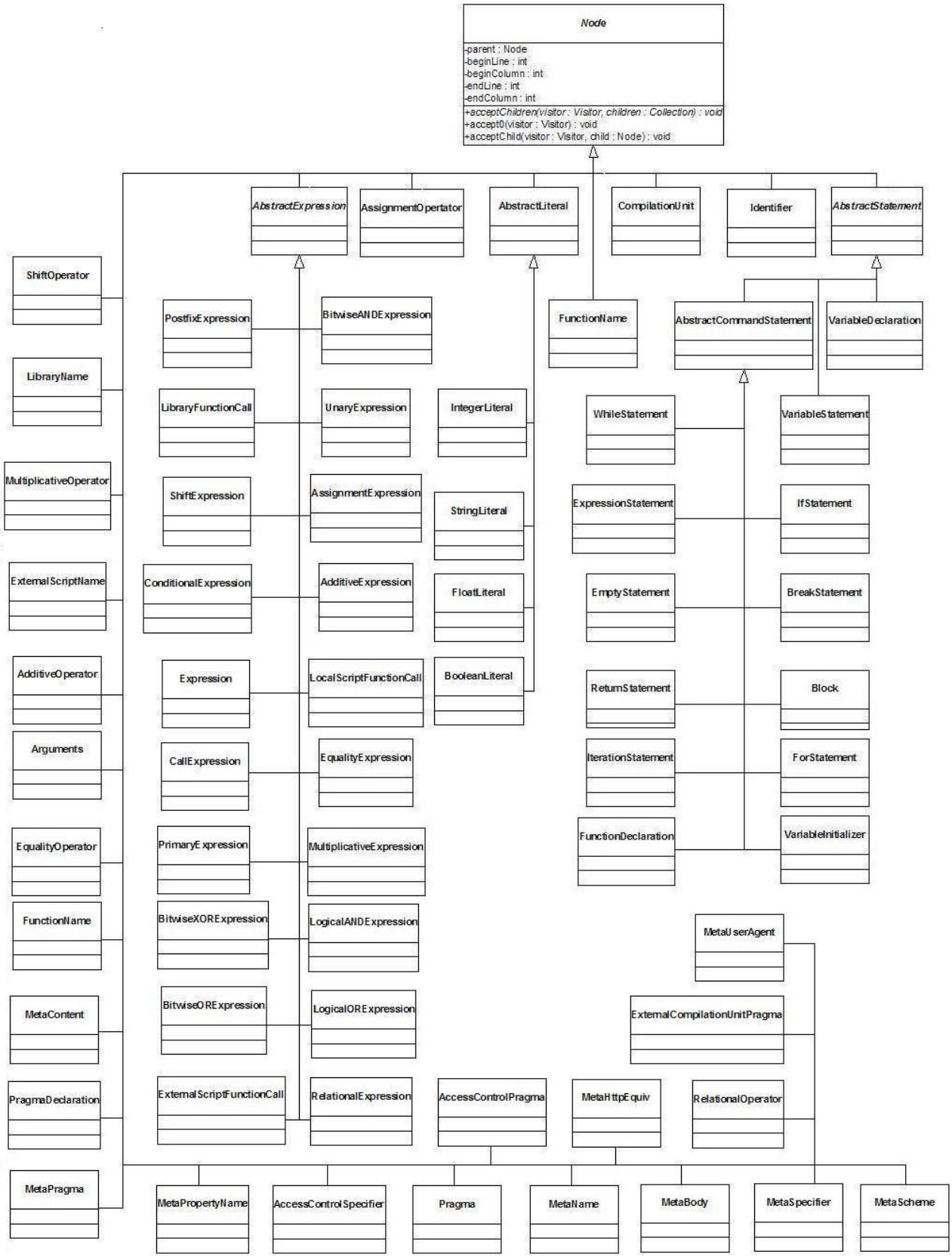

FIG. 4.3 Modelo de classes para AST do *WMLScript*.

4.1.2 CONSTRUÇÃO DA AST DOS PROGRAMAS *WMLSCRIPT*

Nesta seção procuramos mostrar a relação existente entre a gramática do *WMLScript*, o modelo de classes e a construção da AST. Na FIG. 4.4 mostramos em notação *Backus Naur Form* (BNF) o trecho da gramática do *WMLScript* relacionado à declaração de variáveis.

```
1. <variablestatement> ::=
      “var” <variabledeclarationlist>

2. <variabledeclarationlist> ::=
      <variabledeclaration> |
      <variabledeclarationlist> <variabledeclaration>
3. <variabledeclaration> ::=
      “identifier” [<variableinitializer>]

4. <variableinitializer> ::=
       “=” <conditionalexpression>
```

FIG. 4.4 Produções em BNF extraídas da gramática do *WMLScript*

Vamos montar uma árvore sintática para o código fonte da FIG. 4.5 que possui uma função com uma variável declarada e inicializada no seu escopo. As regras formalizadas nas produções da FIG. 4.4 possuem uma correspondência com os nós instanciados na AST construída para função da FIG. 4.5, observe nas regras que o símbolo *VariableDeclaration* produz *Identifier* e *VariableInitializer*, na AST ilustrada na FIG. 4.6 *Identifier* e *VariableInitializer* são filhos diretos de *VariableDeclaration.*

```
function exemplo() {
  var x = 3;
}
```

FIG. 4.5 Função em *WMLScript* com declaração de variável.

Utilizamos uma abordagem *top-down* característica do *parser* gerado nesta dissertação para ilustrar a construção da AST. O JavaCC inicia a construção da árvore pelo nó

CompilationUnit definido como nó raiz, e procede com a construção no sentido da raiz para as folhas. Após instanciar o nó raiz o *parser* segue a leitura dos *tokens* instanciando as classes apropriadas sempre que ocorrer um casamento com alguma das produções definidas na gramática da linguagem. Na AST da FIG. 4.6 vamos considerar a parte destacada da árvore onde instanciamos as classes *VariableStatement*, *VariableDeclaration*, *VariableInitializer* e *Identifier*, para ilustrar o processo de construção da AST, considerando que os nós destacados possuem símbolos correspondentes nas produções retiradas da gramática do *WMLScript* (FIG. 4.4).

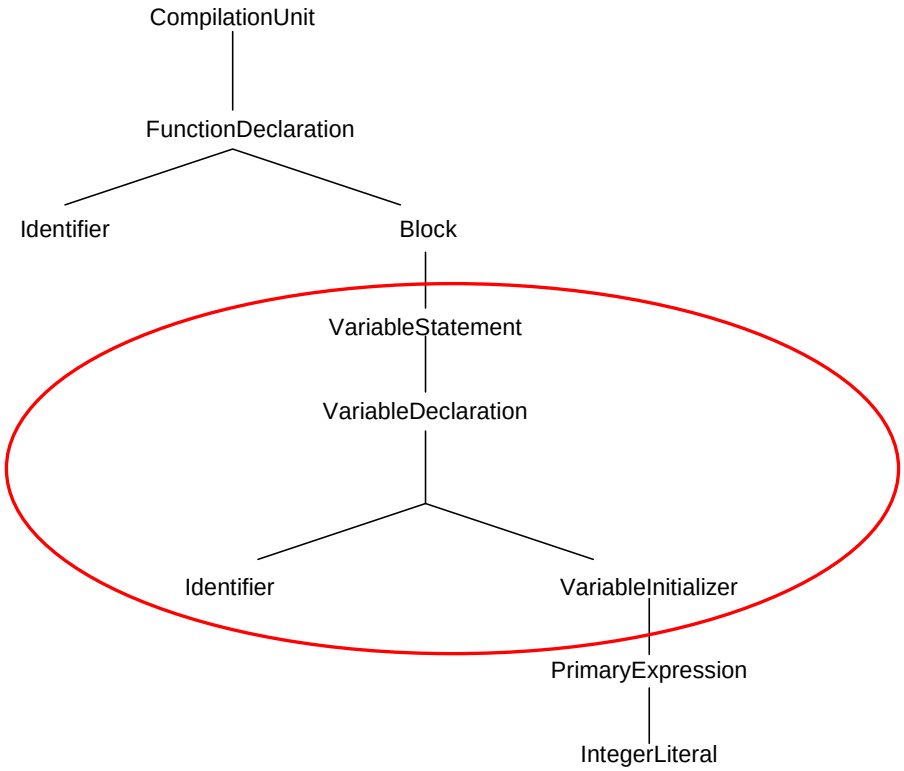

FIG. 4.6 AST que representa a função da FIG. 4.5.

Neste exemplo o nó objeto da classe *VariableStatement* foi instanciado recebendo como argumento uma referência para o filho *VariableDeclaration*, que por sua vez referencia *Identifier* e *VariableInitializer*. Na FIG. 4.7 mostramos o método correspondente ao símbolo

não terminal *VariableDeclaration*, implementado no JavaCC. Este é o código que efetivamente cria a AST, relacionando a sintaxe concreta com a sintaxe abstrata.

```
/* Produção em BNF equivalente ao método implementado:
 * VariableDeclaration ::=
 *     Identifier (VariableInitializer)
 */
VariableDeclaration variabledeclaration() :
{

   //declaração de variáveis.
   Identifier i = null;
   VariableInitializer variableinitializer = null;

}
   {
       //produção em notação equivalente à BNF.
       i = identifier() ( variableinitializer = variableinitializer() )?
       {
           //instanciando classe para representação da AST.
           VariableDeclaration    v    =    new    VariableDeclaration(i,
           variableinitializer);
           return v;
      }
   }
```

FIG. 4.7 Método para o não terminal *VariableDeclaration*

O analisador sintático gerado pelo JavaCC possui um método recursivo para cada não terminal, cada método retorna um objeto da classe correspondente no modelo. Na FIG. 4.7 o método retorna um objeto da classe *VariableDeclaration*, as variáveis "i" e "variableinitializer" recebem objetos das classes *Identifier* e *VariableInitializer* a partir da chamada recursiva de seus métodos. Observe que a chamada ao construtor do não terminal *VariableDeclaration* irá instanciar um objeto dessa classe passando como argumento os filhos *Identifier* e *VariableInitializer* respectivamente assim como ilustramos na árvore da FIG. 4.6. Na próxima seção vamos mostrar sucintamente como implementamos o arquivo de descrição da gramática com as definições do analisador léxico e do analisador sintático e como procedemos na construção da AST dos programas.

4.2 IMPLEMENTAÇÃO DO ANALISADOR SINTÁTICO NO JAVACC

O *parser* gerado pelo JavaCC realiza análise sintática descendente recursiva, neste tipo de abordagem cada não terminal é implementado por um método recursivo que reproduz as produções da gramática da linguagem. A sintaxe utilizada para descrição da gramática é semelhante à sintaxe da linguagem de programação Java, é permitido ao programador utilizar recursos da linguagem, como por exemplo, a definição de comentários, *try-catch-finaly*, declaração de variáveis, utilização de classes e objetos. O JavaCC ainda disponibiliza algumas APIs [JAVACC] para facilitar a implementação do analisador sintático como por exemplo as rotinas para manipulação dos *tokens*, estas APIs podem ser acessadas diretamente durante a especificação da gramática.

A seguir vamos mostrar como foi implementado o *parser* do *WMLScript* no JavaCC. Utilizamos a estrutura do arquivo de descrição da gramática apresentado no capítulo 3 para mostrar as etapas do desenvolvimento do analisador sintático.

4.2.1 ESPECIFICAÇÃO DA GRAMÁTICA DO *WMLSCRIPT* NO JAVACC

Nesta seção vamos mostrar como especificamos a gramática do *WMLScript* no JavaCC, primeiramente vamos descrever as especificações léxicas da linguagem, e então vamos descrever os métodos implementados para realizar a análise sintática. Cada método do analisador sintático está vinculado a uma produção da gramática da linguagem, sempre que apresentarmos um método vamos detalhar a produção equivalente em BNF.

Antes de especificar o analisador léxico e sintático, o JavaCC permite definir métodos Java a serem implementados pela classe do *parser* a ser gerada. Na FIG. 4.8 mostramos na seção definida entre as estruturas *Parser_Begin/Parser_End*, uma unidade de compilação Java onde escrevemos o método que foi utilizado pela ferramenta de detecção de padrões para realizar a chamada ao *parser* e fazer a análise sintática em um programa passado como argumento, este método retorna um objeto do tipo *CompilationUnit* que é o tipo do nó raiz da árvore sintática gerada.

```
PARSER_BEGIN(AstFactory)

package br.eb.ime.de9.wsdt.dom.parser;

class AstFactory {

   public static CompilationUnit buildAst(java.io.InputStream program)
   {

     CompilationUnit cu = null;
     ASTFactory t = new ASTFactory(program);

     try {
           cu = t.compilationunit();

     } catch (Exception e) {

          System.out.println(e.getMessage());
          e.printStackTrace();

     }
        return cu;
    }
 }
 PARSER_END(AstFactory)
```

FIG 4.8 Método que retorna a AST dos programas *WMLScript*.

Vamos iniciar pela especificação do analisador léxico, primeiramente definimos a seção estereotipada *Skip*, nela declaramos por meio de expressões regulares as construções que serão ignoradas pelo analisador léxico, definimos os espaços em branco, tabulações, quebras de linha e as possíveis estruturas de comentários.

```
SKIP :
{
    " "
  | "\t"
  | "\n"
  | "\r"
  | <"//" (~["\n","\r"])* ("\n"|"\r"|"\r\n")>
  | <"/*" (~["*"])* "*" (~["/"] (~["*"])* "*")* "/">
}
```

FIG 4.9 Definição das expressões que serão ignoradas pelo analisador léxico.

Na FIG. 4.10 definimos os símbolos terminais da linguagem, apresentamos o conjunto de palavras reservadas, literais, identificadores e operadores.

```
/* PALAVRAS RESERVADAS */
TOKEN :
    {
            < ACCESS: "access" >
       |    < AGENT: "agent" >
        |   < BREAK: "break">
        |   < CONTINUE: "continue">
        |   < DOMAIN: "domain">
        |   < ELSE: "else">
        |   < EQUIV: "equiv">
        |   < EXTERN: "extern">
        |   < FOR: "for">
        |   < FUNCTION: "function">
        |   < HEADER: "header">
        |   < HTTP: "http">
        |   < IF: "if">
        |   < ISVALID: "isvalid">
        |   < META: "meta">
        |   < NAME: "name">
        |   < PATH: "path">
        |   < RETURN: "return">
        |   < TYPEOF: "typeof">
        |   < USE: "use">
        |   < USER: "user">
        |   < VAR: "var">
        |   < WHILE: "while">
        |   < URL: "url">
    }

/* LITERAIS */
TOKEN :
    {
            < BOOLEANLITERAL: "true" | "false" >
         |  < NUMERICLITERAL:  //decimais, octais, hexadecimal
            (
                (["0"-"9"] (["0"-"9"])*)
              | (["0"-"7"] (["0"-"7"])* ["o","0"])
              | (["0"-"9"] (["0"-"7","A"-"F","a"-"f"])* ["h","H"])
              | ( ["+","-"] (["0"-"9"] (["0"-"9"])*) )
            )
          >
        |  < FLOATLITERAL:    ((["0"-"9"])* "." (["0"-"9"])*)>
        |  < STRINGLITERAL:   (
                                    ("\""(~["\"","\n","\r"])* "\"")
                                 |  ("\'"(~["\'","\n","\r"])* "\'")
                               )
            >

    }
```

```
/* IDENTIFICADORES */
TOKEN :
    {
            < IDENTIFIER: <LETTER> (<LETTER>|<DIGIT>)* >
          |  < #LETTER: ["_","a"-"z","A"-"Z"] >
          |  < #DIGIT: ["0"-"9"] >
    }

/* OPERADORES */
TOKEN :
       {
           < ATR: "=" >
       |   < MAIOR: ">" >
       |   < MENOR: "<" >
       |   < IGUAL: "==" >
       |   < MENORIGUAL: "<=" >
       |   < MAIORIGUAL: ">=" >
       |   < DIFERENTE: "!=" >
       |   < VIRGULA: "," >
       |   < NEGLOG: "!" >
       |   < NEGBIT: "~" >
       |   < COND: "?" >
       |   < DOISPTS: ":" >
       |   < PONTO: "." >
       |   < E: "&&" >
       |   < OU: "||" >
       |   < INC: "++" >
       |   < DEC: "--" >
       |   < SOM: "+" >
       |   < SUB: "-" >
       |   < MUL: "*" >
       |   < DIV: "/" >
       |   < INTDIV: "div">
       |   < ES: "&" >
       |   < OUS: "|" >
       |   < XOU: "^" >
       |   < RES: "%" >
       |   < MOVESQ: "<<" >
       |   < MOVDIR: ">>" >
       |   < SOMATR: "+=" >
       |   < SUBATR: "-=" >
       |   < MULATR: "*=" >
       |   < DIVATR: "/=" >
       |   < MOVESQATR: "<<=" >
       |   < MOVDIRATR: ">>=" >
       |   < XMOVDIRATR: ">>>=" >
       |   < INTDIVATR: "div=" >
       |   < PE: "(" >
       |   < PD: ")" >
       |   < CE: "{" >
       |   < CD: "}" >
       |   < PV: ";" >
       |   < TRALHA: "#" >
    }
```

FIG. 4.10 Conjunto dos *tokens* aceitos pela linguagem *WMLScript*.

Finalmente definimos a especificação sintática do *WMLScript*, nesta etapa descrevemos os métodos que serão gerados pelo JavaCC para o analisador sintático. Na FIG. 4.11 separamos os métodos definidos para os não terminais *LibraryFunctionCall*, *LibraryName*, *FunctionName* e *Arguments*, além do método definido para guardar informações dos identificadores (*Identifier*). Observe que os métodos retornam objetos das classes de mesmo nome. A especificação completa do analisador sintático encontra-se no apêndice (ver APÊNDICE 1) desta dissertação.

```
/**********************************************************************
*PRIMEIRAMENTE DEFINIMOS A PRODUÇÃO EM BNF DE CADA MÉTODO E ENTÃO
*IMPLEMENTAMOS O MÉTODO CORRESPONDENTE NO JAVACC. APRESENTAMOS SOMENTE
*AS INSTRUÇÕES NOS MÉTODOS QUE CORRESPONDEM À DEFINIÇÃO DAS PRODUÇÕES
*DA GRAMÁTICA E QUE DIZEM RESPEITO À CONSTRUÇÃO DA AST DOS PROGRAMAS
*WMLSCRIPT. O DOCUMENTO COMPLETO COM A ESPECIFICAÇÃO DE TODOS OS MÉTODOS
*DO WMLSCRIPT ENCONTRA-SE NO ANEXO DESTA DISSERTAÇÃO.
**********************************************************************
*    LibraryFunctionCall ::=
*         LibraryName pt FunctionName Arguments
*
*/
LibraryFunctionCall libraryfunctioncall() :
    {
        LibraryName libraryname = null;
        FunctionName functionname = null;
        Arguments arguments = null;
    }
        {
                    libraryname = libraryname() <PONTO> functionname =
                    functionname() arguments = arguments()
            {
                    LibraryFunctionCall f = new LibraryFunctionCall(
                    libraryname, functionname, arguments);
                    return f;
            }
        }


/*   LibraryName ::=
 *         Identifier
 */
LibraryName libraryname() :
    {
        Identifier identifier = null;
    }
        {
            identifier = identifier() //Chamada da função
            {
                return new LibraryName(identifier);
            }
        }
```

```
/*
 *     FunctionName ::=
 *          Identifier
 */
FunctionName functionname() :
    {
        Identifier identifier = null;
    }
        {
            identifier = identifier()
            {
                FunctionName f = new FunctionName(identifier);
                return f;
            }
        }


/*
 *     Arguments ::=
 *             ( )
 *           |  ( ArgumentList )
 */
Arguments arguments() :
    {
        Collection argumentlist = null;
        Token t;
        Token t1;
    }
        {

                t = <PE> ( argumentlist = arglist() )?  t1 = <PD>
                {
                    Arguments a = new Arguments(argumentlist);
                    return a;
                }
        }


/*
 * ESTE MÉTODO INSTANCIA UM OBJETO PARA CADA IDENTIFICADOR RECONHECIDO
 * PELO ANALISADOR LÉXICO E ARMAZENA A STRING DO TOKEN EM UM ATRIBUTO
 * DO OBJETO INSTANCIADO.
 */
Identifier identifier() :
    {
        Token i;
    }
        {
            i = <IDENTIFIER> { return new Identifier(i.image); }
        }
```

FIG 4.11 Especificação sintática do *WMLScript* no JavaCC.

A seguir na FIG. 4.12 detalhamos as classes *LibraryFunctionCall*, *FunctionName*, *Arguments*, *LibraryName* e *Identifier*. Estas classes foram desenvolvidas como *javabeans*[2], cada uma implementada em arquivo Java distinto. Instanciamos objetos a partir das classes apresentadas para representação dos nós da AST dos programas desenvolvidos em *WMLScript*. Estas classes já apresentam as informações sintáticas não presentes no modelo da FIG. 4.3.

```
public class LibraryFunctionCall extends AbstractExpression {

    private LibraryName libraryname;
    private FunctionName functionname;
    private Arguments arguments;

    public LibraryFunctionCall(LibraryName libraryname,
            FunctionName functionname, Arguments arguments) {
        super();
        this.libraryname = libraryname;
        this.functionname = functionname;
        this.arguments = arguments;
    }

    public Arguments getArguments() {
        return arguments;
    }

    public void setArguments(Arguments arguments) {
        this.arguments = arguments;
    }

    public FunctionName getFunctionname() {
        return functionname;
    }

    public void setFunctionname(FunctionName functionname) {
        this.functionname = functionname;
    }

    public LibraryName getLibraryname() {
        return libraryname;
        }

        public void setLibraryname(LibraryName libraryname) {
            this.libraryname = libraryname;
        }
    }
```

[2] Um JavaBean é apenas uma classe Java que segue um conjunto de convenções simples, como por exemplo a implementação de um construtor e dos métodos *get* e *set* para seus atributos privados.

```
public class LibraryName extends Node {

    private Identifier identifier;

    public LibraryName(Identifier identifier) {
        super();
        this.identifier = identifier;
    }

    public Identifier getIdentifier() {
        return identifier;
    }

    public void setIdentifier(Identifier identifier) {
        this.identifier = identifier;
    }
}


public class FunctionName extends Node {

    private Identifier identifier;

    public FunctionName(Identifier identifier,int bl,int bc,int el,int ec) {
        super();
        this.identifier = identifier;
    }

    public Identifier getIdentifier() {
        return identifier;
    }

    public void setIdentifier(Identifier identifier) {
        this.identifier = identifier;
    }
}

public class Arguments extends Node {

    private Collection argumentlist;

    public Arguments(Collection argumentlist,int bl,int bc,int el,int ec) {
        super();
        this.argumentlist = argumentlist;
    }

    public Collection getArgumentlist() {
        return argumentlist;
    }

    public void setArgumentlist(Collection argumentlist) {
        this.argumentlist = argumentlist;
    }
}
```

```
public class Identifier extends Node {

    private String nome;

    public Identifier(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

}
```

FIG 4.12 Classes utilizadas para representação da AST do *WMLScript*.

4.3 CONSIDERAÇÕES FINAIS

Neste capítulo explicamos o processo de construção do analisador léxico e do analisador sintático para o *WMLScript* no JavaCC. No próximo capítulo vamos detalhar a ferramenta de detecção de padrões de código implementada para o *WMLScript* no ambiente Eclipse, apresentamos os *plugins WMLScript Development Tooling* e *WMLScript Pattern Detector* que juntos realizam a análise sintática e a detecção dos padrões de código.

5 INTEGRAÇÃO COM O ECLIPSE

Este capítulo detalha os principais componentes da ferramenta de detecção de padrões do *WMLScript,* a saber, o *plugin WMLScript Development Tooling* (WDT) e o *plugin WMLScript Pattern Detector* (WPD), enfatizando a integração destes com a plataforma Eclipse.

Primeiramente apresentamos o *plugin* WDT responsável por realizar a análise sintática nos programas desenvolvidos em *WMLScript.* Este *plugin* foi projetado para desempenhar um papel análogo ao do *plugin* JDT [JDT] distribuído com a plataforma Eclipse e que foi utilizado em [ALBUQUERQUE, 2005] na PDM para realizar as operações que envolvem a AST dos programas desenvolvidos em Java.

Em seguida desenvolvemos o *plugin* WPD que efetivamente analisa os programas desenvolvidos em *WMLScript* e realiza a busca dos padrões de código, este *plugin* foi implementado na mesma arquitetura da PDM, reutilizamos o algoritmo de detecção de padrões proposto nesta ferramenta, entretanto foi preciso ajustar diversos pontos de integração para adequar o funcionamento às particularidades da linguagem *WMLScript*.

O ambiente escolhido Eclipse é de origem Java, desta forma um dos esforços foi a adequação do ambiente para trabalhar com a linguagem *WMLScript.*

5.1 *PLUGIN WMLSCRIPT DEVELOPMENT TOOLING* (WDT)

Antes de realizar a detecção dos padrões de código no *WMLScript* foi preciso implementar um analisador sintático para linguagem. Assim como descrito no capítulo 4, utilizamos o JavaCC para gerar o analisador sintático do *WMLScript* em Java e para construir a AST dos programas. Criamos no *plugin* WDT o pacote *br.eb.ime.de9.wsdt.dom.parser* que contém o analisador sintático gerado pelo JavaCC.

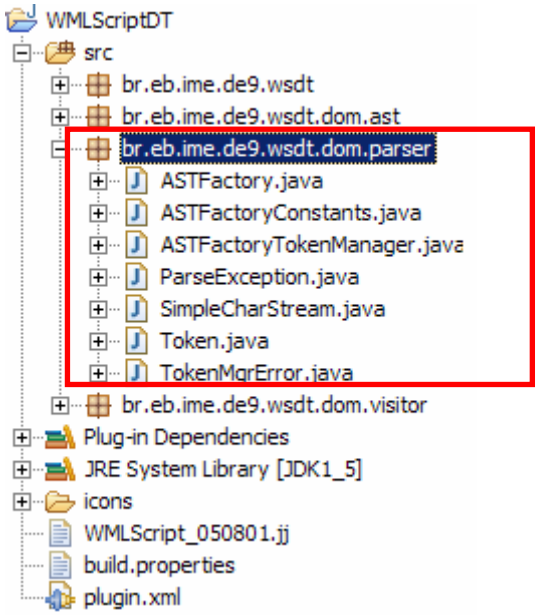

FIG. 5.1 Pacote com as classes geradas para o analisador sintático.

As classes geradas pelo JavaCC possuem as seguintes responsabilidades na análise sintática dos programas *WMLScript*:

- *TokenMgrError*: é uma subclasse de *Throwable* no Java, define uma classe simples para tratamento de erros. Ela á utilizada durante a análise léxica.
- *ParseException*: é uma subclasse de *Exception*, consequentemente de *Throwable*. É utilizada pelo analisador sintático.
- *Token*: classe utilizada para representação dos *tokens*. Possui alguns atributos como *image* que representa a seqüência de caracteres que define o *token*.
- *SimpleCharStream*: classe responsável por fornecer os caracteres ao analisador léxico.
- *ASTFactoryConstants*: interface utilizada tanto pelo analisador léxico quanto pelo analisador sintático.
- *ASTFactoryTokenManager*: classe que implementa o analisador léxico.
- *ASTFactory*: classe que implementa o analisador sintático.

A classe *ASTFactory* é quem disponibiliza o método responsável por iniciar a análise sintática de um programa *WMLScript.* Na chamada do método o programa *WMLScript* deve

ser passado como argumento. A seguir detalhamos o código gerado pelo JavaCC, o método retorna um objeto do tipo *CompilationUnit* que é o tipo do nó raiz da AST gerada.

```
public static CompilationUnit buildAst(java.io.InputStream program)
{
      CompilationUnit cu = null;
      ASTFactory t = new ASTFactory(program);

      try {
          cu = t.compilationunit();
          GenericVisitor pv = new GenericVisitor();
          cu.accept(pv);

      } catch (Exception e) {

              System.out.println("Oops.");
              System.out.println(e.getMessage());
              e.printStackTrace();
      }
      return cu;

}
```

FIG. 5.2 Método responsável por iniciar a análise sintática nos programas *WMLScript.*

Ao explicar, no capítulo 4, como procedemos na construção da AST dos programas *WMLScript* utilizando o JavaCC, detalhamos o modelo de classes que foi implementado para representação da AST dos programas, dispomos as classes no pacote *br.eb.ime.de9.wsdt.dom.ast* do *plugin WMLScriptDT* para serem utilizadas pelo analisador sintático do *WMLScript*.

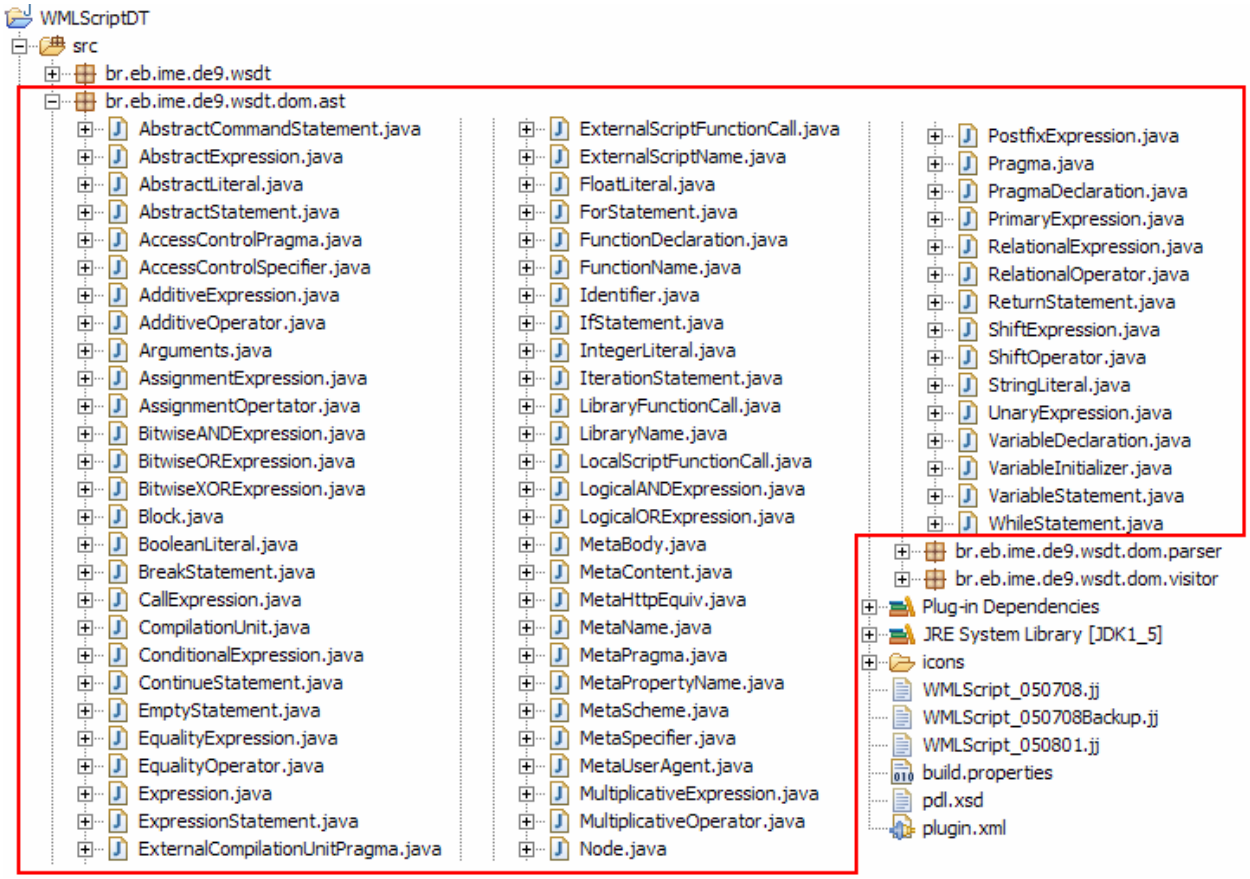

FIG. 5.3 Classes implementadas para AST dos programas *WMLScript*.

O *plugin* WDT além de prover os recursos para análise sintática, disponibiliza um pacote com as classes que implementam o padrão de projeto *visitor*, para realizar a visita aos nós da AST dos programas, durante a visita dos nós é que executamos o algoritmo de detecção de padrões, sendo este um ponto de integração com a ferramenta PDM.

Na FIG. 5.4 ilustramos o pacote *br.eb.ime.de9.wsdt.dom.visitor* com as classes que implementam o padrão de projeto e que devem ser estendidas para realizar a detecção dos padrões de código.

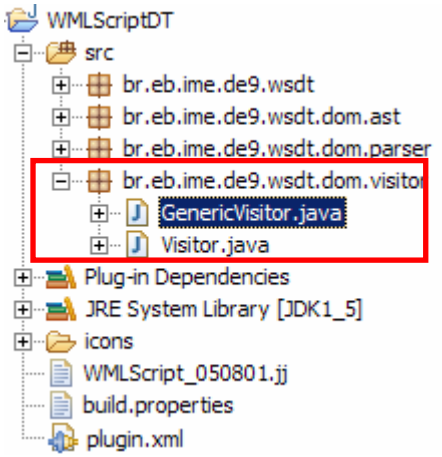

FIG. 5.4 Classes que implementam o padrão de projeto *visitor*.

5.2 *PLUGIN WMLSCRIPT PATTERN DETECTOR* (WPD)

O *plugin WMLScript Pattern Detector* (WPD) é responsável por realizar a detecção dos padrões de código no *WMLScript.* Utilizamos neste *plugin* a mesma arquitetura da ferramenta de detecção de padrões PDM. Primeiramente trabalhamos para adequar o ambiente de desenvolvimento do Eclipse para atender às necessidades do *WMLScript*, configuramos o *workbench*[3] com um editor de texto ao invés de utilizar o editor padrão para o Java, implementamos as interfaces para visualização da AST dos programas *WMLScript*, para seleção dos padrões de código a serem buscados nos programas e utilizamos a *problems view* para listar as ocorrências dos padrões encontrados. Na FIG. 5.5 mostramos um instantâneo da ferramenta em execução.

[3] A *workbench* do Eclipse define os pontos de extensão do *framework* para implementar componentes de interface com o usuário.

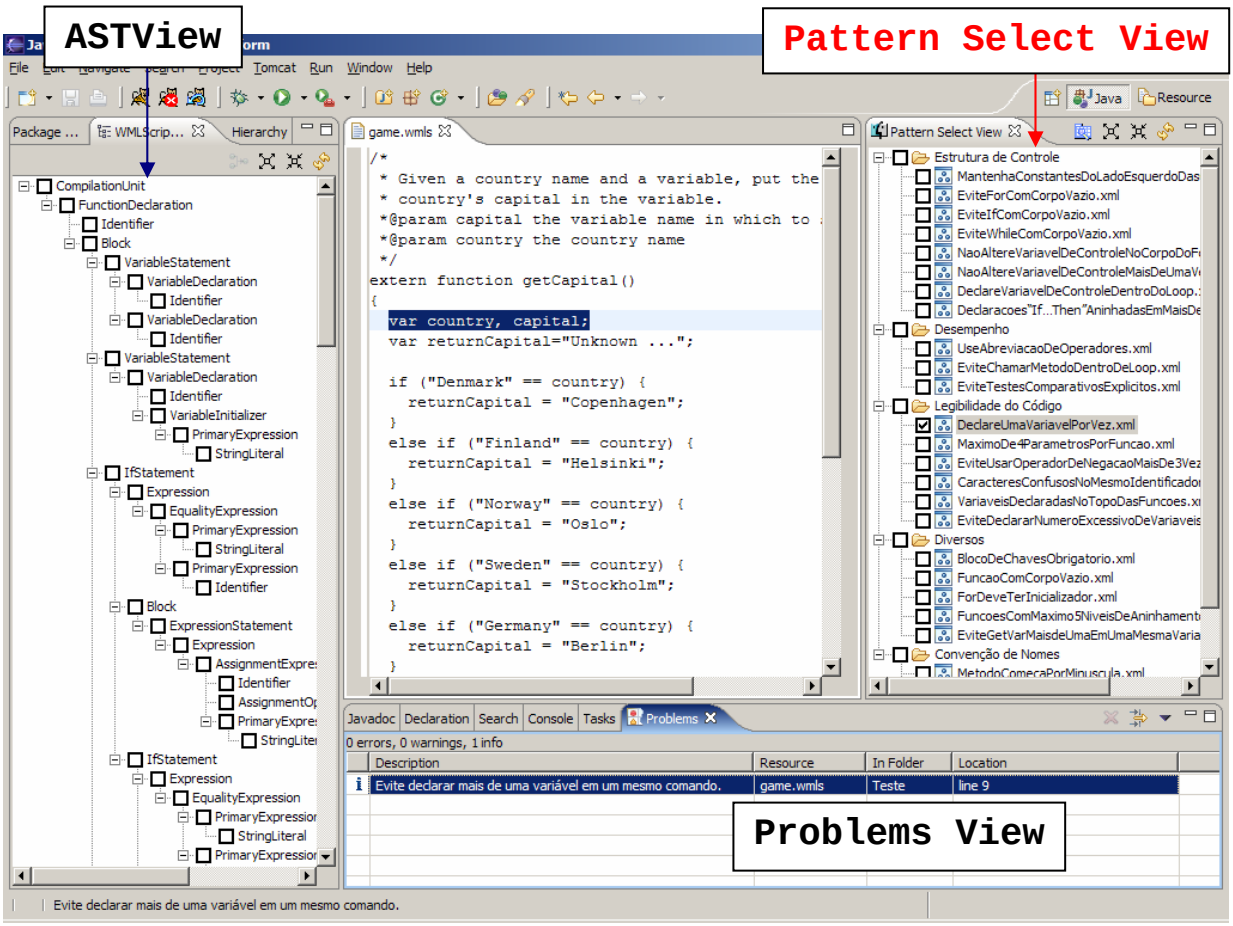

FIG. 5.5 Ferramenta de detecção de padrões de código do *WMLScript* .

Na próxima seção vamos detalhar como procedemos na adequação da interface da ferramenta para a linguagem de programação *WMLScript*, foi preciso estender o *framework* do Eclipse no *plugin* WPD, para adaptar a interface para o desenvolvimento com o *WMLScript* e para realizar a detecção dos padrões de código.

5.2.1 ADEQUAÇÃO DA INTERFACE PARA O *WMLSCRIPT*

A fim de adequar a ferramenta de detecção de padrões para trabalhar com o *WMLScript* foi preciso configurar o *plugin* WPD para estender o Eclipse nos pontos de extensão de interface com o usuário. O primeiro passo foi apresentar para o Eclipse as classes que vão

implementar as *views AST* e *Pattern Select*, posteriormente estendemos uma funcionalidade que o Eclipse disponibiliza para inserir marcações no editor e avisar na *problems view*, esta última foi utilizada para interface de detecção de padrões. Abaixo apresentamos a configuração das extensões no descritor *plugin.xml* que acompanha o projeto e foi utilizado para descrever os pontos de integração da ferramenta com a plataforma Eclipse.

```
<plugin>
  <extension name="Views of WMLScript Pattern Detector"
      id="WMLScriptPatternDetect"
    point="org.eclipse.ui.views">
    <view name="Pattern Select View"
            icon="img/patternselect.gif"
            category="WMLScriptPatternDetect"
            class="br.eb.ime.de9.wspd.view.
            patternselectview.PatternSelectView"
            id="br.eb.ime.de9.wspd.view.
            patternselectview.PatternSelectView">
      </view>
    <view name="WMLScript AST View"
        icon="img/astview.gif"
        category="WMLScriptPatternDetect"
        class="br.eb.ime.de9.wspd.view.astview.ASTView"
        id="br.eb.ime.de9.wspd.view.astview.ASTView">
      </view>
  </extension>
  <extension name="codeInspectionMarker"
    id="codeInspectionMarker"
      point="org.eclipse.core.resources.markers">
    <super type="org.eclipse.core.resources.problemmarker"/>
    <super type="org.eclipse.core.resources.textmarker"/>
    <persistent value="false"/>
   </extension>
</plugin>
```

FIG. 5.6 Configuração das extensões do Eclipse no descritor do *plugin*.

No projeto do *plugin* implementamos no pacote *br.eb.ime.de9.wspd.view.astview* as classes para visualização da AST dos programas, estas classes desenham a AST a partir da árvore gerada pelo analisador sintático desenvolvido para o *WMLScript*. Já o pacote *br.eb.ime.de9.wspd.view.patternselectview* implementa as classes responsáveis por disponibilizar em uma interface amigável para o usuário uma listagem com todos os padrões de código implementados e disponíveis para análise dos programas desenvolvidos em *WMLScript*.

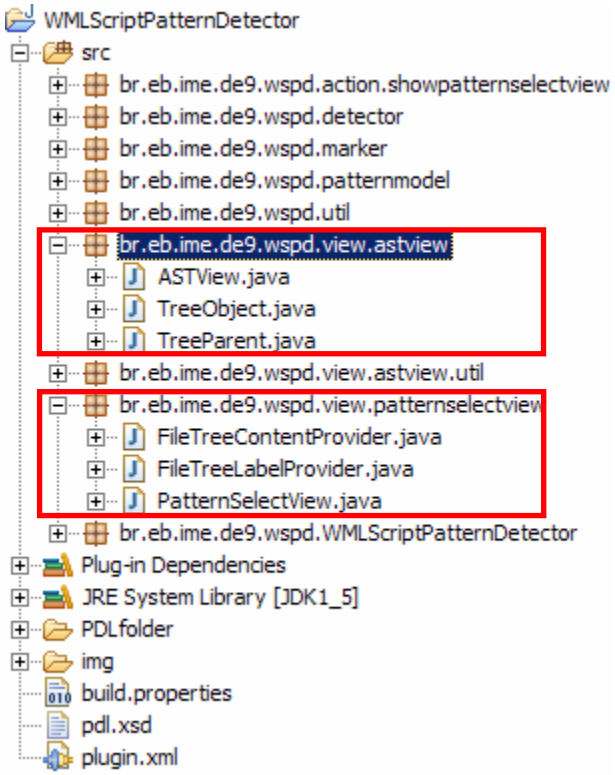

FIG. 5.7 Pacotes com as classes implementadas para interface com o usuário.

A seguir vamos explicar como procedemos no processo de integração do *plugin* WPD com o analisador sintático desenvolvido para o *WMLScript*, as funcionalidades que reutilizamos da PDM e os ajustes que foram necessários para realizar a detecção dos padrões de código.

5.2.2 INTEGRAÇÃO COM O ANALISADOR SINTÁTICO DO *WMLSCRIPT*

O *plugin* WPD foi implementado com a mesma arquitetura da PDM (capítulo 2) na plataforma Eclipse. Para realizar a detecção de padrões de código no *WMLScript* foi preciso integrar a ferramenta com o analisador sintático.

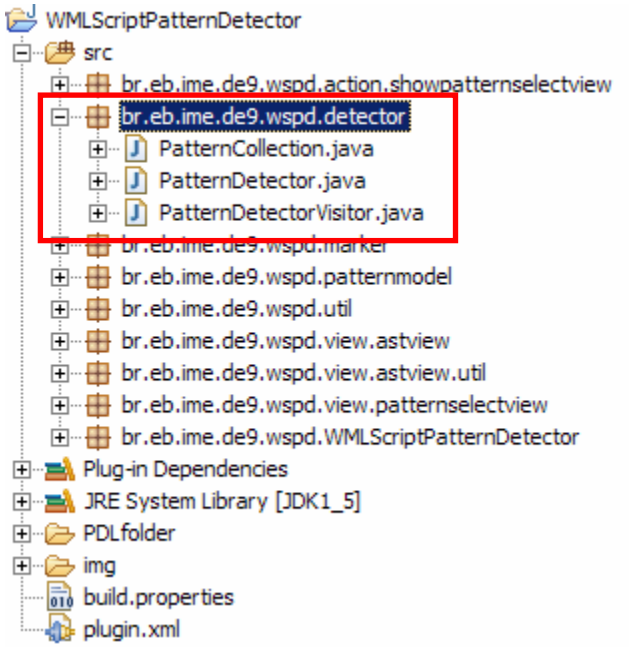

FIG. 5.8 Projeto do *plugin* WPD na plataforma Eclipse.

No pacote *br.eb.ime.de9.wspd.detector* destacado na FIG. 5.6, implementamos a classe *PatternDetector* que utiliza o método *run* para capturar o código fonte do programa no editor e realizar uma chamada ao analisador sintático do *WMLScript* disponível no *plugin* WDT.

```
/**
 * Captura o código fonte do programa no editor do Eclipse.
 * Realiza a chamada ao analisador sintático do WMLScript.
 */
public void run() {

    //limpa as marcações dos padrões previamente encontrados
    markerBeanCollection.clear();
    markerClear();

    //captura o código do programa no editor do Eclipse
       String editorContent = getEditorContent();
       InputStream in = new
       Java.io.ByteArrayInputStream(editorContent.getBytes());

       //Chama o analisador sintático para o programa WMLScript
       CompilationUnit cu = ASTFactory.buildAst(in);
      cu.accept(patternDetectorVisitor);
}
```

FIG. 5.9 Método que implementa a chamada ao analisador sintático do *WMLScript*.

Ainda no mesmo pacote a classe *PatternCollection* implementa os métodos *descendingVisit* e *ascendingVisit* que são executados durante a visita na descida e na subida dos nós da árvore sintática dos programas, estes métodos implementam as regras do algoritmo de detecção de padrões de código da PDM. Nesta etapa foi necessário ajustar o algoritmo para utilizar a AST desenvolvida para o *WMLScript* e permitir que o mesmo reconheça os atributos particulares da linguagem. Na FIG. 5.8 expandimos os pacotes *br.eb.ime.de9.wspd.patternmodel* e *br.eb.ime.de9.wspd.util* que implementam as classes responsáveis pelo reconhecimento dos atributos da AST do *WMLScript.*

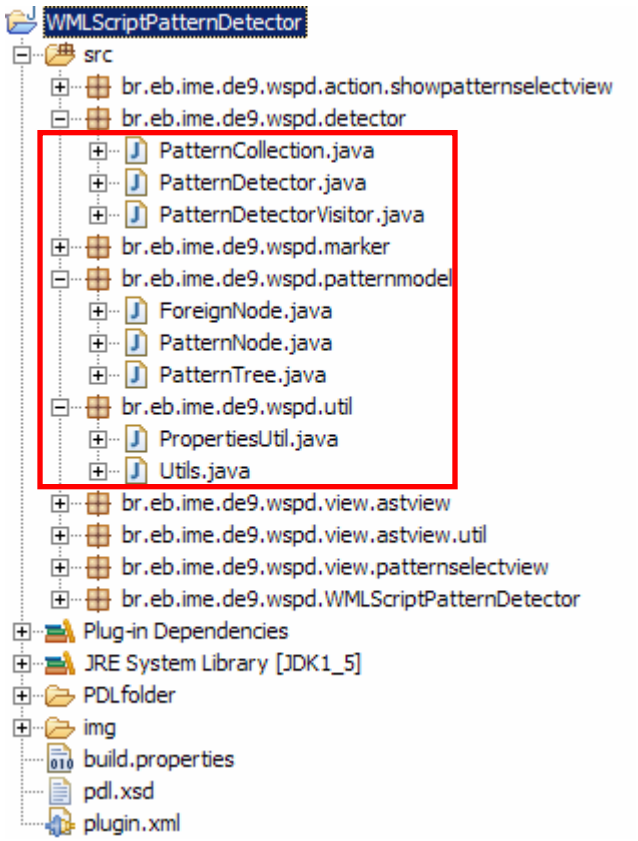

FIG. 5.10 Classes que implementam os atributos da AST do *WMLScript.*

O pacote *br.eb.ime.de9.wspd.patternmodel* implementa as classes utilizadas para representação dos padrões de código. Utilizamos a API DOM para ler os padrões, assim como os atributos e elementos especiais definidos em PDL. No algoritmo de detecção de padrões,

procuramos por casamentos entre os elementos do padrão e os nós da AST durante a visita da árvore, neste momento são verificados os atributos que podem ser descritivos ou definir algum comportamento desejado. Vamos explicar como procedemos na implementação dos atributos específicos do *WMLScript*. Os atributos especiais *_Mark, _Not, _InParentSubTree, _MaxSubTreeLevel* e *_Bind* foram explicados no capítulo 2 e não tiveram qualquer alteração nesta dissertação.

5.2.3 ATRIBUTOS DA AST DO *WMLSCRIPT*

A fim de detectar os padrões de código do *WMLScript* tivemos que adaptar o *plugin* WPD para reconhecer os atributos específicos para AST da linguagem. Implementamos um procedimento para o algoritmo de detecção de padrões identificar os atributos e em seguida armazenar o conteúdo dos mesmos. O primeiro atributo implementado foi o atributo “nome” do elemento *Identifier*. O padrão que busca por ocorrências de variáveis declaradas com letra maiúscula da categoria convenção de nomes é um exemplo de padrão que utiliza este atributo. A seguir mostramos o padrão em PDL.

```
<VariableDeclaration>
   <Identifier nome="[A-Z].*" _Mark="Variavel deve ter o nome iniciado”
   com letra minuscula"/>
</VariableDeclaration>
```

FIG. 5.11 Padrão que busca por variáveis declaradas com letra maiúscula.

O algoritmo deve ser capaz de identificar o atributo “nome” no elemento *identifier*, armazenar a expressão regular e realizar o teste para verificar se a *string* guardada para variável satisfaz a expressão regular caracterizando uma ocorrência do padrão. O segundo atributo implementado foi o “*operator*” no elemento *UnaryExpression*, utilizamos este atributo no padrão de código da categoria legibilidade, que busca por ocorrências do operador de negação sendo utilizado mais de três vezes em uma mesma função.

```
<FunctionDeclaration _Distinct="1" _Mark="Evite utilizar operador de
negação mais de 3 vezes na mesma função.">
    <UnaryExpression operator="!"/>
    <UnaryExpression operator="!"/>
    <UnaryExpression operator="!"/>
    <UnaryExpression operator="!"/>
</FunctionDeclaration>
```

FIG. 5.12 Padrão de código utilizando o atributo *operator* no elemento *UnaryExpression*

Outro exemplo é o atributo *value* do elemento *StringLiteral*, utilizado no padrão de código do *WMLScript* que restringe utilizar a função *GetVar*[4] mais de uma vez em uma mesma variável. Descrevemos na FIG. 5.11 um padrão implementado em PDL utilizando o atributo *value*, neste caso específico verificamos com o atributo especial *_Bind* a existência de mais de uma chamada à função *GetVar* em uma mesma variável.

```
<FunctionDeclaration _Mark="Evite Utilizar GetVar mais de uma vez em uma
mesma variável.">
    <LibraryFunctionCall>
        <LibraryName>
            <Identifier nome="WMLBrowser"/>
        </LibraryName>
        <FunctionName>
            <Identifier nome="getVar"/>
        </FunctionName>
        <Arguments>
            <StringLiteral _Bind="X,value"/>
        </Arguments>
    </LibraryFunctionCall>
    <LibraryFunctionCall>
        <LibraryName>
            <Identifier nome="WMLBrowser"/>
        </LibraryName>
        <FunctionName>
            <Identifier nome="getVar"/>
        </FunctionName>
        <Arguments>
            <StringLiteral _Bind="X,value"/>
        </Arguments>
    </LibraryFunctionCall>
</FunctionDeclaration>
```

FIG. 5.13 Padrão de código utilizando o atributo *value* no elemento *StringLiteral*

[4] *GetVar* é uma função da biblioteca *WMLBrowser* do *WMLScript*, ela é utilizada para capturar o conteúdo de uma variável passada como parâmetro.

5.3 CONSIDERAÇÕES FINAIS

Neste capítulo detalhamos os *plugins* implementados para realizar a detecção de padrões de código do *WMLScript* na plataforma Eclipse, o primeiro WDT ficou responsável por manter as funcionalidades relacionadas à análise sintática no *WMLScript*, enquanto no *plugin* WPD implementamos a interface para detecção dos padrões de código e a integração com o algoritmo de detecção da PDM. No próximo capítulo vamos descrever os vinte e quatro padrões de código implementados para analisar os programas desenvolvidos em *WMLScript.*

6 DEFINIÇÃO DOS PADRÕES DE CÓDIGO PARA O WMLSCRIPT

Os sistemas de *software* estão a cada dia maiores e mais complexos para atender as demandas por novas tecnologias e funcionalidades. A utilização dos padrões de código a partir do inicio de um projeto pode render resultados significativos na redução dos altos custos de manutenção dos sistemas, na prevenção de *bugs* utilizando padrões fundamentados em experiências passadas, além de contribuir para qualificação dos programadores que são ambientados com boas práticas de codificação. Com o uso dos padrões de código a produtividade cresce em situações que um programador tem que compreender ou alterar código de algum outro profissional. A padronização contribui para uniformizar o código nos sistemas, é um trabalho contínuo que deve ser executado durante todo o período de codificação e pode auxiliar consideravelmente programadores que costumam preocupar somente se os sistemas estão funcionando corretamente, esquecendo de boas práticas que podem evitar *bugs*, contribuir para o desempenho da aplicação ou simplesmente aumentar a clareza e legibilidade do código.

Os padrões de código não devem ser restritos a convenções de nomenclatura, é importante considerar regras que podem reduzir a probabilidade de erros nos programas. A definição de melhores práticas, muitas vezes é uma questão particular da linguagem, para propor os padrões apresentados, estudamos a especificação do *WMLScript*, realizamos testes de desempenho e questionamos programadores sobre práticas suscetíveis a erros. A fim de complementar o estudo, consultamos convenções estabelecidas para outras linguagens [HOTWORK; HOFF; VBSCRIPT; JAVASCRIPT]. Como resultado, definimos um conjunto de vinte e quatro padrões nas categorias de convenção de nomes, legibilidade do código, estruturas de controle, desempenho e diversos. Os padrões foram implementados na linguagem PDL, e estão disponíveis na ferramenta de detecção de padrões para o *WMLScript.*

A seguir explicamos cada um dos padrões implementados, apresentamos um exemplo em *WMLScript* e uma justificativa para utilização de acordo com as categorias estabelecidas, enfatizamos as possíveis contribuições para melhoria do código nos sistemas que utilizam o *WMLScript*.

6.1 PADRÕES DE CÓDIGO DO *WMLSCRIPT*

Apesar dos padrões de código serem uma prática amplamente difundida em diversas linguagens de programação, como por exemplo, em Java onde a *sun* disponibiliza um conjunto de convenções e melhores práticas, não constatamos a existência de qualquer padronização para o código *WMLScript*, acreditamos que isto deve-se ao fato do *WMLScript* ser uma linguagem relativamente pouco explorada no desenvolvimento de aplicações de maior porte. Entretanto, há iniciativas, como as citadas no inicio da dissertação [UPT; DIVAS; AKMOSOFT], para utilização da linguagem em aplicações maiores, nas quais a necessidade de adoção de padrões se justifica. Propomos nesta dissertação um conjunto de padrões de código para controlar o desenvolvimento das aplicações em *WMLScript*, os padrões foram divididos nas categorias: convenção de nomenclatura, estruturas de controle, legibilidade do código, desempenho e diversos. Os padrões descritos e justificados a seguir foram implementados na linguagem de descrição de padrões PDL e estão disponíveis para análise de aplicações desenvolvidas em *WMLScript*, bastando para isso utilizar a ferramenta de detecção de padrões de código proposta nesta dissertação e disponível integrada ao ambiente Eclipse.

6.1.1 CONVENÇÃO DE NOMENCLATURA

Os padrões da categoria convenção de nomenclatura devem levar à utilização de uma nomenclatura homogênea pelos programadores. Procuramos com os padrões desta categoria facilitar a identificação de variáveis, métodos e das varáveis de controle das estruturas de *for*. Os padrões desta categoria fazem sentido quando utilizados durante todo processo de codificação de um sistema.

***Variables names must begin with a lower-case letter – Pattern* 1.1**

Justificativa	Iniciar todas as variáveis com letra minúscula ajuda na legibilidade do código.
Exemplo	`Function declareVariaveis(){`

	var **Abacaxi**; //Violação
	var **Laranja**; //Violação
	}

Padrão em PDL	<VariableDeclaration> <Identifier nome="[A-Z].*" _MaxSubTreeLevel="1" _Mark="Variavel deve ter o nome comecado com letra minuscula"/> </VariableDeclaration>
Conseqüência	Legibilidade, Manutenibilidade.

***All functions names must begin with a lower-case letter – Pattern* 1.2**

Justificativa	Comece o nome das funções com letra minúscula. Isso torna o código mais legível.

Exemplo	Function **DeclareVariaveis**(){ //Violação
	....
	}

Padrão em PDL	<FunctionDeclaration _Mark="Nomes de métodos devem começar com letras minúsculas."> <Identifier nome="[A-Z].*" _MaxSubTreeLevel="1"/> </FunctionDeclaration>
Conseqüência	Legibilidade, Manutenibilidade

***For loops variables must be identified as i, j, k – Pattern* 1.3**

Justificativa	Use os nomes i, j, k para variáveis de controle de um *for* (quando do tipo inteiro). A convenção é resultado de uma tradição. O uso desses nomes deve ser uniforme por todos os códigos do projeto, tornando assim fácil e rápida a identificação de variáveis de controle em um *for*.

Exemplo	for (**z** = 0; **z** < 100 ; ++**z**) { //Violação
	j = j + 1;
	}

Padrão em PDL	<FunctionDeclaration> <OR> <ForStatement _Mark="For com nome de variavel de controle diferente de (i,j,k)"> <VariableDeclaration _MaxSubTreeLevel="1"> <Identifier nome="[^i-k]" _MaxSubTreeLevel="1"/> </VariableDeclaration> <Expression _MaxSubTreeLevel="1"/> <Expression _MaxSubTreeLevel="1"/> </ForStatement> <ForStatement _Mark="For com nome de variavel de controle diferente de (i,j,k)"> <Expression _MaxSubTreeLevel="1"> <AssignmentExpression> <Identifier nome="[^i-k]" _MaxSubTreeLevel="1"/> </AssignmentExpression> </Expression> <Expression _MaxSubTreeLevel="1"/> <Expression _MaxSubTreeLevel="1"/>

	</ForStatement> <ForStatement _Mark="For com nome de variavel de controle diferente de (i,j,k)"> <PrimaryExpression _InParentSubTree="Expression"> <Identifier nome="[^i-k]" _MaxSubTreeLevel="1"/> </PrimaryExpression> <Expression _MaxSubTreeLevel="1"/> <Expression _MaxSubTreeLevel="1"/> </ForStatement> </OR> </FunctionDeclaration>
Conseqüência	Legibilidade, Manutenibilidade.

6.1.2 ESTRUTURAS DE CONTROLE

Uma melhor utilização das estruturas de controle disponíveis em *WMLScript* pode contribuir para manutenibilidade dos programas desenvolvidos na linguagem, consideramos que os padrões propostos podem favorecer a simplicidade e clareza do código nos programas.

***Avoid "if…then" nested in more than 3 levels - Pattern* 2.1**

Justificativa	Uma lógica utilizando *Ifs* aninhados em mais de 3 níveis pode ser difícil de compreender.

Exemplo	Function ifAninhados (){
	if (x == true){
	if (y == true){
	if (z == true){
	if (true){} //Violação
	}
	}
	}
	}

Padrão em PDL	<IfStatement _Distinct="1" _Mark="If aninhado em mais de 3 níveis."> <Block> <IfStatement> <Block> <IfStatement> <Block> <IfStatement/> </Block> </IfStatement>

	</Block> </IfStatement> </Block> </IfStatement>
Conseqüência	Legibilidade, Manutenibilidade.

***Declare the for loop control variable inside the loop itself - Pattern* 2.2**

Justificativa	Declare variável de controle dentro do *for* ao invés de utilizar uma variável já existente.

Exemplo	Function ifAninhados (){
	var i;
	for (i = 0; i < 100 ; ++i) //Violação
	{
	}
	}

Padrão em PDL	<ForStatement _Mark="Variáveis de controle devem ser declaradas dentro do for."> <VariableDeclaration _MaxSubTreeLevel="1" _Not="true"/> </ForStatement>
Conseqüência	Legibilidade, Manutenibilidade.

***Avoid for statements with empty body - Pattern* 2.3**

Justificativa	*Statements for* seguidos imediatamente por ponto e vírgula ou com bloco vazio geralmente são estruturas desnecessárias e podem gerar erros de entendimento ao sugerir que o comando seguinte faz parte do *for*.
Exemplo	**for (i = 0; i < 100 ; ++i);** //Violação
Padrão em PDL	<FunctionDeclaration> <OR> <ForStatement _Mark="Evite For com corpo vazio."> <Block _MaxSubTreeLevel="1"> <Node _Not="true"/> </Block> </ForStatement> <ForStatement _Mark="Evite For com corpo vazio."> <EmptyStatement _MaxSubTreeLevel="1"/> </ForStatement> </OR> </FunctionDeclaration>
Conseqüência	Legibilidade.

***Avoid if statements with empty body - Pattern* 2.4**

Justificativa	*Statements if* seguidos imediatamente por ponto e vírgula ou com bloco vazio geralmente são estruturas desnecessárias e podem gerar erros de entendimento ao sugerir que o comando seguinte faz parte do *if*.

Exemplo	`if ( y == true );     //Violação`
	`if ( y == true ){`
	`}                    //Violação`

Padrão em PDL	`<FunctionDeclaration>`
	`  <OR>`
	`  <IfStatement _Mark="Evite If com corpo vazio.">`
	`  <Block _MaxSubTreeLevel="1">`
	`  <Node _Not="true"/>`
	`  </Block>`
	`  </IfStatement>`
	
	`  <IfStatement _Mark="Evite If com corpo vazio.">`
	`  <EmptyStatement _MaxSubTreeLevel="1"/>`
	`  </IfStatement>`
	`  </OR>`
	`</FunctionDeclaration>`

Conseqüência	Legibilidade.

***Avoid while statements with empty body – Pattern* 2.5**

Justificativa	Se um *statement while* estiver seguido imediatamente por um ponto e vírgula, ou com bloco vazio geralmente são estruturas desnecessárias e podem gerar erros de entendimento ao sugerir que o comando seguinte faz parte do *while*.

Exemplo	`while ( x > y );        //Violação`
	`while ( x > y )`
	`{}                    //Violação`

Padrão em PDL	`<FunctionDeclaration>`
	`  <OR>`
	`  <WhileStatement _Mark="Evite While com corpo vazio.">`
	`  <Block _MaxSubTreeLevel="1">`
	`  <Node _Not="true"/>`
	`  </Block>`
	`  </WhileStatement>`
	
	`  <WhileStatement _Mark="Evite While com corpo vazio.">`
	`  <EmptyStatement _MaxSubTreeLevel="1"/>`
	`  </WhileStatement>`
	`  </OR>`
	`</FunctionDeclaration>`

Conseqüência	Legibilidade.

***Use constants in the left side of the comparisons – Pattern* 2.6**

Justificativa	Utilize constantes do lado esquerdo das comparações. Uma digitação comum enquanto escrevemos código é usar "=" ao invés de "==" em operações de igualdade. Colocando constantes no lado esquerdo da comparação, isso irá fazer com que o compilador gere uma mensagem de erro. Por exemplo, o compilador do *WMLScript* dará uma mensagem de erro para: if(3 = x) mas não para: if(x = 3).
Exemplo	`if(x == 3)  //Violação` `if(3 == x)  //Correto`

Padrão em PDL	`<FunctionDeclaration>`
	`<OR>`
	`<EqualityExpression _Distinct="1" _Mark="As constantes devem ficar do lado esquerdo das comparações.">`
	`<Identifier/>`
	`<IntegerLiteral/>`
	`</EqualityExpression>`
	
	`<EqualityExpression _Distinct="1" _Mark="As constantes devem ficar do lado esquerdo das comparações.">`
	`<Identifier/>`
	`<BooleanLiteral/>`
	`</EqualityExpression>`
	
	`<EqualityExpression _Distinct="1" _Mark="As constantes devem ficar do lado esquerdo das comparações.">`
	`<Identifier/>`
	`<FloatLiteral/>`
	`</EqualityExpression>`
	
	`<EqualityExpression _Distinct="1" _Mark="As constantes devem ficar do lado esquerdo das comparações.">`
	`<Identifier/>`
	`<StringLiteral/>`
	`</EqualityExpression>`
	`</OR>`
	`</FunctionDeclaration>`

Conseqüência	Manutenibilidade.

***Do not alter a control variable more than once in while structures - Pattern* 2.7**

Justificativa	O comportamento destas estruturas quando há modificações em suas variáveis de controle dificulta a manutenção e compreensão.

Exemplo	`while ( x < 5){`
	`x++;`
	`x = x + 2;  //Violação`
	`}`

Padrão em PDL	`<FunctionDeclaration>`
	`<OR>`
	`<WhileStatement _Distinct="1" _Mark="While com variavel de controle atualizada mais de uma vez.">`
	`<Expression _MaxSubTreeLevel="1">`
	`<Identifier _Bind="X,nome" />`
	`</Expression>`

	<Block _MaxSubTreeLevel="1"> <OR> <PostfixExpression> <Identifier _Bind="X,nome" /> </PostfixExpression> <AssignmentExpression> <Identifier _Bind="X,nome" /> </AssignmentExpression> </OR> </Block> </WhileStatement> <WhileStatement _Distinct="1" _Mark="While com variavel de controle atualizada mais de uma vez."> <Expression _MaxSubTreeLevel="1"> <Identifier/> <Identifier _Bind="X,nome" /> </Expression> <Block _MaxSubTreeLevel="1"> <OR> <PostfixExpression> <Identifier _Bind="X,nome" /> </PostfixExpression> <AssignmentExpression> <Identifier _Bind="X,nome" /> </AssignmentExpression> </OR> </Block> </WhileStatement> </OR> </FunctionDeclaration>
Conseqüência	Legibilidade, Manutenibilidade.

***Do not alter a control variable inside for structures - Pattern* 2.8**

Justificativa	O comportamento destas estruturas quando há modificações em suas variáveis de controle dificulta a manutenção e compreensão.

Exemplo	for (i = 0; i < 100 ; ++i)
	{
	if (s == ""){
	i++; //Violação
	}
	i = 5; //Violação
	}

Padrão em PDL	<FunctionDeclaration> <OR> <ForStatement _Mark="For com variavel de controle atualizada no corpo"> <VariableDeclaration _MaxSubTreeLevel="1"> <Identifier _Bind="X,nome" /> </VariableDeclaration> <Block _MaxSubTreeLevel="1"> <AssignmentExpression> <Identifier _Bind="X,nome" _MaxSubTreeLevel="1"/> </AssignmentExpression>

	</Block> </ForStatement> <ForStatement _Mark="For com variavel de controle atualizada no corpo"> <VariableDeclaration _MaxSubTreeLevel="1"> <Identifier _Bind="X,nome" /> </VariableDeclaration> <Block _MaxSubTreeLevel="1"> <PostfixExpression> <Identifier _Bind="X,nome" _MaxSubTreeLevel="1"/> </PostfixExpression> </Block> </ForStatement> <ForStatement _Mark="For com variavel de controle atualizada no corpo"> <Expression _MaxSubTreeLevel="1"> <AssignmentExpression> <Identifier _Bind="X,nome" /> </AssignmentExpression> </Expression> <Block _MaxSubTreeLevel="1"> <AssignmentExpression> <Identifier _Bind="X,nome" _MaxSubTreeLevel="1"/> </AssignmentExpression> </Block> </ForStatement> <ForStatement _Mark="For com variavel de controle atualizada no corpo"> <Expression _MaxSubTreeLevel="1"> <AssignmentExpression> <Identifier _Bind="X,nome" /> </AssignmentExpression> </Expression> <Block _MaxSubTreeLevel="1"> <PostfixExpression> <Identifier _Bind="X,nome" _MaxSubTreeLevel="1"/> </PostfixExpression></Block> </ForStatement></OR></FunctionDeclaration>
Conseqüência	Legibilidade, Manutenibilidade.

6.1.3 LEGIBILIDADE DO CÓDIGO

Nesta categoria definimos padrões que verificam a utilização de práticas não recomendadas que podem de alguma forma comprometer a legibilidade ou a compreensão do código nos programas.

***Do not use similar characters in the same identifier – Pattern* 3.1**

Justificativa	Não utilize os caracteres ‘l’(ele minúsculo) e ‘1’ (um) ou ‘O’ e ‘0’ (zero) no mesmo identificador. Por exemplo, é preferível usar o "L" ao invés de "l", pois o "l" pode ser confundido facilmente com o número um.

Exemplo	`Function declareVariaveis(){`
	`var 1l; //Violação`
	`}`

Padrão em PDL	`<Identifier nome=".*(l.*1|1.*l|o.*0|0.*o|O.*0|0.*O).*" _Mark="Não utilize caracteres parecidos no mesmo identificador"/>`
Conseqüência	Legibilidade, Manutenibilidade.

***Avoid declaring more than one variable in a same command line – Pattern* 3.2**

Justificativa	Não declare mais de uma variável em uma mesma instrução. A declaração de várias variáveis de uma só vez pode se tornar ilegível.

Exemplo	`Function declareVariaveis(){`
	`  var t1, t2;  //Violação`
	`  var t3;`
	`  var t4;`
	`}`

Padrão em PDL	`<VariableStatement _Mark="Evite declarar mais de uma variável em um mesmo comando.">`
	`  <VariableDeclaration/>`
	`  <VariableDeclaration/>`
	`</VariableStatement>`

Conseqüência	Legibilidade.

***Avoid declaring an excessive number of variables inside of just one function – Pattern* 3.3**

Justificativa	Esse padrão é importante para manter a simplicidade das funções, colaborando com a legibilidade e manutenibilidade.

Exemplo	`Function declareVariaveis(){`
	`  var t1;`
	`  var t2;`
	`  var t3;`
	`  var t4;`
	`  var t5;`
	`  var t6;  //Violação`
	`}`

Padrão em PDL	`<FunctionDeclaration _Distinct="1" _Mark="Evite declarar numero excessivo de variaveis numa mesma função.">`
	`  <VariableDeclaration/>`
	`  <VariableDeclaration/>`
	`  <VariableDeclaration/>`
	`  <VariableDeclaration/>`
	`  <VariableDeclaration/>`
	`</FunctionDeclaration>`

Conseqüência	Legibilidade, Manutenibilidade.

***Avoid the excessive use of the logical not operator – Pattern* 3.4**

Justificativa	O operador de negação prejudica a legibilidade do código. Use no máximo três vezes esse operador dentro de uma mesma função.

Exemplo	`Function declareVariaveis(){`
	`  var j;`
	`  var y;`
	`for (i = 0; i < 100 ; ++i ){`
	`  if ( !s == !j ){`
	`  y = !j;`
	`  j = !y; //Violação`
	`  }`
	`  }`
	`}`

Padrão em PDL	`<FunctionDeclaration _Distinct="1" _Mark="Evite utilizar operador de negação mais de 3 vezes na mesma função.">`
	`  <UnaryExpression operator="!"/>`
	`  <UnaryExpression operator="!"/>`
	`  <UnaryExpression operator="!"/>`
	`  <UnaryExpression operator="!"/>`
	`</FunctionDeclaration>`

Conseqüência	Legibilidade, Manutenibilidade.

***Limit the number of parameters declared by function – Pattern* 3.5**

Justificativa	Uma grande quantidade de parâmetros indica complexidade para interfacear a chamada às funções e deve ser evitado. Propomos uma limitação de quatro para o número de parâmetros recebidos por uma função no *WMLScript*.

Exemplo	`Function declareVariaveis(a, b, c, d, e) //Violação`
	`{`
	`}`

Padrão em PDL	`<FunctionDeclaration _Distinct="1" _Mark="Declare no máximo 4 parâmetros por função.">`
	`  <Identifier _MaxSubTreeLevel="1"/>`
	`  <Identifier _MaxSubTreeLevel="1"/>`
	`  <Identifier _MaxSubTreeLevel="1"/>`
	`  <Identifier _MaxSubTreeLevel="1"/>`
	`  <Identifier _MaxSubTreeLevel="1"/>`
	`  <Identifier _MaxSubTreeLevel="1"/>`
	`</FunctionDeclaration>`

Conseqüência	Legibilidade, Manutenibilidade.

6.1.4 DESEMPENHO

Esta categoria de padrões tem uma importância maior para o *WMLScript*, considerando que a linguagem é executada em *microbrowsers* com reduzido poder de processamento, nesta situação otimizações mesmo que pequenas podem revelar bons resultados. Neste sentido realizamos alguns testes de desempenho com estruturas da linguagem que apresentavam comportamentos análogos, em alguns testes conseguimos definir boas práticas como no caso dos padrões 4.2 e 4.3, em outros não tivemos nenhuma melhora de desempenho como foi o caso no qual comparamos a verificação de *string* vazia com o operador "==" ou com a função *String.isEmpty()* disponível na biblioteca do *WMLScript*, neste teste obtivemos tempos iguais de resposta.

***Avoid calling methods inside a loop condition – Pattern* 4.1**

Justificativa	A menos que o compilador otimize-o, a condição do *loop* será calculada para cada iteração sobre o *loop*. Se o valor de condição não sofre mudança, ele será executado mais rápido se a chamada for realizada fora do *loop*.

Exemplo	`Function declareVariaveis(){`
	`  var j;`
	`  for (i = 0; i < bar.size() ; ++i ){ //Violação`
	`  }`
	`}`

Padrão em PDL	`<FunctionDeclaration>`
	`<OR>`
	`<ForStatement _Mark="Evite chamar método dentro de condição de loop.">`
	`  <LocalScriptFunctionCall _InParentSubTree="Expression1"/>`
	`</ForStatement>`
	
	`<ForStatement _Mark="Evite chamar método dentro de condição de loop.">`
	`  <LibraryFunctionCall _InParentSubTree="Expression1"/>`
	`</ForStatement>`
	`</OR>`
	`</FunctionDeclaration>`

Conseqüência	Eficiência.

***Use the increment operator – Pattern* 4.2**

Justificativa	Constatamos que o melhor tempo de resposta para incrementar uma variável é utilizando o operador de incremento.
Exemplo	**i = i +1;** // VIOLACAO **i += 1;** //VIOLAÇÃO i ++ ; //Melhor tempo de resposta.
Padrão em PDL	<ExpressionStatement _Mark="Use operador de incremento."> <OR> <AssignmentExpression> <Identifier/> <AssignmentOpertator/> <PrimaryExpression _MaxSubTreeLevel="1"> <IntegerLiteral value="1"/> </PrimaryExpression> </AssignmentExpression> <AssignmentExpression> <Identifier _Bind="X,nome"/> <AssignmentOpertator/> <AdditiveExpression _MaxSubTreeLevel="1"> <Identifier _Bind="X,nome"/> <AdditiveOperator/> <PrimaryExpression _MaxSubTreeLevel="1"> <IntegerLiteral value="1" _MaxSubTreeLevel="1"/> </PrimaryExpression> </AdditiveExpression> </AssignmentExpression> </OR> </ExpressionStatement>
Conseqüência	Eficiência.

***Avoid explicit comparative tests – Pattern* 4.3**

Justificativa	Para expressões booleanas ('if', 'for', 'while' e o primeiro operando do operador ternário '?:'), não use testes comparativos explícitos do tipo igualdade. A comparação explícita tem um custo de execução maior, a otimização e desempenho são prioridade na codificação com o *WMLScript*.

Exemplo	extern function foo(){
	if (bar()){}
	if (**true != bar()**){}// Violação
	}

Padrão em PDL	<FunctionDeclaration> <OR> <ConditionalExpression _Mark="Evite testes comparativos explícitos"> <PrimaryExpression> <EqualityExpression> <Node/> <BooleanLiteral/> </EqualityExpression> </PrimaryExpression> </ConditionalExpression>

	<ConditionalExpression _Mark="Evite testes comparativos explícitos"> <PrimaryExpression> <EqualityExpression> <BooleanLiteral/> <Node/> </EqualityExpression> </PrimaryExpression> </ConditionalExpression> <WhileStatement _Mark="Evite testes comparativos explícitos"> <EqualityExpression _InParentSubTree="Expression"> <Node/> <BooleanLiteral/> </EqualityExpression> </WhileStatement> <WhileStatement _Mark="Evite testes comparativos explícitos"> <EqualityExpression _InParentSubTree="Expression"> <BooleanLiteral/> <Node/> </EqualityExpression> </WhileStatement> <IfStatement _Mark="Evite testes comparativos explícitos"> <EqualityExpression _InParentSubTree="Expression"> <Node/> <BooleanLiteral/> </EqualityExpression> </IfStatement> <IfStatement _Mark="Evite testes comparativos explícitos"> <EqualityExpression _InParentSubTree="Expression"> <BooleanLiteral/> <Node/> </EqualityExpression> </IfStatement> <ForStatement _Mark="Evite testes comparativos explícitos"> <EqualityExpression _InParentSubTree="Expression1"> <BooleanLiteral/> <Node/> </EqualityExpression> </ForStatement> <ForStatement _Mark="Evite testes comparativos explícitos"> <EqualityExpression _InParentSubTree="Expression1"> <Node/> <BooleanLiteral/> </EqualityExpression> </ForStatement> </OR> </FunctionDeclaration>
Conseqüência	Eficiência.

6.1.5 DIVERSOS

Nesta seção separamos os padrões que não foram enquadrados em nenhuma das categorias propostas até o momento. Definimos padrões para evitar práticas não recomendadas de programação, que podem ser suscetíveis a erros, como por exemplo, os padrões 5.1 e 5.2. Definimos também o padrão 5.4 que evita complexidades desnecessárias limitando um nível máximo de aninhamento para blocos dentro de uma função. E finalmente escrevemos o padrão 5.5 que define uma boa prática para utilização da função *getVar* da biblioteca *WMLBrowser* do *WMLScript*.

***The brace blocks must be mandatory – Pattern* 5.1**

Justificativa	Todo comando de fluxo de controle ('if', 'else', 'while', 'for') deve ser seguido por um bloco com chaves de abertura e fechamento, mesmo que este esteja vazio ou contenha apenas uma linha. A aplicação de chaves para delimitar os blocos aumenta a legibilidade e consistência do código, além de evitar erros como na inserção de algum novo comando que deveria estar inserido no bloco.
Exemplo	`for (i = 0; i < 100 ; ++i );` //Violação

Padrão em PDL	`<FunctionDeclaration>`
	`<OR>`
	`  <IfStatement _Mark="If sem bloco com chaves" >`
	`  <Block _Not="true"/>`
	`  </IfStatement>`
	`  <WhileStatement _Mark="While sem bloco com chaves">`
	`  <Block _Not="true"/>`
	`  </WhileStatement>`
	`  <ForStatement _Mark="While sem bloco com chaves">`
	`  <Block _Not="true"/>`
	`  </ForStatement>`
	`</OR>`
	`</FunctionDeclaration>`

Conseqüência	Legibilidade, Manutenibilidade.

***The for control variable must be initialized – Pattern* 5.2**

Justificativa	Um *loop for* deve ter sua variável de controle inicializada para garantir um funcionamento lógico esperado da estrutura de controle.

Exemplo	**for (var i; i < 100 ; ++i){}** //Violação
Padrão em PDL	<FunctionDeclaration> <OR> <ForStatement _Mark="For deve ter um inicializador"> <Expression _MaxSubTreeLevel="1"> <AssignmentExpression _Not="true"/> </Expression> <Expression _MaxSubTreeLevel="1"/> <Expression _MaxSubTreeLevel="1"/> </ForStatement> <ForStatement _Mark="For deve ter um inicializador"> <VariableDeclaration _MaxSubTreeLevel="1"> <VariableInitializer _Not="true"/> </VariableDeclaration> <Expression _MaxSubTreeLevel="1"/> <Expression _MaxSubTreeLevel="1"/> </ForStatement> </OR> </FunctionDeclaration>
Conseqüência	Legibilidade, Manutenibilidade.

***Avoid functions with empty body – Pattern* 5.3**

Justificativa	Funções vazias devem ser removidas. Exclua as funções com corpo vazio. Primeiramente remova as referências à função e depois remova a função.
Exemplo	**extern function testa(){}** //Violação
Padrão em PDL	<FunctionDeclaration _Mark="Função com Corpo Vazio."> <Block _MaxSubTreeLevel="1"> <Node _Not="true"/> </Block> </FunctionDeclaration>
Conseqüência	Manutenibilidade.

***Use maximum of four levels nesting in blocks inside the functions – Pattern* 5.4**

Justificativa	As funções devem ser do menor tamanho possível, para melhorar a legibilidade e manutenibilidade. Neste contexto consideramos um aninhamento máximo de 4 níveis de blocos razoável para o escopo de uma função.
Exemplo	**for (i = 0; i < 100 ; ++i){** **if (!v){** **if (x){** **if (y){** **if (z){} //Violação** **}**

	} **}** **}**
Padrão em PDL	<FunctionDeclaration _Distinct="1"> <Block> <Block _Mark="As funções devem ter no maximo 5 niveis de aninhamento"> <Block> <Block> <Block/> </Block> </Block> </Block> </Block> </FunctionDeclaration>
Conseqüência	Legibilidade, Manutenibilidade.

***Avoid using the getVar function more than once in the same variable – Pattern* 5.5**

Justificativa	Ao invés de usar *getVar* para uma mesma variável, faça uso de variável temporária. Isto evita que a função método seja novamente executado.

Exemplo	extern function testa()
	{
	var c = WMLBrowser.getVar(“ola”);
	....
	var c = WMLBrowser.getVar(“ola”); //Violação
	}

Padrão em PDL	<FunctionDeclaration _Mark="Evite Utilizar GetVar mais de uma vez em uma mesma variável."> <LibraryFunctionCall> <LibraryName> <Identifier nome="WMLBrowser"/> </LibraryName> <FunctionName> <Identifier nome="getVar"/> </FunctionName> <Arguments> <StringLiteral _Bind="X,value"/> </Arguments> </LibraryFunctionCall> <LibraryFunctionCall> <LibraryName> <Identifier nome="WMLBrowser"/> </LibraryName> <FunctionName> <Identifier nome="getVar"/> </FunctionName> <Arguments> <StringLiteral _Bind="X,value"/> </Arguments> </LibraryFunctionCall> </FunctionDeclaration>
Conseqüência	Eficiência.

6.2 CONSIDERAÇÕES FINAIS

Neste capítulo definimos um conjunto de padrões de código para a linguagem *WMLScript*. Acreditamos que os padrões estabelecidos possam de alguma forma contribuir para melhoria do código nos programas, constatamos atualmente aplicações sendo desenvolvidas em *WMLScript* sem qualquer controle do processo de codificação. Não encontramos nenhum esforço anterior para definição de padrões de código para o *WMLScript*, acreditamos que este fato deve-se à linguagem ser utilizada geralmente para validações ou alguma lógica simples no lado cliente das aplicações. Esperamos desta forma contribuir para melhoria do código nos sistemas, principalmente os sistemas de maior porte construídos em *WMLScript.*

No próximo capítulo realizamos um estudo de caso aplicando os padrões de código em dezesseis programas desenvolvidos em *WMLScript*. Para realização dos testes dispomos as aplicações no ambiente Eclipse e executamos os padrões de código separadamente em cada um dos programas.

7 ESTUDO DE CASO

Realizamos um estudo de caso para verificar a utilidade dos padrões de codificação propostos nesta dissertação e o impacto que eles podem ter nos programas desenvolvidos em *WMLScript*, bem como validar a ferramenta implementada. O estudo de caso foi realizado com a aplicação dos vinte e quatro padrões descritos no capítulo 6 em dezesseis programas *WMLScript*. Os programas utilizados neste estudo de caso são de domínio público e encontram-se disponíveis no *Nokia Mobile Toolkit* [NOKIA], em livros didáticos e artigos que vamos referenciar na TAB. 7.1, quando descrevemos a funcionalidade e a origem de cada um dos programas analisados.

A seguir vamos detalhar a abordagem utilizada neste estudo de caso, elucidamos cada um dos dezesseis programas que foram analisados e os resultados obtidos com a aplicação dos padrões de codificação.

7.1 ABORDAGEM PROPOSTA

O estudo de caso foi realizado em dezesseis programas descritos a seguir que implementam funcionalidades em *WMLScript*. Na TAB. 7.1 detalhamos cada um dos programas que foram analisados, a funcionalidade que eles implementam e uma referência para origem de cada um dos programas.

TAB. 7.1 Programas utilizados no estudo de caso proposto.

Arquivos Analisados	Funcionalidade do Programa	Origem
bships.wmls	Jogo de batalha naval calcula quantidade de acertos e número de tentativas.	[FROST, 2000]
bships2.wmls	Batalha naval este permite selecionar o tamanho do campo da batalha.	[MANN, 2000]

calc.wmls	Calculadora. Realiza as quatro operações básicas.	[FROST, 2000]
comb.wmls	Calcula fatorial e combinatória.	[PONCET]
currency.wmls	Conversão entre moedas de diferentes países.	[NOKIA]
getCapital.wmls	O usuário informa o país e o programa retorna a capital.	[NOKIA]
kitcalc.wmls	Calcula o tamanho de um cômodo e valida informações do tipo ventilação suficiente para o tamanho informado.	[BENNETT, 2000]
logicaforca.wmls	Jogo da forca.	[SCHAFER]
magic.wmls	Jogo "mágico" em que o objetivo é igualar as linhas e colunas de uma matriz.	[SHARP]
mortgage.wmls	Calcula o valor de prestações.	[NOKIA]
schedule.wmls	Programa sumarizado para acompanhamento dos jogos de primavera em Bostom.	[NOKIA]
validateFormEg1.wmls	Valida as informações nome de usuário, senha, data e e-mail.	[DEVELOPERS HOME]
validateFormEg12.wmls	Valida as informações do usuário.	[DEVELOPERS HOME]
validation.wmls	Validação de datas.	[SCHAFER, 2002]
validator.wmls	Verifica se um número pertence a um determinado conjunto de números.	[OPENWAVE, 2000]
windex.wmls	Calcula o índice de massa corporal.	[NOKIA]

Após coletar os programas desenvolvidos em *WMLScript* para realização do estudo de caso, submetemos cada um deles a uma avaliação a partir da aplicação dos padrões propostos no capítulo 6, cujo resultado apresentamos na próxima seção.

7.2 RESULTADOS

Após avaliar cada um dos programas com os padrões propostos, conseguimos resultados interessantes em algumas das categorias para obter melhorias nos programas. Encontramos 21 ocorrências de padrões de nomenclatura. Conseguimos encontrar um total de 92 ocorrências de padrões na categoria de estruturas de controle contribuindo principalmente na manutenibilidade dos programas. Na categoria de legibilidade foram encontradas 15 ocorrências de padrões que podem contribuir para simplificar a lógica dos programas. Na aplicação dos padrões da categoria desempenho, conseguimos encontrar 6 ocorrências, dessas as mais interessantes foram as 3 ocorrências do padrão 4.1 pois dependendo do desempenho da função chamada na condição de um *loop for*, considerando as limitações de ambiente impostas para os programas *WMLScript*, estas ocorrências podem ser um gargalo para aplicação. Finalmente, encontramos 58 ocorrências do padrão 5.1 que determina a obrigatoriedade para utilização dos blocos de chaves, este padrão previne erros de lógica e facilita a legibilidade dos programas.

TAB. 7.2 Ocorrências dos padrões de código nos programas analisados.

Padrão encontrado	**Conseqüência**	**Número de ocorrências**	**Arquivos fonte com alguma ocorrência dos padrões de código**
Padrão 1.1	L, M	12, 4	currency, kitCalc
Padrão 1.2	L, M	1	bships2
Padrão 1.3	L, M	1,2	bships2, forca
Padrão 2.2	L, M	1, 2, 1	bships2, forca, magic
Padrão 2.6	M	6, 8, 21, 1, 21, 3, 2, 26	bships, calc, currency, kitcalc, magic, mortgage, validateFormEg1, validation
Padrão 3.1	L, M	9, 4, 2, 4	validation, magic, validateFormEg1, windex
Padrão 3.2	L	1, 1, 1, 2	bships, bships2, comb, forca
Padrão 3.3	L, M	1, 1, 2, 1, 1, 1, 1	calc, currency, magic, validateFormEg1, kitcalc, validateFormEg12, validation
Padrão 3.4	L, M	1	validation

Padrão 3.5	L, M	1	validateFormEg1
Padrão 4.1	E	1, 2	kitcalc, forca
Padrão 4.2	E	2	bships
Padrão 4.3	E	1	Magic
Padrão 5.1	L, M	5, 16, 8, 10, 2, 6, 11	bships, currency, kitcalc, magic, mortgage, validateFormEg1, validation
Padrão 5.2	L, M	1	Forca

Conseqüência: L – Legibilidade
M – Manutenibilidade
E – Eficiência

7.3 APLICAÇÃO DOS PADRÕES

Exibimos nesta seção um exemplo de trecho de código encontrado para alguns padrões selecionados. Deixamos de fora alguns padrões, sobretudo os de nomenclatura, por julgarmos que seriam mais relevantes em um contexto onde fossem adotados como convenção de codificação. Apresentamos também uma proposta de melhoria, feita manualmente, para cada trecho de código apresentado.

7.3.1 APLICAÇÃO DO PADRÃO 3.4

O padrão 3.4 restringe para três o número de vezes que o operador de negação pode ser utilizado sem comprometer a legibilidade nos programas. A utilização excessiva do operador de negação pode ser desnecessária ou ainda aumentar a complexidade no código. Na FIG. 7.1 apresentamos uma ocorrência do padrão encontrado pela ferramenta no programa

Validation.wmls e na FIG. 7.2 uma solução mais interessante para implementação da mesma lógica utilizando um menor número de operadores.

```
if ((month == 2) && (!leap) && (day > 28))
      ok = false;

////////////////// End of validation ////////////////////////////

if ((!daycheck) || (!monthcheck) || (!yearcheck))   formatwrong();
```

FIG. 7.1 Ocorrência do padrão no programa Validation.wmls.

```
   if ((month == 2) && (!leap) && (day > 28))
   {
      ok = false;
   }
/////////////////////// End of validation ///////////////////////

   if (!(daycheck && monthcheck && yearcheck))
   {
      formatwrong();
   }
```

FIG. 7.2 Melhoria proposta no programa Validation.wmls.

7.3.2 APLICAÇÃO DO PADRÃO 4.1

A chamada de uma função na condição do *loop for* pode comprometer o desempenho da aplicação, a condição do *loop* é calculada em cada iteração. Se o valor da condição não sofre mudança é aconselhável realizar a chamada da função antes da utilização na estrutura de controle. A seguir ilustramos uma ocorrência deste padrão encontrada no programa kitCalc.wmls durante a realização do estudo de caso.

```
for (var i=0; i<String.elements(fuel, SEPARATOR); i++) {
  var thisFuel = String.elementAt(fuel, i, SEPARATOR);
  if (String.find(GASES, thisFuel) >= 0)
    gasesSelected = true;
  if (thisFuel == WOOD)
    woodSelected = true;
  if (gasesSelected && woodSelected)
    break;
}
```

FIG. 7.3 Fragmento de código encontrado no programa kitCalc.wmls.

```
var nElements = String.elements(fuel, SEPARATOR);

for (var i=0; i<nElements; i++) {

   var thisFuel = String.elementAt(fuel, i, SEPARATOR);
   if (String.find(GASES, thisFuel) >= 0)
   {
     gasesSelected = true;
   }
   if (thisFuel == WOOD)
   {
     woodSelected = true;
   }
   if (gasesSelected && woodSelected)
   {
     break;
   }
 }
```

FIG. 7.4 Melhoria proposta no programa kitCalc.wmls.

7.3.3 APLICAÇÃO DO PADRÃO 4.2

Constatamos em testes realizados na linguagem *WMLScript* que o melhor tempo de resposta para incrementar uma variável é utilizando o operador de incremento. No exemplo a seguir mostramos duas ocorrências encontradas do padrão dentro de uma estrutura de *for*.

```
for (var i=0; i<4; ++i) {
    for (var j=0; j<4; ++j) {
            var cell = WMLBrowser.getVar ("d_" + i + "_" + j);
            if (cell != ' ') {
                shots = shots + 1;
                if (cell != '*')
                hits = hits + 1;
            }
     }
}
```

FIG. 7.5 Aplicação do padrão 4.2 no programa bships.wmls.

```
    for (var i=0; i<4; ++i) {
        for (var j=0; j<4; ++j) {
            var cell = WMLBrowser.getVar ("d_" + i + "_" + j);
            if (cell != ' ') {
                shots++;
                if (cell != '*')
                hits++;
            }
        }
    }
```

FIG. 7.6 Melhoria proposta no programa bships.wmls.

7.3.4 APLICAÇÃO DO PADRÃO 4.3

Evite utilizar testes comparativos explícitos para expressões booleanas. A comparação tem um custo de execução maior quando utilizamos o operador “==”. O desempenho é uma prioridade na codificação com o *WMLScript*.

```
while (validinputs == false){
  if (Lang.parseInt(row) > 4){
    row = Dialogs.prompt("Re-enter row value 1 - 4","");
    continue;
  }
  if (Lang.parseInt(col) > 4){
    col = Dialogs.prompt("Re-enter column value 1 - 4","");
    colindex = (Lang.parseInt(col) - 1);
    continue;
  }
  val = Lang.parseInt(val);
  if (val > 100){
    val = Dialogs.prompt("Re-enter value 0 - 100","");
    continue;
  }
  validinputs = true;
}
```

FIG. 7.7 Aplicação do padrão 4.3 no programa magic.wmls.

```
while (!validinputs){
  if (Lang.parseInt(row) > 4){
    row = Dialogs.prompt("Re-enter row value 1 - 4","");
    continue;
  }
  if (Lang.parseInt(col) > 4){
    col = Dialogs.prompt("Re-enter column value 1 - 4","");
    colindex = (Lang.parseInt(col) - 1);
    continue;
  }
  val = Lang.parseInt(val);
  if (val > 100){
    val = Dialogs.prompt("Re-enter value 0 - 100","");
    continue;
  }
  validinputs = true;
}
```

FIG. 7.8 Melhoria proposta no programa magic.wmls.

7.3.5 APLICAÇÃO DO PADRÃO 5.1

Todo comando de fluxo de controle deve ser seguido por um bloco com chaves de abertura e fechamento. A aplicação de chaves para delimitar os blocos aumenta a legibilidade

e consistência do código, além de evitar erros como na inserção de algum novo comando que deveria estar inserido no bloco.

```
  if (to == "DEM")
    multiplier = 1.0;
  else if (to == "FIM")
    multiplier = DEM_FIM;
  else if (to == "FRF")
    multiplier = DEM_FRF;
  else if (to == "USD")
    multiplier = DEM_USD;
```

FIG. 7.9 Ocorrência do padrão 5.1 no programa currency.wmls.

```
if (to == "DEM")
{
   multiplier = 1.0;
}
else if (to == "FIM")
{
   multiplier = DEM_FIM;
}
else if (to == "FRF")
{
   multiplier = DEM_FRF;
}
else if (to == "USD")
{
    multiplier = DEM_USD;
}
```

FIG. 7.10 Melhoria proposta no programa currency.wmls.

7.3.6 APLICAÇÃO DO PADRÃO 5.2

Um *loop for* deve ter sua variável de controle inicializada no *loop* para garantir clareza sobre o funcionamento lógico esperado da estrutura de controle.

```
var idx,hang = 0;
var x = 1;
var blank,word = "";

idx = Lang.random(6);
word = String.elementAt(words,idx," ");
for (x; x <= String.length(word); x++ ) {
  blank = blank + "*";
}
```

FIG. 7.11 Ocorrência do padrão 5.2 no programa logicaForca.wmls.

```
var idx;
var hang = 0;
var blank;
var word = "";
var wLength;

idx = Lang.random(6);
word = String.elementAt(words,idx," ");
wLength = String.length(word);

for (var x = 1; x <= wLength; x++ )
{
  blank = blank + "*";
}
```

FIG. 7.12 Melhoria proposta no programa logicaForca.wmls.

7.4 CONSIDERAÇÕES FINAIS

Neste capítulo procuramos comprovar por um estudo de caso os benefícios que podem ser alcançados com a utilização dos padrões de código propostos para o *WMLScript*. A análise realizada nos programas serviu também para testar a ferramenta. Consideramos que o estudo de caso realizado neste capítulo apresentou resultados satisfatórios na validação da ferramenta com um total de 194 padrões encontrados nos dezesseis programas analisados, acreditamos que a aplicação dos padrões pode realmente contribuir para melhoria da qualidade das aplicações desenvolvidas em *WMLScript*.

8 CONCLUSÕES

Conforme apresentado no inicio da dissertação, constatamos um interesse de empresas no desenvolvimento de aplicações utilizando a linguagem *WMLScript*, por exemplo, a empresa UPT [UPT] que utiliza o *WMLScript* na implementação de jogos, a Divas *Software* [DIVAS] no desenvolvimento de aplicações empresariais e a AKMOSOFT [AKMOSOFT] em sistemas de informação customizados cliente/servidor para *web*, todas utilizam ou já utilizaram o *WMLScript*, na maioria das vezes conjugado a outras linguagens e tecnologias. Entretanto, não encontramos ferramentas para apoiar a codificação nesta linguagem.

Implementamos neste trabalho uma ferramenta para detecção de padrões de código no *WMLScript* análoga a ferramenta implementada por [ALBUQUERQUE, 2005] para a linguagem Java. Em nosso conhecimento, não há outra ferramenta similar disponível para o *WMLScript*, nem tampouco um conjunto de padrões indicado para disciplinar a codificação na linguagem. Para desenvolver a ferramenta, criamos um analisador sintático para *WMLScript* utilizando o gerador JavaCC, e então integramos o analisador sintático ao algoritmo de detecção de padrões desenvolvido por Albuquerque.

Realizamos, ainda, a integração da ferramenta com a plataforma Eclipse, neste propósito desenvolvemos dois *plugins*: o primeiro para manter o analisador sintático do *WMLScript* e realizar as operações na AST dos programas e o segundo, com arquitetura análoga à ferramenta PDM, para realizar a busca dos padrões, conforme explicado no capítulo 5. A utilização de práticas de compiladores na implementação de uma ferramenta para melhoria dos sistemas desenvolvidos em *WMLScript*, foi uma iniciativa interessante e que pode despertar o interesse na utilização das práticas de compiladores na melhoria dos processos de engenharia de *software*.

Neste trabalho definimos os padrões de codificação para o *WMLScript*, justificamos cada um dos padrões e mostramos a sua implementação em PDL. Estes padrões foram resultado de uma pesquisa de recomendações em diversas outras linguagens, entre elas C++ [HOFF], Java [HOTWORK], JavaScript [JAVASCRIPT] e VBScript [VBSCRIPT], além de um estudo detalhado da linguagem *WMLScript* [WMLSCRIPT].

Para verificação da funcionalidade da ferramenta e o impacto da utilização dos padrões, realizamos o estudo de caso apresentado no capítulo 8 e aplicamos os padrões de codificação

em dezesseis programas de pequeno porte desenvolvidos em *WMLScript*. Obtivemos alguns resultados interessantes, como por exemplo, os quinze padrões encontrados na categoria legibilidade para simplificar a lógica dos programas. Na categoria desempenho, encontramos sete ocorrências entre elas destacamos a do padrão 4.1, que controla a utilização de função na condição do *for*. Apresentamos ainda diversas outras ocorrências encontradas dos demais padrões e mostramos os benefícios que podem ser alcançados com a aplicação nos programas *WMLScript*.

8.1 TRABALHOS FUTUROS

Como desenvolvemos a ferramenta integrada na plataforma Eclipse, o próximo objetivo é criar um ambiente de desenvolvimento para o *WMLScript* na mesma plataforma, sugerimos para tanto a implementação de um editor para linguagem, com recursos do tipo *highlight*, *code complete* e *outline view* para navegação entre as funções. Outro objetivo, ainda no ambiente, seria a integração com algum *service pack* da Nokia [NOKIA] para executar os programas desenvolvidos no *WMLScript* sem sair do ambiente de desenvolvimento.

Além disso, existem melhorias a serem realizadas, que se aplicam tanto à versão Java como à versão *WMLScript* da ferramenta. Entre essas melhorias estão:

- *Wizard* para geração de novos padrões.
- Aplicar a inspeção sob vários arquivos simultaneamente.
- Melhorar o tratamento de erros para padrões especificados erroneamente.
- Classificação dos padrões em níveis de prioridades, possibilitando que se faça a inspeção somente para determinados níveis pré-selecionados.
- Criar uma versão para distribuição.

Planeja-se ainda estender a PDL com condições oriundas de análise de fluxo de controle e análise de fluxo de dados, o que permitirá a especificação de padrões mais expressivos.

9 REFERÊNCIAS BIBLIOGRÁFICAS

ALBUQUERQUE, Eduardo S., **Detecção de Padrões de Código Java na Plataforma Eclipse.** 2005. 84p. Dissertação (Mestrado em Engenharia de Sistemas) – Instituto Militar de Engenharia, 2005.

AKMOSOFT. **Akmosoft Software Development Company.** Disponível : http://www.akmosoft.com/index.html [capturado em 24 dez. 2005].

AHO, Alfred, SETHI, Ravi, ULLMAN, Jeffrey. **Compilers Principles, Techniques and Tools**. Addison-Wesley Publishing Company, 1998.

BENNETT, Chris, COYLE, Frank. **WMLScript.** Outubro de 2000. Disponível : http://www.informit.com/articles/article.asp?p=19623&rl=1 [capturado em 09 abr. 2006].

BISON. **Bison**. Manual. Dezembro de 2004. Disponível: http://www.gnu.org/software/bison/manual/ [capturado em 11 jan. 2006].

BOTELHO, Tomas, GUEDES, Luiz Carlos. **An Implementation of a Weakly Typed Language For Resource Constrained Machines.** VI Simpósio Brasileiro de Linguagens de Programação, 2002.

BRIAND, L., EMAM, El K., LAINTENBERGER, O., FUSSBROICH, T., **Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects.** Proceedings of the 20th International Conference on Software Engineering, 1998, pp. 340-349.

CODEPRO. **CodePro Advisor on-line documentation**. Disponível : http://www.instantiations.com/codepro/ws/docs [capturado em 15 out. 2005].

DELAMARO, Márcio. **Como Construir um Compilador Utilizando Ferramentas Java.** Novatec Editora Ltda, 2004. ISBN 85-7522-055-1.

DEVELOPER HOME. **Developers' Home.** Disponível : http://www.developershome.com/ [capturado em 12 ago. 2005].

DIVAS. **Divas Software.** Disponível : http://www.divassoftware.com/french/index.htm [capturado em 12 ago. 2005].

ECLIPSE 2003. **Eclipse Platform Technical Overview**, Object Technology International, fevereiro de 2003. Disponível : http://www.eclipse.org/whitepapers/eclipse-overview.pdf [capturado em 15 jan. 2006].

ECLIPSE. **Site oficial do projeto Eclipse**. Disponível : http://www.eclipse.org [capturado em 12 fev. 2006].

FINDBUGS. **FindBugs.** Manual. Março de 2006. Disponível : http://findbugs.sourceforge.net/manual/index.html [capturado em 12 mar. 2006].

FROST, Martin. **Learning WML & WMLScript.** O'Reilly, 2000. ISBN 1-56592-947-0.

GAMMA, R. Helm, JOHNSON, R., VLISSIDES, J. **Design Patterns – Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1994.

HOFF, Todd. **C ++ Coding Standard.** Disponível : http://www.possibility.com/Cpp/CppCodingStandard.html [capturado em 15 fev. 2006].

HOTWORK. **Java Programming Guidelines**. Fevereiro de 2004. Disponível : http://hotwork.sourceforge.net/hotwork/manual/guidelines/code-guidelines-user-guide.html [capturado em 10 ago. 2005].

JAVA. **Site oficial da linguagem Java**. Disponível : http://java.sun.com [capturado em 10 ago. 2005].

JAVACC. **Site oficial do JavaCC**. Disponível : https://javacc.dev.java.net/ [capturado em 22 jul. 2005].

JAVA.NET. **Java.net the Source for Java Technology Collaboration.** Disponível : http://java.net/ [capturado em 26 fev. 2005].

JAVASCRIPT. **JavaScript Conventions.** Disponível : http://dojotoolkit.org/js_style_guide.html [capturado em 28 mar. 2006].

JDT. **JDT Plug-in Developer Guide**. Disponível : http://www.eclipse.org/documentation/pdf/org.eclipse.jdt.doc.isv_3.0.1.pdf [capturado em 11 set. 2005].

JOHNSON, S.C. **Lint, a C program checker**, In Unix Programer's Manual, volume 2A, capítulo 15, páginas 292-303. Bell Laboratories, 1978.

MANN, Steve. **Programming Applications with the Wireless Application Protocol: The Complete Developer's Guide.** Wiley, 2000.

NOKIA. **Nokia Tools and SDKs.** Disponível : http://www.forum.nokia.com/main/1,6566,033,00.html [capturado em 11 ago. 2005].

OMA. **Site oficial da Open Mobile Alliance** . Disponível : http://www.openmobilealliance.org/index.html [capturado em 26 out. 2005].

OPENWAVE. **OpenWave Systems Inc.** Dezembro de 2000. Disponível : http://www.openwave.com [capturado em 19 dez. 2005].

PMD. **Site oficial do PMD**. Disponível : http://pmd.sourceforge.net/ [capturado em 11 dez. 2005].

PONCET, Eric. **Combination in WML/WMLScript.** Disponível : http://mobile.eric-poncet.com/phone/combination_wml.html [capturado em 22 fev. 2005].

SCHAFER, Steve. **Interactive Fun and Games with WAP and WML.** Disponível : http://www.developer.com/ws/proto/article.php/1472181 [capturado em 26 fev. 2006].

SCHAFER, Steve. **Learning WML - Variables and Scripting.** Julho de 2002. Disponível : http://www.developer.com/lang/other/article.php/1434711 [capturado em 26 fev. 2006].

SHARP, Kevin. **Building Apps with WMLScript.** Disponível : http://www.developer.com/lang/other/article.php/628571 1434711 [capturado em 26 fev. 2006].

SUN. **Site oficial da Sun Microsystems do Brasil**. Disponível : http://br.sun.com/ [capturado em 26 out. 2005].

UPT. **UPT Software Development Company.** Disponível : http://www.uptsoft.com/ [capturado em 10 mar. 2005].

VBSCRIPT. **VBScript Coding Conventions.** Disponível : http://www.bvu.edu/departments/businessservices/infoservices/2fix/policies/appstd.doc [capturado em 09 fev. 2006].

WMLSCRIPT. **WMLScript Specification.** Disponível : http://www.openmobilealliance.org/release_program/browsing_v23.html [capturado em 26 out. 2005].

XML. **XML Specifications**. Official Web Site for XML. Disponível : http://www.w3c.org/XML [capturado em 30 nov. 2005].

XPATH. **Site oficial do XPath**. Disponível : http://www.w3.org/TR/xpath [capturado em 26 fev. 2006].

YACC. **Yacc : Yet Another Compiler-Compiler.** Manual. Disponível : http://dinosaur.compilertools.net/yacc/index.html [capturado em 12 abr. 2005].

10. APÊNDICES

10.1 APÊNDICE 1: ESPECIFICAÇÃO DO ANALISADOR LÉXICO E SINTÁTICO DO *WMLSCRIPT* NO JAVACC

O código abaixo foi implementado no arquivo “wmlscriptParser.jj” no JavaCC.

```
PARSER_BEGIN(ASTFactory)

package br.eb.ime.de9.wsdt.dom.parser;
import java.util.*;
import br.eb.ime.de9.wsdt.dom.ast.*;
import br.eb.ime.de9.wsdt.dom.visitor.*;

class ASTFactory {

  public static CompilationUnit buildAst(java.io.InputStream program)
  {
    CompilationUnit cu = null;
       ASTFactory t = new ASTFactory(program);
       try {
             cu = t.compilationunit();
             GenericVisitor pv = new GenericVisitor();
             cu.accept(pv);
    } catch (Exception e) {
        System.out.println("Oops.");
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
    return cu;
  }
}
PARSER_END(ASTFactory)
```

```
SKIP :
{
    " "
  | "\t"
  | "\n"
  | "\r"
  | <"//" (~["\n","\r"])* ("\n"|"\r"|"\r\n")>
  | <"/*" (~["*"])* "*" (~["/"] (~["*"])* "*")* "/">
}

/* PALAVRAS RESERVADAS */
TOKEN :
    {
            < ACCESS: "access" >
        |   < AGENT: "agent" >
        |   < BREAK: "break">
        |   < CONTINUE: "continue">
        |   < DOMAIN: "domain">
        |   < ELSE: "else">
        |   < EQUIV: "equiv">
        |   < EXTERN: "extern">
        |   < FOR: "for">
        |   < FUNCTION: "function">
        |   < HEADER: "header">
        |   < HTTP: "http">
        |   < IF: "if">
        |   < ISVALID: "isvalid">
        |   < META: "meta">
        |   < NAME: "name">
        |   < PATH: "path">
        |   < RETURN: "return">
        |   < TYPEOF: "typeof">
        |   < USE: "use">
        |   < USER: "user">
        |   < VAR: "var">
        |   < WHILE: "while">
        |   < URL: "url">
    }
```

```
/* OPERADORES */
TOKEN :
    {
            < ATR: "=" >
        |   < MAIOR: ">" >
        |   < MENOR: "<" >
        |   < IGUAL: "==" >
        |   < MENORIGUAL: "<=" >
        |   < MAIORIGUAL: ">=" >
        |   < DIFERENTE: "!=" >
        |   < VIRGULA: "," >
        |   < NEGLOG: "!" >
        |   < NEGBIT: "~" >
        |   < COND: "?" >
        |   < DOISPTS: ":" >
        |   < PONTO: "." >
        |   < E: "&&" >
        |   < OU: "||" >
        |   < INC: "++" >
        |   < DEC: "--" >
        |   < SOM: "+" >
        |   < SUB: "-" >
        |   < MUL: "*" >
        |   < DIV: "/" >
        |   < INTDIV: "div">
        |   < ES: "&" >
        |   < OUS: "|" >
        |   < XOU: "^" >
        |   < RES: "%" >
        |   < MOVESQ: "<<" >
        |   < MOVDIR: ">>" >
        |   < XMOVDIR: ">>>" >
        |   < SOMATR: "+=" >
        |   < SUBATR: "-=" >
        |   < MULATR: "*=" >
        |   < DIVATR: "/=" >
        |   < EATR: "&=" >
        |   < OUATR: "|=" >
        |   < XOUATR: "^=" >
        |   < RESATR: "%=" >
```

```
        |   < MOVESQATR: "<<=" >
        |   < MOVDIRATR: ">>=" >
        |   < XMOVDIRATR: ">>>=" >
        |   < INTDIVATR: "div=" >
        |   < PE: "(" >
        |   < PD: ")" >
        |   < CE: "{" >
        |   < CD: "}" >
        |   < PV: ";" >
        |   < TRALHA: "#" >
    }

/* LITERAIS */
TOKEN :
    {
           < BOOLEANLITERAL: "true" | "false" >

        |   < NUMERICLITERAL:  //numeros decimais, octais, hexadecimal
            (
                (["0"-"9"] (["0"-"9"])*) //decimal
              | (["0"-"7"] (["0"-"7"])* ["o","0"]) //octal
              | (["0"-"9"] (["0"-"7","A"-"F","a"-"f"])* ["h","H"])
              | ( ["+","-"] (["0"-"9"] (["0"-"9"])*) ) //decimal
            )
            >
        |   < FLOATLITERAL:   ((["0"-"9"])* "." (["0"-"9"])*)>//flutuante
        |   < STRINGLITERAL:
            (
                 ("\""(~["\"","\n","\r"])* "\"")
              |  ("\'"(~["\'","\n","\r"])* "\'")
             )
            >
    }


/* IDENTIFICADORES */
TOKEN :
    {
       < IDENTIFIER: <LETTER> (<LETTER>|<DIGIT>)* >
         |
```

```
        < #LETTER: ["_","a"-"z","A"-"Z"] >
          |
        < #DIGIT: ["0"-"9"] >
    }

/*****************************************************************************
*****************************************************************************
* A SEGUIR APRESENTAMOS A ESPECIFICAÇÃO SINTÁTICA DA GRAMÁTICA DO WMLSCRIPT
* ANTES DE CADA MÉTODO APRESENTAMOS EM BNF A(S) REGRA(S) EQUIVALENTE
*****************************************************************************
*****************************************************************************/

/*****************************************************************************
* <compilationUnit> ::=
*       [<pragmas>] <functionDeclarations>
*
* <pragmas> ::=
*       <pragma> |
*       <pragmas> <pragma>
*
* <functionDeclarations> ::=
*       <functionDeclaration> |
*       <functionDeclarations> <functionDeclaration>
*****************************************************************************/
CompilationUnit compilationunit() :
    {
        Collection pragmas = new ArrayList();
        Collection functiondeclarations = new ArrayList();
        Pragma pragma = null;
        FunctionDeclaration functiondeclaration = null;
    }
        {
            ( pragma = pragma() { pragmas.add(pragma); } )*
                    (    functiondeclaration    =    functiondeclaration()    {
                    functiondeclarations.add(functiondeclaration); } )+
            {
                CompilationUnit c  =  new            CompilationUnit(pragmas,
                functiondeclarations,functiondeclaration.getBeginLine(),
                functiondeclaration.getBeginColumn(),functiondeclaration.getE
                ndLine(),functiondeclaration.getEndColumn())
```

```
            c.setNodeParents();
            return c;
        }
    }

/*****************************************************************************
* <pragma> ::=
*      “use” <pragmadeclaration>
*****************************************************************************/
Pragma pragma() :
    {
        PragmaDeclaration pragmadeclaration = null;
        Token t;
    }
        {
            t = <USE> pragmadeclaration = pragmadeclaration()
            {
                Pragma                p                =                new
                Pragma(pragmadeclaration,t.beginLine,t.beginColumn,pragmadecl
                aration.getEndLine(),pragmadeclaration.getEndColumn());
                p.setNodeParents();
                return p;
            }
        }

/*****************************************************************************
* <pragmadeclaration> ::=
*        <externalcompilationunitpragma>
*      | <accesscontrolpragma>
*      | <metapragma>
*****************************************************************************/
PragmaDeclaration pragmadeclaration() :
    {
             ExternalCompilationUnitPragma  externalcompilationunitpragma  =
             null;
             AccessControlPragma accesscontrolpragma = null;
             MetaPragma metapragma = null;
    }
        {
                  (
```

```
                    externalcompilationunitpragma                    =
                    externalcompilationunitpragma()
                    {PragmaDeclaration          p          =          new
                    PragmaDeclaration(externalcompilationunitpragma,null
                    ,
                    null,externalcompilationunitpragma.getBeginLine(),ex
                    ternalcompilationunitpragma.getBeginColumn(),externa
                    lcompilationunitpragma.getEndLine(),externalcompilat
                    ionunitpragma.getEndColumn());
                    p.setNodeParents();
                    return p;}
                |   accesscontrolpragma = accesscontrolpragma()
                    {PragmaDeclaration          p1          =          new
                    PragmaDeclaration(null,accesscontrolpragma,
                    null,accesscontrolpragma.getBeginLine(),accesscontro
                    lpragma.getBeginColumn(),accesscontrolpragma.getEndL
                    ine(),accesscontrolpragma.getEndColumn());
                    p1.setNodeParents();
                    return p1;}
                |   metapragma = metapragma()
                    {PragmaDeclaration          p2          =          new
                    PragmaDeclaration(null,null,
                    metapragma,metapragma.getBeginLine(),metapragma.getB
                    eginColumn(),metapragma.getEndLine(),metapragma.getE
                    ndColumn());
                    p2.setNodeParents();
                    return p2;
            }
        )
    }

/*****************************************************************************
* <metapragma> ::=
*        “meta” <metaspecifier>
*****************************************************************************/
MetaPragma metapragma() :
    {
        MetaSpecifier metaspecifier = null;
        Token t;
    }
```

```
        {
            t = <META> metaspecifier = metaspecifier()
            {
                        MetaPragma   mp   =   new   MetaPragma(metaspecifier,
                        t.beginLine,
                        t.beginColumn,metaspecifier.getEndLine(),metaspecifi
                        er.getEndColumn());
                        mp.setNodeParents();
                        return mp;
            }
        }

/*************************************************************************
* <metaspecifier> ::=
*           <metaname>
*        |  <metahttpequiv>
*        |  <metauseragent>
*************************************************************************/
MetaSpecifier metaspecifier() :
    {
        MetaName metaname = null;
        MetaHttpEquiv metahttpequiv = null;
        MetaUserAgent metauseragent = null;
    }
        {
            (
                    metaname = metaname()
                                {MetaSpecifier         ms         =         new
                                MetaSpecifier(metaname,null,
                                null,metaname.getBeginLine(),metaname.getBegin
                                Column(),metaname.getEndLine(),metaname.getEnd
                                Column()); ms.setNodeParents();
                                return ms;}
                              |    metahttpequiv = metahttpequiv()
                                {MetaSpecifier      ms1      =            new
                                MetaSpecifier(null,metahttpequiv,
                                null,metahttpequiv.getBeginLine(),metahttpequi
                                v.getBeginColumn(),metahttpequiv.getEndLine(),
                                metahttpequiv.getEndColumn());
                                ms1.setNodeParents();
```

```
                        return ms1;}
                    |   metauseragent = metauseragent()
                        {MetaSpecifier ms2 = new
                        MetaSpecifier(null,null,
                        metauseragent,metauseragent.getBeginLine(),met
                        auseragent.getBeginColumn(),metauseragent.getE
                        ndLine(),metauseragent.getEndColumn());
                        ms2.setNodeParents();
                        return ms2;}
            )
        }

/************************************************************************
* <metaname> ::=
*       "name" <metabody>
************************************************************************/
MetaName metaname() :
    {
        MetaBody metabody = null;
        Token t;
    }
        {
            t = <NAME> metabody = metabody()
            {
                MetaName m = new MetaName(metabody, t.beginLine,
                t.beginColumn,metabody.getEndLine(),metabody.getEndColumn());
                m.setNodeParents();
                return m;
            }
        }

/************************************************************************
* <metahttpequiv> ::=
*       "http" "equiv" <metabody>
************************************************************************/
MetaHttpEquiv metahttpequiv() :
    {
        MetaBody metabody = null;
        Token t;
    }
```

```
        {
            t = <HTTP> <EQUIV> metabody = metabody()
            {
                MetaHttpEquiv m = new MetaHttpEquiv(metabody, t.beginLine,
                t.beginColumn,metabody.getEndLine(),metabody.getEndColumn());
                m.setNodeParents();
                return m;
            }
        }

/************************************************************************
* <metauseragent> ::=
*       “user” “agent” <metabody>
************************************************************************/
MetaUserAgent metauseragent() :
    {
        MetaBody metabody = null;
        Token t;
    }
        {
            t = <USER> <AGENT> metabody = metabody()
            {
                        MetaUserAgent m = new MetaUserAgent(metabody,
                        t.beginLine,
                        t.beginColumn,metabody.getEndLine(),metabody.getEndC
                        olumn());
                        m.setNodeParents();
                         return m;
            }
        }

/************************************************************************
* <metabody> ::=
*       <metapropertyname> <metacontent> [<metascheme>]
************************************************************************/
MetaBody metabody() :
    {
        MetaPropertyName metapropertyname = null;
        MetaContent metacontent = null;
        MetaScheme metascheme = null;
```

```
    }
        {
                    metapropertyname   =   metapropertyname()   metacontent   =
                    metacontent() ( metascheme = metascheme() )?
                    {
                          MetaBody m = (metascheme==null)?
                          new     MetaBody(metapropertyname,     metacontent,
                          metascheme,metapropertyname.getBeginLine(),metaprope
                          rtyname.getBeginColumn(),metacontent.getEndLine(),me
                          tacontent.getEndColumn()):
                          new     MetaBody(metapropertyname,     metacontent,
                          metascheme,metapropertyname.getBeginLine(),metaprope
                          rtyname.getBeginColumn(),metascheme.getEndLine(),met
                          ascheme.getEndColumn());
                          m.setNodeParents();
                          return m;
                    }
        }
/**************************************************************************
* <metapropertyname> ::=
*       “stringliteral”
**************************************************************************/
MetaPropertyName metapropertyname() :
    {
        StringLiteral stringliteral = null;
    }
        {
            stringliteral = stringliteral()
            {
                          MetaPropertyName          m          =          new
                          MetaPropertyName(stringliteral,stringliteral.getBegi
                          nLine(),stringliteral.getBeginColumn(),stringliteral
                          .getEndLine(), stringliteral.getEndColumn());
                          m.setNodeParents();
                          return m;
            }
        }
/**************************************************************************
* <metacontent> ::=
*       “stringliteral”
```

```
*****************************************************************************/
MetaContent metacontent() :
    {
        StringLiteral stringliteral = null;
    }
        {
            stringliteral = stringliteral()
            {
                        MetaContent            m            =            new
                        MetaContent(stringliteral,stringliteral.getBeginLine
                        (),stringliteral.getBeginColumn(),stringliteral.getE
                        ndLine(),    stringliteral.getEndColumn());
                        m.setNodeParents();
                        return m;
            }
        }

/*****************************************************************************
* <metascheme> ::=
*       “stringliteral”
*****************************************************************************/
MetaScheme metascheme() :
    {
        StringLiteral stringliteral = null;
    }
        {
            stringliteral = stringliteral()
            {
                        MetaScheme             m            =            new
                        MetaScheme(stringliteral,stringliteral.getBeginLine(
                        ),stringliteral.getBeginColumn(),stringliteral.getEn
                        dLine(),stringliteral.getEndColumn());
                        m.setNodeParents();
                        return m;
            }
        }

/*****************************************************************************
* <metascheme> ::=
*       “access” <accesscontrolspecifier>
```

```
*****************************************************************************/
AccessControlPragma accesscontrolpragma() :
{
    AccessControlSpecifier accesscontrolspecifier = null;
    Token t;
}
    {
        t = <ACCESS> accesscontrolspecifier = accesscontrolspecifier()

        {
                    AccessControlPragma           a           =           new
                    AccessControlPragma(accesscontrolspecifier,   t.beginLine,
                    t.beginColumn,
                    accesscontrolspecifier.getEndLine(),accesscontrolspecifie
                    r.getEndColumn());
                    a.setNodeParents();
                    return a;
        }
    }

/*****************************************************************************
* <accesscontrolspecifier> ::=
*        “domain” “stringliteral”
*      | “path” “stringliteral”
*      | “domain” “stringliteral” “path” “stringliteral”
*
*****************************************************************************/
AccessControlSpecifier accesscontrolspecifier() :
    {
        StringLiteral stringliteral = null;
        StringLiteral stringliteral1 = null;
        Token t;
    }
        {
            (
                                t = <DOMAIN> stringliteral = stringliteral() (
                                <PATH> stringliteral1 = stringliteral() )?
                           |   t = <PATH> stringliteral = stringliteral()
            )
            {
```

```
                    AccessControlSpecifier a = (stringliteral1==null)?
                    new           AccessControlSpecifier(stringliteral,
                    stringliteral1,                        t.beginLine,
                    t.beginColumn,stringliteral.getEndLine(),stringliter
                    al.getEndColumn()):
                    new           AccessControlSpecifier(stringliteral,
                    stringliteral1,                        t.beginLine,
                    t.beginColumn,stringliteral1.getEndLine(),stringlite
                    ral1.getEndColumn());
                    a.setNodeParents();
                    return a;
          }
       }


/*************************************************************************
* <externalcompilationunitpragma> ::=
*        “url” “identifier” “stringliteral”
*************************************************************************/
ExternalCompilationUnitPragma externalcompilationunitpragma() :
   {
       Identifier identifier = null;
       StringLiteral stringliteral = null;
       Token t;
   }
       {
                t  =  <URL>  identifier  =  identifier()  stringliteral  =
                stringliteral()

          {
                    ExternalCompilationUnitPragma      e      =      new
                    ExternalCompilationUnitPragma(identifier,stringliter
                    al,           t.beginLine,           t.beginColumn,
                    stringliteral.getEndLine(),
                    stringliteral.getEndColumn());
                    e.setNodeParents();
                    return e;
          }
       }
```

```
StringLiteral stringliteral() :
    {
        Token t;
    }
        {
            t = <STRINGLITERAL>
            {
                        return   new   StringLiteral(t.image,   t.beginLine,
                        t.beginColumn, t.endLine, t.endColumn);
            }
        }

IntegerLiteral integerliteral() :
    {
        Token t;
    }
        {
            t = <NUMERICLITERAL>

            {
                        return   new   IntegerLiteral(t.image,   t.beginLine,
                        t.beginColumn, t.endLine, t.endColumn);
            }
        }

FloatLiteral floatliteral() :
    {
        Token t;
    }
        {
            t = <FLOATLITERAL>
            {
                        return   new   FloatLiteral(t.image,   t.beginLine,
                        t.beginColumn, t.endLine, t.endColumn);
            }
        }
BooleanLiteral booleanliteral() :
    {
        Token t;
```

```
    }
        {
            t = <BOOLEANLITERAL>
            {
                        return   new   BooleanLiteral(t.image,   t.beginLine,
                        t.beginColumn,t.endLine,t.endColumn);
            }
        }

/*****************************************************************************
* <functiondeclaration> ::=
*       ["extern"] "function" "identifier" "(" [<formalparameterlist>] ")"
*       <block> [";"]
*
* <formalparameterlist> ::=
*           "identifier"
*        |  <formalparameterlist> "," "identifier"
*****************************************************************************/
FunctionDeclaration functiondeclaration() :
    {
        Identifier i = null;
        Identifier i2 = null;
        boolean extern=false;
        Collection formalparameterlist  = new ArrayList();
        Block b = null;
        Token t;
    }
        {
               (<EXTERN> {extern=true;} )? t = <FUNCTION> i = identifier()
               <PE>   (   i2 = identifier() { formalparameterlist.add(i2); }
               ( <DOISPTS> <IDENTIFIER> )?
                   (   <VIRGULA> i2 = identifier() (<DOISPTS> <IDENTIFIER>)?
               { formalparameterlist.add(i2); } )*
               )?  <PD>    b = block() (<PV>)?

            {
                        FunctionDeclaration  f  =  new  FunctionDeclaration(i,
                        formalparameterlist,    b,    extern,    t.beginLine,
                        t.beginColumn,b.getEndLine(),b.getEndColumn());
                        f.setNodeParents();
```

```
                        return f;
                    }
                }

/*****************************************************************************
* <block> ::=
*       "{" [<statementlist>] "}"
*
* <statementlist> ::=
*           <statement>
*         |  <statementlist> <statement>
*****************************************************************************/
Block block() :
    {
        Collection statementlist  = new ArrayList();
        AbstractStatement statement = null;
        Token t;
        Token t1;
    }
        {
                    t    =    <CE>    (    statement    =    statement()    {
                    statementlist.add(statement); } )* t1 = <CD>
            {
                        Block  b  =  new  Block(statementlist,  t.beginLine,
                        t.beginColumn, t1.endLine, t1.endColumn);
                        b.setNodeParents();
                        return b;
            }
        }

/*****************************************************************************
* <statement> ::=
*        <block>
*    |   <variablestatement>
```

```
*    |   <emptystatement>
*    |   <expressionstatement>
*    |   <ifstatement>
*    |   <iterationstatement>
*    |   <continuestatement>
*    |   <breakstatement>
*    |   <returnstatement>
*****************************************************************************/
AbstractStatement statement() :
    {
        AbstractStatement s = null;
    }
        {
            (
                    s = block()
                |   s = variablestatement()
                |   s = emptystatement()
                |   s = expressionstatement()
                |   s = ifstatement()
                |   s = iterationstatement()
                |   s = continuestatement()
                |   s = breakstatement()
                |   s = returnstatement()
            )
                {
                    return s;
                }
        }
/*****************************************************************************
* <variablestatement> ::=
*        “var” <variabledeclarationlist> “;”
*
* <variabledeclarationlist> ::=
*            <variabledeclaration>
*        |  <variabledeclarationlist> “,”<variabledeclaration>
*****************************************************************************/
VariableStatement variablestatement() :
    {
        Collection variabledeclarationlist = new ArrayList();
        VariableDeclaration variabledeclaration = null;
```

```
        Token t;
        Token t1;
    }
        {

                t = <VAR> (variabledeclaration = variabledeclaration() (
                <DOISPTS>           <IDENTIFIER>           )?           {
                variabledeclarationlist.add(variabledeclaration);    }    )
                (<VIRGULA> variabledeclaration = variabledeclaration() (
                <DOISPTS>           <IDENTIFIER>           )?           {
                variabledeclarationlist.add(variabledeclaration);  })*  t1
                = <PV>

           {
                     VariableStatement          v          =          new
                     VariableStatement(variabledeclarationlist,
                     t.beginLine,        t.beginColumn,        t1.endLine,
                     t1.endColumn);
                     v.setNodeParents();
                     return v;
           }
        }

/*****************************************************************************
* <variabledeclaration> ::=
*        “identifier” [<variableinitializer>]
*****************************************************************************/
VariableDeclaration variabledeclaration() :
    {
        Identifier i = null;
        VariableInitializer variableinitializer = null;
    }
        {
                  i    =    identifier()    (    variableinitializer    =
                  variableinitializer() )?

           {
                     VariableDeclaration v;
                     if (variableinitializer == null){
```

```
                        v        =        new        VariableDeclaration(i,
                        variableinitializer,i.getBeginLine(),i.getBeginColum
                        n(),i.getEndLine(),i.getEndColumn());
                }else{
                       v        =        new        VariableDeclaration(i,
                       variableinitializer,i.getBeginLine(),i.getBeginColumn(
                       ),variableinitializer.getEndLine(),variableinitializer
                       .getEndColumn() );
                }
                v.setNodeParents();
                return v;
            }
        }

Identifier identifier() :
    {
        Token i;
    }
        {
            i = <IDENTIFIER>
            {
                        return    new    Identifier(i.image,    i.beginLine,
                        i.beginColumn,i.endLine,i.endColumn);
            }
        }

/*************************************************************************
* <variableinitializer> ::=
*        "=" <conditionalexpression>
*************************************************************************/
VariableInitializer variableinitializer() :
    {
        AbstractExpression conditionalexpression = null;
        Token t;
    }
        {
            t = <ATR> conditionalexpression = conditionalexpression()

            {
```

```
                    VariableInitializer v = new
                    VariableInitializer("=",conditionalexpression,
                    t.beginLine,
                    t.beginColumn,conditionalexpression.getEndLine(),
                    conditionalexpression.getEndColumn());
                    v.setNodeParents();
                    return v;
            }
        }

/*****************************************************************************
* <conditionalexpression> ::=
*         <logicalorexpression>
*      |  <logicalorexpression> "?" <assignmentexpression>
*         ":" <assignmentexpression>
*****************************************************************************/
AbstractExpression conditionalexpression() :
    {
        AbstractExpression logicalorexpression = null;
        AbstractExpression assignmentexpression = null;
        AbstractExpression assignmentexpression1 = null;
    }
        {
                logicalorexpression = logicalorexpression()
                (<COND>  assignmentexpression  =  assignmentexpression()
                <DOISPTS>  assignmentexpression1  =  assignmentexpression()
                )?

            {
                    ConditionalExpression c;
                    if((assignmentexpression1==null)
                    &&(assignmentexpression==null)){
                    return logicalorexpression;
            }else{
                    c  =  new  ConditionalExpression(logicalorexpression,
                    assignmentexpression,
                    assignmentexpression1,logicalorexpression.getBeginLine
                    (),logicalorexpression.getBeginColumn(),assignmentexpr
                    ession1.getEndLine(),assignmentexpression1.getEndColum
                    n());
```

```
                    c.setNodeParents();
                    return c;
                }
            }
        }

/*****************************************************************************
* <logicalorexpression> ::=
*         <logicalandexpression>
*      |  <logicalorexpression> “||” <logicalandexpression>
*****************************************************************************/
AbstractExpression logicalorexpression() :
    {
        AbstractExpression  logicalandexpression = null;
        AbstractExpression  logicalorexpression = null;
    }
        {
            logicalandexpression    =    logicalandexpression()    (    <OU>
            logicalorexpression = logicalorexpression() )?
            {
                LogicalORExpression l;
                if (logicalorexpression==null){
                    return logicalandexpression;
                }else{
                    l    =    new    LogicalORExpression(logicalandexpression,
                    logicalorexpression,"||",logicalandexpression.getBeginLin
                    e(),logicalandexpression.getBeginColumn(),
                    logicalorexpression.getEndLine(),
                    logicalorexpression.getEndColumn());
                    l.setNodeParents();
                    return l;
                }
            }
        }



/*****************************************************************************
* <logicalandexpression> ::=
```

```
*         <bitwiseorexpression>
*       |  <logicalandexpression> “&&” <bitwiseorexpression>
*****************************************************************************/
AbstractExpression logicalandexpression() :
    {
        AbstractExpression bitwiseorexpression = null;
        AbstractExpression logicalandexpression = null;
    }
        {
                  bitwiseorexpression   =   bitwiseorexpression()   (   <E>
                  logicalandexpression = logicalandexpression() )?
           {
               LogicalANDExpression l;
               if  (logicalandexpression==null){
                   return bitwiseorexpression;
               }else{
                             l                    =                    new
                             LogicalANDExpression(bitwiseorexpression,
                             logicalandexpression,                   "&&",
                             bitwiseorexpression.getBeginLine(),
                             bitwiseorexpression.getBeginColumn(),
                             logicalandexpression.getEndLine(),
                             logicalandexpression.getEndColumn());
                             l.setNodeParents();
                             return l;
               }
           }
       }
/*****************************************************************************
* <bitwiseorexpression> ::=
*         <bitwisexorexpression>
*       |  <bitwiseorexpression> “|” <bitwisexorexpression>
*****************************************************************************/
AbstractExpression bitwiseorexpression() :
    {
        AbstractExpression bitwisexorexpression = null;
        AbstractExpression bitwiseorexpression = null;
    }
        {
```

```
                bitwisexorexpression   =   bitwisexorexpression()   (<OUS>
                bitwiseorexpression = bitwiseorexpression() )?

                {
                BitwiseORExpression b;
                if (bitwiseorexpression==null){
                    return bitwisexorexpression;
                }else{
                        b  =  new  BitwiseORExpression(bitwisexorexpression,
                    bitwiseorexpression,"|",bitwisexorexpression.getBeginL
                    ine(),
                        bitwisexorexpression.getBeginColumn(),bitwiseorexp
                    ression.getEndLine(),bitwiseorexpression.getEndColumn(
                    ));
                    b.setNodeParents();
                    return b;
                }
            }
        }

/************************************************************************
* <bitwisexorexpression> ::=
*         <bitwiseandexpression>
*      |  <bitwisexorexpression> "^" <bitwiseandexpression>
************************************************************************/
AbstractExpression bitwisexorexpression() :
    {
        AbstractExpression bitwiseandexpression = null;
        AbstractExpression bitwisexorexpression = null;
    }
        {
                bitwiseandexpression  =  bitwiseandexpression()  (  <XOU>
                bitwisexorexpression = bitwisexorexpression() )?
                {
                    BitwiseXORExpression b;
                    if (bitwisexorexpression==null){
                        return bitwiseandexpression;
                    }else{
                        b = new BitwiseXORExpression(bitwiseandexpression,
                        bitwisexorexpression,"^",bitwiseandexpression.getB
```

```
                        eginLine(),bitwiseandexpression.getBeginColumn(),
                        bitwisexorexpression.getEndLine(),
                        bitwisexorexpression.getEndColumn());
                        b.setNodeParents();
                    return b;
                }
            }
        }

/************************************************************************
* <bitwiseandexpression> ::=
*         <equalityexpression>
*      |  <bitwiseandexpression> “&” <equalityexpression>
************************************************************************/
AbstractExpression bitwiseandexpression() :
    {
        AbstractExpression equalityexpression = null;
        AbstractExpression bitwiseandexpression = null;
    }
        {
            equalityexpression      =      equalityexpression()      (<ES>
            bitwiseandexpression = bitwiseandexpression() )?
            {
                BitwiseANDExpression b;
                if (bitwiseandexpression==null){
                    return equalityexpression;
                }else{
                    b   =   new   BitwiseANDExpression(equalityexpression,
                    bitwiseandexpression,                              "&",
                    equalityexpression.getBeginLine(),equalityexpression.g
                    etBeginColumn(),    bitwiseandexpression.getEndLine(),
                    bitwiseandexpression.getEndColumn());
                    b.setNodeParents();
                    return b;
                }
            }
        }

EqualityOperator equalityoperator() :
    {
```

```
    }
        {
                    <IGUAL> { return new EqualityOperator("=="); }
                |   <DIFERENTE> { return new EqualityOperator("!=");
        }

/*****************************************************************************
* <equalityexpression> ::=
*         <relationalexpression>
*      |  <equalityexpression> “==” <relationalexpression>
*      |  <equalityexpression> “!=” <relationalexpression>
*****************************************************************************/
AbstractExpression equalityexpression() :
    {
        AbstractExpression relationalexpression = null;
        AbstractExpression equalityexpression = null;
        EqualityOperator equalityoperator = null;
    }
        {
            relationalexpression = relationalexpression() ( equalityoperator
            = equalityoperator()   equalityexpression = equalityexpression()
            )?
            {
                EqualityExpression e;
                if (equalityexpression == null){
                    return relationalexpression;
                }else{
                        e   =   new   EqualityExpression(relationalexpression,
                    equalityexpression,                     equalityoperator,
                    relationalexpression.getBeginLine(),relationalexpression.
                    getBeginColumn(),equalityexpression.getEndLine(),equality
                    expression.getEndColumn());
                    e.setNodeParents();
                    return e;
                }
            }
        }

RelationalOperator relationaloperator() :
    {
```

```
    }
        {
                <MENOR> {return new RelationalOperator("<");}
            |   <MAIOR> {return new RelationalOperator(">");}
            |   <MENORIGUAL> {return new RelationalOperator("<=");}
            |   <MAIORIGUAL> {return new RelationalOperator(">=");}
        }

/*****************************************************************************
* <relationalexpression> ::=
*         <shiftexpression>
*      |  <relationalexpression> “<”  <shiftexpression>
*      |  <relationalexpression> “>” <shiftexpression>
*      |  <relationalexpression> “<=” <shiftexpression>
*      |  <relationalexpression> “>=” <shiftexpression>
*****************************************************************************/
AbstractExpression relationalexpression() :
    {
        AbstractExpression relationalexpression = null;
        AbstractExpression shiftexpression = null;
        RelationalOperator relationaloperator = null;
    }
        {
            shiftexpression  =  shiftexpression()  (  relationaloperator  =
            relationaloperator()            relationalexpression            =
            relationalexpression() )?
            {

                RelationalExpression r;
                if (relationalexpression == null){
                    return shiftexpression;
                }else{
                    r     =     new     RelationalExpression(shiftexpression,
                    relationalexpression,                  relationaloperator,
                    shiftexpression.getBeginLine(),shiftexpression.getBeginCo
                    lumn(),                relationalexpression.getEndLine(),
                    relationalexpression.getEndColumn() );
                    r.setNodeParents();
                    return r;
                }
```

```
            }
        }

ShiftOperator shiftoperator() :
    {
    }
        {
                <MOVESQ> { return new ShiftOperator("<<"); }
            |   <MOVDIR> { return new ShiftOperator(">>"); }
            |   <XMOVDIR> { return new ShiftOperator(">>>"); }
        }

/**************************************************************************
* <shiftexpression> ::=
*         <additiveexpression>
*      |  <shiftexpression>  “<<” <additiveexpression>
*      |  <shiftexpression> “>>” <additiveexpression>
*      |  <shiftexpression> “>>>” <additiveexpression>
**************************************************************************/
AbstractExpression shiftexpression() :
    {
        AbstractExpression additiveexpression = null;
        AbstractExpression shiftexpression = null;
        ShiftOperator shiftoperator = null;
    }
        {
            additiveexpression  =  additiveexpression()  (  shiftoperator  =
            shiftoperator() shiftexpression = shiftexpression() )?
            {
                ShiftExpression s;
                if (shiftexpression==null){
                    return additiveexpression;
                }else{
                    s    =         new    ShiftExpression(additiveexpression,
                    shiftexpression,
                    shiftoperator,additiveexpression.getBeginLine(),additivee
                    xpression.getBeginColumn(),shiftexpression.getEndLine(),s
                    hiftexpression.getEndColumn());
                    s.setNodeParents();
                    return s;
```

```
            }
        }
    }

AdditiveOperator additiveoperator() :
    {
    }
        {
                <SOM> { return new AdditiveOperator("+"); }
            |   <SUB> { return new AdditiveOperator("-"); }
        }

/*****************************************************************************
*  <additiveexpression> ::=
*         <multiplicativeexpression>
*      |  <additiveexpression>  “+” <multiplicativeexpression>
*      |  <additiveexpression> “-” <multiplicativeexpression>
*****************************************************************************/
AbstractExpression additiveexpression() :
    {
        AbstractExpression multiplicativeexpression = null;
        AbstractExpression multiplicativeexpression2 = null;
        AdditiveOperator additiveoperator = null;
    }
        {
                multiplicativeexpression   =   multiplicativeexpression()   (
                additiveoperator             =             additiveoperator()
                multiplicativeexpression2 = multiplicativeexpression() )*
                {
                    AdditiveExpression a;
                    if (multiplicativeexpression2 == null){
                    return multiplicativeexpression;
                }else{
                    a   =   new   AdditiveExpression(multiplicativeexpression,
                    multiplicativeexpression2,
                    additiveoperator,multiplicativeexpression.getBeginLine(),
                    multiplicativeexpression.getBeginColumn(),multiplicativee
                    xpression2.getEndLine(),multiplicativeexpression2.getEndC
                    olumn());
                    a.setNodeParents();
```

```
                return a;
            }
        }
    }

MultiplicativeOperator multiplicativeoperator() :
    {
    }
        {
            <MUL> { return new MultiplicativeOperator("*");}
        |   <DIV> { return new MultiplicativeOperator("/");}
        |   <INTDIV> { return new MultiplicativeOperator("div");}
        |   <RES> { return new MultiplicativeOperator("%");}
        }

/*************************************************************************
*  <multiplicativeexpression> ::=
*         <unaryexpression>
*      |  <multiplicativeexpression> “*” <unaryexpression>
*      |  <multiplicativeexpression> “/” <unaryexpression>
*      |  <multiplicativeexpression> “div” <unaryexpression>
*      |  <multiplicativeexpression> “%” <unaryexpression>
*************************************************************************/
AbstractExpression multiplicativeexpression() :
    {
        AbstractExpression unaryexpression = null;
        AbstractExpression multiplicativeexpression = null;
        MultiplicativeOperator multiplicativeoperator = null;
        int el;
        int ec;
    }
        {
            unaryexpression = unaryexpression() ( multiplicativeoperator =
            multiplicativeoperator()       multiplicativeexpression       =
            multiplicativeexpression() )?
            {
                MultiplicativeExpression m;
                if (multiplicativeexpression == null){
                 return unaryexpression;
                }else{
```

```
                el = multiplicativeexpression.getEndLine();
                ec = multiplicativeexpression.getEndColumn();
                    m   =   new   MultiplicativeExpression(unaryexpression,
                multiplicativeexpression,
                multiplicativeoperator,unaryexpression.getBeginLine(),
                unaryexpression.getBeginColumn(), el, ec);
                m.setNodeParents();
                return m;
            }
         }
      }

/*************************************************************************
*  <unaryexpression> ::=
*         <postfixexpression>
*      |  “typeof” <unaryexpression>
*      |  “isvalid” <unaryexpression>
*      |  “++” <unaryexpression>
*      |  “--” <unaryexpression>
*      |  “+” <unaryexpression>
*      |  “-” <unaryexpression>
*      |  “~” <unaryexpression>
*      |  “!” <unaryexpression>
*************************************************************************/
AbstractExpression unaryexpression() :
   {
      AbstractExpression postfixexpression = null;
      AbstractExpression unaryexpression = null;
      Identifier identifier = null;
      String op = null;
      Token t = null;
      int el = 0;
      int ec = 0;
   }
      {
         (
                postfixexpression = postfixexpression()
             | t = <TYPEOF> unaryexpression = unaryexpression()  { op =
               "typeof";  el  =  unaryexpression.getEndLine();  ec  =
               unaryexpression.getEndColumn();}
```

```
            | t = <ISVALID> unaryexpression = unaryexpression() { op =
              "isvalid"; el = unaryexpression.getEndLine(); ec =
              unaryexpression.getEndColumn();}
            | t = <INC> identifier = identifier() { op = "++"; el =
              identifier.getEndLine(); ec = identifier.getEndColumn();}
            | t = <DEC> identifier = identifier()       { op = "--";
              el      =      identifier.getEndLine();      ec      =
              identifier.getEndColumn();}
          |  t = <SOM> unaryexpression = unaryexpression() { op = "+";
              el     =     unaryexpression.getEndLine();     ec     =
              unaryexpression.getEndColumn();}
          |  t = <SUB> unaryexpression = unaryexpression() { op = "-";
              el     =     unaryexpression.getEndLine();     ec     =
              unaryexpression.getEndColumn();}
          |  t = <NEGBIT> unaryexpression = unaryexpression() { op =
              "~";   el   =   unaryexpression.getEndLine();   ec   =
              unaryexpression.getEndColumn();}
          |  t = <NEGLOG> unaryexpression = unaryexpression() { op =
              "!";   el   =   unaryexpression.getEndLine();   ec   =
              unaryexpression.getEndColumn(); }
        )
        {
            UnaryExpression u;
            if (postfixexpression == null){
                u                          =                          new
                UnaryExpression(postfixexpression,unaryexpression,identif
                ier,op, t.beginLine, t.beginColumn, el, ec);
                u.setNodeParents();
                return u;
            }else{
                return postfixexpression;
            }
        }
    }


PostfixExpression postfixnocallexpression() :
    {
        Identifier identifier = null;
```

```
        String op = null;
        Token t;
    }
        {
            identifier = identifier()
            (
                t = <INC> { op = "++";}
            |   t = <DEC> { op = "--";}
            )

            {
                PostfixExpression p = new PostfixExpression(null,identifier,
                op,identifier.getBeginLine(),identifier.getBeginColumn(),t.en
                dLine,t.endColumn);
                p.setNodeParents();
                return p;
            }
}

/*****************************************************************************
*  <postfixexpression> ::=
*         <callexpression>
*      |  “identifier” “++”
*      |  “identifier” “--”
*****************************************************************************/
AbstractExpression postfixexpression() :
    {
        AbstractExpression callexpression = null;
        PostfixExpression postfixnocallexpression = null;
    }
        {
            (
                    LOOKAHEAD(postfixnocallexpression())
                    postfixnocallexpression = postfixnocallexpression()
                    {
                        return postfixnocallexpression;
                    }
                |
                    callexpression = callexpression()
                    {
```

```
                    return callexpression;
                }
            )
        }

/*****************************************************************************
*  <callexpression> ::=
*         <primaryexpression>
*      |  <localscriptfunctioncall>
*      |  <externalscriptfunctioncall>
*      |  <libraryfunctioncall>
*****************************************************************************/
AbstractExpression callexpression() :
    {
        PrimaryExpression primaryexpression = null;
        LocalScriptFunctionCall localscriptfunctioncall = null;
        ExternalScriptFunctionCall externalscriptfunctioncall = null;
        LibraryFunctionCall libraryfunctioncall = null;
    }
        {
            (
                    LOOKAHEAD(2)
                    libraryfunctioncall   =   libraryfunctioncall()   {return
                    libraryfunctioncall;}
                |
                    LOOKAHEAD(2)
                    localscriptfunctioncall    =    localscriptfunctioncall()
                    {return localscriptfunctioncall;}
                |
                    LOOKAHEAD(2)
                    externalscriptfunctioncall = externalscriptfunctioncall()
                    {return externalscriptfunctioncall;}
                |
                    primaryexpression   =   primaryexpression()   {   return
                    primaryexpression; }
            )
        }
/*****************************************************************************
*  <primaryexpression> ::=
*         “identifier”
```

```
*      |  “literal”
*      |  “(” <expression> “)”
*****************************************************************************/
PrimaryExpression primaryexpression() :
    {
        Identifier identifier = null;
        AbstractLiteral literal = null;
        Expression expression = null;
        int beginLine;
        int beginColumn;
        int endLine;
        int endColumn;
        Token t;
        Token t1;
    }
        {
            (
                identifier = identifier()
                {
                    beginLine = identifier.getBeginLine();
                    beginColumn = identifier.getBeginColumn();
                    endLine = identifier.getEndLine();
                    endColumn = identifier.getEndColumn();
                }
                |
                (
                    literal = stringliteral()
                |   literal = integerliteral()
                |   literal = floatliteral()
                |   literal = booleanliteral()
                )
                    {
                        beginLine = literal.getBeginLine();
                        beginColumn = literal.getBeginColumn();
                        endLine = literal.getEndLine();
                        endColumn = literal.getEndColumn();
                    }

                |   t = <PE> expression = expression() t1 = <PD>
                    {
```

```
                    beginLine = t.beginLine;
                    beginColumn = t.beginColumn;
                    endLine = t1.endLine;
                    endColumn = t1.endColumn;
                }
            )
            {
                PrimaryExpression p = new PrimaryExpression(identifier,
                literal, expression, beginLine, beginColumn, endLine,
                endColumn);
                p.setNodeParents();
                return p;
            }
        }

/*****************************************************************************
*  <localscriptfunctioncall> ::=
*       <functionname> <arguments>
*****************************************************************************/
LocalScriptFunctionCall localscriptfunctioncall() :
    {
        FunctionName functionname = null;
        Arguments arguments = null;
    }
        {
            functionname = functionname() arguments = arguments()
            {
                LocalScriptFunctionCall l = new
                LocalScriptFunctionCall(functionname,
                arguments,functionname.getBeginLine(),functionname.getBeginCo
                lumn(),arguments.getEndLine(),arguments.getEndColumn());
                l.setNodeParents();
                return l;
            }
        }


/*****************************************************************************
*  <externalscriptfunctioncall> ::=
*       <externalscriptname> “#” <functionname> <arguments>
```

```
*****************************************************************************/
ExternalScriptFunctionCall externalscriptfunctioncall() :
    {
        ExternalScriptName externalscriptname = null;
        FunctionName functionname = null;
        Arguments arguments = null;
    }
        {
            externalscriptname = externalscriptname() <TRALHA> functionname
            = functionname() arguments = arguments()
            {
                ExternalScriptFunctionCall          e          =          new
                ExternalScriptFunctionCall(externalscriptname,  functionname,
                arguments,                 externalscriptname.getBeginLine(),
                externalscriptname.getBeginColumn(),arguments.getEndLine(),ar
                guments.getEndColumn());
                e.setNodeParents();
                return e;
            }
        }

/*****************************************************************************
*  <libraryfunctioncall> ::=
*      <libraryname> “.” <functionname> <arguments>
*****************************************************************************/
LibraryFunctionCall libraryfunctioncall() :
    {
        LibraryName libraryname = null;
        FunctionName functionname = null;
        Arguments arguments = null;
    }
        {
                    libraryname   =   libraryname()   <PONTO>   functionname   =
                    functionname() arguments = arguments()
            {
                LibraryFunctionCall f = new LibraryFunctionCall( libraryname,
                functionname,
                arguments,libraryname.getBeginLine(),libraryname.getBeginColu
                mn(),
                arguments.getEndLine(),arguments.getEndColumn() );
```

```
            f.setNodeParents();
            return f;
        }
    }

/*****************************************************************************
*  <functionname> ::=
*      "identifier"
*****************************************************************************/
FunctionName functionname() :
    {
        Identifier identifier = null;
    }
        {
            identifier = identifier()
            {
                FunctionName f = new
                FunctionName(identifier,identifier.getBeginLine(),identifier.
                getBeginColumn(),identifier.getEndLine(),identifier.getEndCol
                umn());
                f.setNodeParents();
                return f;
            }
        }

/*****************************************************************************
*  <libraryname> ::=
*      "identifier"
*****************************************************************************/
LibraryName libraryname() :
    {
        Identifier identifier = null;
    }
        {
            identifier = identifier()
            {
                LibraryName l = new
                LibraryName(identifier,identifier.getBeginLine(),identifier.g
                etBeginColumn(),identifier.getEndLine(),identifier.getEndColu
                mn());
```

```
            l.setNodeParents();
            return l;
        }
    }

/*****************************************************************************
*  <externalscriptname> ::=
*      "identifier"
*****************************************************************************/
ExternalScriptName externalscriptname() :
    {
        Identifier identifier = null;
    }
        {
            identifier = identifier()
            {
                ExternalScriptName            e            =            new
                ExternalScriptName(identifier,identifier.getBeginLine(),ident
                ifier.getBeginColumn(),identifier.getEndLine(),identifier.get
                EndColumn());
                e.setNodeParents();
                return e;
            }
        }

/*****************************************************************************
*  <arglist> ::=
*        <assignmentexpression>
*     |  <arglist> "," <assignmentexpression>
*****************************************************************************/
Collection arglist() :
    {
        AbstractExpression assignmentexpression = null;
        Collection argumentlist = new ArrayList();
    }
        {
            (assignmentexpression          =          assignmentexpression()
            {argumentlist.add(assignmentexpression);}
            (<VIRGULA>    assignmentexpression    =    assignmentexpression()
            {argumentlist.add(assignmentexpression);})* )
```

```
            {
                return argumentlist;
            }
        }

/*****************************************************************************
*  <arguments> ::=
*         “(” “)”
*     |   “(” <arglist> “)”
*****************************************************************************/
Arguments arguments() :
    {
        Collection argumentlist = null;
        Token t;
        Token t1;
    }
        {

                t = <PE>
                ( argumentlist = arglist() )?
                t1 = <PD>
                {
                    Arguments              a              =              new
                    Arguments(argumentlist,t.beginLine,t.beginColumn,t1.endLi
                    ne,t1.endColumn);
                    a.setNodeParents();
                    return a;
                }
        }

/*****************************************************************************
*  <assignmentexpression> ::=
*         <conditionalexpression>
*     |   “identifier” <assignmentoperator> <assignmentexpression>
*****************************************************************************/
AbstractExpression assignmentexpression() :
    {
        AbstractExpression conditionalexpression = null;
        Identifier identifier = null;
        AssignmentOpertator assignmentoperator = null;
```

```
    AbstractExpression assignmentexpression = null;
    int beginLine;
    int beginColumn;
    int endLine;
    int endColumn;
}
    {
        (
                LOOKAHEAD(2)identifier = identifier() assignmentoperator
                =     assignmentoperator()     assignmentexpression     =
                assignmentexpression()
                {
                    beginLine = identifier.getBeginLine();
                    beginColumn = identifier.getBeginColumn();
                    endLine = assignmentexpression.getEndLine();
                    endColumn = assignmentexpression.getEndColumn();
                }
            |   conditionalexpression = conditionalexpression()
                {
                    beginLine = conditionalexpression.getBeginLine();
                    beginColumn = conditionalexpression.getBeginColumn();
                    endLine = conditionalexpression.getEndLine();
                    endColumn = conditionalexpression.getEndColumn();
                }
        )
            {
                AssignmentExpression a;
                if (assignmentexpression == null){
                    return conditionalexpression;
                }else{
                    a                        =                        new
                    AssignmentExpression(conditionalexpression,identifier,
                    assignmentoperator,  assignmentexpression,  beginLine,
                    beginColumn, endLine, endColumn);
                    a.setNodeParents();
                    return a;
                }
            }
    }
```

```
/*****************************************************************************
*  <assignmentoperator> ::=
*         "="
*     |   "*="   |   "/="   |   "%="   |   "*="   |   "+="   |   "-="
*     |   "<<="  |   ">>="  |   ">>>=" |   "&="   |   "^="   |   "|="
*     |   "div="
*****************************************************************************/
AssignmentOpertator assignmentoperator() :
    {
    }
        {
                <ATR>          {return new AssignmentOpertator("=");}
            |   <MULATR>       {return new AssignmentOpertator("*=");}
            |   <DIVATR>       {return new AssignmentOpertator("/=");}
            |   <RESATR>       {return new AssignmentOpertator("%=");}
            |   <SOMATR>       {return new AssignmentOpertator("+=");}
            |   <SUBATR>       {return new AssignmentOpertator("-=");}
            |   <MOVESQATR>        {return new AssignmentOpertator("<<=");}
            |   <MOVDIRATR>        {return new AssignmentOpertator(">>=");}
            |   <XMOVDIRATR>   {return new AssignmentOpertator(">>>=");}
            |   <EATR>         {return new AssignmentOpertator("&=");}
            |   <XOUATR>       {return new AssignmentOpertator("^=");}
            |   <OUATR>        {return new AssignmentOpertator("|=");}
            |   <INTDIVATR>        {return new AssignmentOpertator("div=");}
        }

/*****************************************************************************
*  <expression> ::=
*         <assignmentexpression>
*     |   <expression> "," <assignmentexpression>
*****************************************************************************/
Expression expression() :
    {
        AbstractExpression assignmentexpression = null;
        Collection expressionlist = new ArrayList();
        int bl;
        int bc;
    }
        {
            ( assignmentexpression = assignmentexpression()
```

```
            {
                expressionlist.add(assignmentexpression);
                bl = assignmentexpression.getBeginLine();
                bc = assignmentexpression.getBeginColumn();
            }
        )
        (   <VIRGULA>   assignmentexpression   =   assignmentexpression()
        {expressionlist.add(assignmentexpression);} )*
        {
            Expression              e              =              new
            Expression(expressionlist,bl,bc,assignmentexpression.getEndLi
            ne(),assignmentexpression.getEndColumn());
            e.setNodeParents();
            return e;
        }
    }

/*****************************************************************************
*  <emptystatement> ::=
*       ";"
*****************************************************************************/
EmptyStatement emptystatement() :
    {
        Token t;
    }
        {
            t = <PV>
            {return                                                        new
            EmptyStatement(t.beginLine,t.beginColumn,t.endLine,t.endColumn);
            }
        }

/*****************************************************************************
*  <expressionstatement> ::=
*    <expression> ";"
*****************************************************************************/
ExpressionStatement expressionstatement() :
    {
        Expression expression = null;
        Token t;
```

```
    }
        {
            expression = expression() t = <PV>
            {
                ExpressionStatement            e            =            new
                ExpressionStatement(expression,expression.getBeginLine(),expr
                ession.getBeginColumn(),t.endLine,t.endColumn);
                e.setNodeParents();
                return e;
            }
        }

/*************************************************************************
*  <ifstatement> ::=
*    “if” “(” <expression> “)” <statement> [“else” <statement>]
*************************************************************************/
IfStatement ifstatement() :
    {
        Expression expression = null;
        AbstractStatement statement = null;
        AbstractStatement statement1 = null;
        Token t;
        int endLine;
        int endColumn;
    }
        {
                    t = <IF> <PE> expression = expression() <PD> statement =
                statement()
                [ LOOKAHEAD(1) <ELSE> statement1 = statement() ]
                        {
                        endLine                                              =
                        (statement1==null)?statement.getEndLine():statement1.g
                        etEndLine();
                        endColumn                                            =
                        (statement1==null)?statement.getEndColumn():statement1
                        .getEndColumn();
                        }
                        {
```

```
                    IfStatement i = new
                    IfStatement(expression,statement,statement1,t.beginL
                    ine, t.beginColumn,endLine,endColumn);
                    i.setNodeParents();
                    return i;
                  }
      }

/*****************************************************************************
*  <iterationstatement> ::=
*       <whilestatement>
*    |  <forstatement>
*****************************************************************************/
AbstractStatement iterationstatement() :
   {
      WhileStatement whilestatement = null;
      ForStatement forstatement = null;
   }
      {
         (
                whilestatement = whilestatement()   {              return
                whilestatement;}            |   forstatement            =
                forstatement() { return forstatement;}
         )
      }

/*****************************************************************************
*  <whilestatement> ::=
*      “while” “(“ <expression> “)” “statement”
*****************************************************************************/
WhileStatement whilestatement() :
   {
      Expression expression = null;
      AbstractStatement statement = null;
      Token t;
   }
      {
                t = <WHILE> <PE> expression = expression() <PD> statement
                = statement()
                {
```

```
                    WhileStatement w = new WhileStatement(expression,
                    statement,                                t.beginLine,
                    t.beginColumn,statement.getEndLine(),statement.getEn
                    dColumn()  );
                    w.setNodeParents();
                    return w;
          }
       }

/*************************************************************************
*  <forstatement> ::=
*  “for” “(“ [<expression>] “;” [<expression>] “;” [<expression>] “)”
*   “statement”
*   “for” “(“ “var” <variabledeclarationlist> “;” [<expression>] “;”
*   [<expression>] “)”
*   “statement”
*************************************************************************/
ForStatement forstatement() :
    {
       AbstractStatement statement = null;
       Expression expression = null;
       Expression expression1 = null;
       Expression expression2 = null;
       Collection variabledeclarationlist = new ArrayList();
       VariableDeclaration variabledeclaration = null;
       Token t;
    }
       {
              t = <FOR> <PE>
              (

                     expression = expression()
                     |  (     <VAR>     (     variabledeclaration     =
                     variabledeclaration()                            {
                     variabledeclarationlist.add(variabledeclaration);  }  )
                        (      <VIRGULA>      variabledeclaration      =
                     variabledeclaration()                            {
                     variabledeclarationlist.add(variabledeclaration); } )*
                     )
              )
```

```
                    <PV> expression1 = expression() <PV> expression2 =
                    expression() <PD> statement = statement()
            {
                ForStatement f = new ForStatement(expression,
                expression1, expression2, statement,
                variabledeclarationlist,t.beginLine, t.beginColumn,
                statement.getEndLine(),statement.getEndColumn() );
                f.setNodeParents();
                return f;
            }
        }

/*****************************************************************************
*  <continuestatement> ::=
*          “continue” “;”
*****************************************************************************/
ContinueStatement continuestatement() :
    {
        Token t;
        Token t1;
    }
        {
            t = <CONTINUE> t1 = <PV>
            {
                return new ContinueStatement(t.beginLine, t.beginColumn,
                t1.endLine, t1.endColumn );
            }
        }

/*****************************************************************************
*  <breakstatement> ::=
*          “break” “;”
*****************************************************************************/
BreakStatement breakstatement() :
    {
        Token t;
        Token t1;
    }
        {
            t = <BREAK>  t1 = <PV>
```

```
            {
                    return   new   BreakStatement(t.beginLine,   t.beginColumn,
                t1.endLine, t1.endColumn );
            }
        }

/*****************************************************************************
*  <returnstatement> ::=
*          “return” [expression] “;”
*****************************************************************************/
ReturnStatement returnstatement() :
{
    Expression expression = null;
    Token t;
    Token t1;
}
    {
        t = <RETURN> ( expression = expression() )? t1 = <PV>

        {
            ReturnStatement    r    =    new    ReturnStatement(    expression,
            t.beginLine, t.beginColumn, t1.endLine, t1.endColumn );
            r.setNodeParents();
            return r;
        }
    }
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo