Um Plugin Eclipse para COMPGEN

Trabalho de Conclusão de Curso

Engenharia da Computação

Marcelo de Andrade Silva Filho
Orientador: Prof. Joabe Jesus

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

MARCELO DE ANDRADE SILVA FILHO

UM PLUGIN ECLIPSE PARA COMPGEN

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, Julho 2019.

Marcelo de Andrade Silva Filho
Um Plugin Eclipse para COMPGEN/ Marcelo de Andrade Silva Filho. – Recife - PE, Brasil, Julho de 2019-
33 p.
Orientador: Prof. Joabe Bezerra de Jesus Júnior

Trabalho de Conclusão de Curso – Engenharia de Computação
Escola Politécnica de Pernambuco
Universidade de Pernambuco, Julho de 2019.

1. Compilador de compilador. 2. Eclipse. 3. Gerador de parsers. 4. Plugins. 5. COMPGEN. I. Prof. Joabe Bezerra de Jesus Júnior. II. Universidade de Pernambuco. III. Escola Politécnica de Pernambuco. IV. Título

Agradeço aos meus pais primeiramente, pois sem eles não conseguiria superar os desafios diários durante todos os anos do curso.

Às amizades construídas durante minha formação, que me deram suporte para que eu pudesse atingir meus objetivos.

Ao professor Joabe, me que ajudou durante todo o semestre com o trabalho de conclusão de curso, sendo paciente e atencioso em todos os quesitos.

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 4/07/2019, às 14h, reuniu-se para deliberar sobre a defesa da monografia de conclusão de curso do(a) discente **MARCELO DE ANDRADE SILVA FILHO**, orientado(a) pelo(a) professor(a) **JOABE BEZERRA DE JESUS JÚNIOR**, sob título UM PLUGIN ECLIPSE PARA COMPGEN, a banca composta pelos professores:

LUIS CARLOS DE SOUSA MENEZES (PRESIDENTE)
JOABE BEZERRA DE JESUS JÚNIOR (ORIENTADOR)

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

☒ Aprovada ☐ Aprovada com Restrições* ☐ Reprovada

e foi-lhe atribuída nota: 7,5 (sete e meio)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O(A) discente terá _____ dias para entrega da versão final da monografia a contar da data deste documento.

Luis Carlos de Sousa Menezes
AVALIADOR 1: Prof (a) **LUIS CARLOS DE SOUSA MENEZES**

Joabe B. de Jesus Jr.
AVALIADOR 2: Prof (a) **JOABE BEZERRA DE JESUS JÚNIOR**

AVALIADOR 3: Prof (a)

* Este documento deverá ser encadernado juntamente com a monografia em versão final.

Autorização de publicação de PFC

Eu, **Marcelo de Andrade Silva Filho** autor(a) do projeto de final de curso intitulado: **UM PLUGIN ECLIPSE PARA COMPGEN;** autorizo a publicação de seu conteúdo na internet nos portais da Escola Politécnica de Pernambuco e Universidade de Pernambuco.

O conteúdo do projeto de final de curso é de responsabilidade do autor.

Marcelo de Andrade Silva Filho
Marcelo de Andrade Silva Filho

Orientador(a): **Joabe Bezerra de Jesus Júnior**

Coorientador(a):

Prof, de TCC: **Daniel Augusto Ribeiro Chaves**

Data: 4/07/2019

Resumo

Existe uma grande quantidade de geradores de compiladores responsáveis por criar parsers ou analisadores sintáticos para uma linguagem de programação, a partir de uma especificação formal, comumente a notação BNF ou EBNF [1]. Como exemplo podemos citar o ANTLR, JavaCC, Bison, Yacc, Packrat, PyParsing e SableCC. Cada um desses geradores implementa algum tipo de algoritmo de parser, podendo ser LALR, SLR, LR entre outras, que irá determinar a capacidade de criar linguagens mais ou menos poderosas. Este trabalho objetiva desenvolver uma IDE para o gerador de parsers intitulado COMPGEN, desenvolvido no grupo de pesquisa em Linguagens de Programação e Engenharia de Software da Universidade de Pernambuco (UPE).

Abstract

There is a large number of compiler generators responsible for creating parsers or syntactic for a programming language, from a formal form, commonly a BNF or EBNF notation [1]. As an example we can mention ANTLR, JavaCC, Bison, Yacc, Packrat, PyParsing and SableCC. Each of these generators can be used as a type of analysis algorithm, being able to be configured as LALR, SLR, LR among others, which will determine the ability to create more or less powerful languages. This work aims to develop an IDE for the software program generator of the University of Pernambuco (UPE).

Sumário

Sumário

Índice de Figuras

Tabela de Símbolos e Siglas

BNF – Backus Naur Form (Formalismo de Backus-Naur)

EBNF – Extended Backus Normal Form

IDE - Integrated Development Environment

JDT – Java Development Tools

LALR – Look Ahead Left Right

LR – Left-Right

PDE – Plugin Development Environment

SLR – Simple Left-Right

Capítulo 1
Introdução

As ferramentas de construção de software no ambiente de programação podem facilitar grandemente a criação de um compilador eficiente [2]. O uso de um ambiente integrado de desenvolvimento ou IDE (Integrated Development Environment) pode trazer diversos benefícios para as pessoas envolvidas no projeto, assim como também pode aumentar a produtividade, diminuindo gastos e aumentar o desempenho. Com ela é possível medir resultados, fazer a verificação e correção de erros, além de integrar tecnologias [3]. Entretanto, mesmo após anos de desenvolvimento em linguagens e geradores, a maioria dos geradores de parsers não possui uma IDE, o que dificulta o trabalho de especificação da linguagem, tornando o processo menos produtivo e amigável.

Em particular, poucos geradores como o ANTLR possuem uma IDE disponível, como, por exemplo, o ANTLRWorks, que visa uma rápida prototipagem de gramáticas usando o gerador ANTLR. O ANTLRWorks foi desenvolvido sobre a plataforma Eclipse, que é uma plataforma de desenvolvimento de software livre extensível, baseada em Java. Dentre os recursos disponíveis nesta IDE podemos citar: realce de sintaxe, completar automaticamente o código (code completion) e auxílio à navegação.

Enquanto ferramentas como o ANTLR possuem uma IDE bastante robusta e completa, o COMPGEN, gerador no qual esse projeto está sendo fundamentado, está entre os diversos geradores de parsers que possui um limitado conjunto de ferramentas. O COMPGEN é um gerador de linguagens que implementa um algoritmo LR(k), ou seja, um LR com um lookahead de k [1]. Além do gerador de parsers, ele possui apenas uma ferramenta de edição com uma interface web para a escrita de uma gramática usando uma notação similar a EBNF, uma linguagem que pode ser considerada uma metalinguagem pois através dela, pode-se criar outras linguagens através de sua especificação formal. [4] Esta gramática ainda conta com elementos

de Orientação à Objetos, tais como a herança que é a possibilidade de reuso do código e a abstração, representação de um objeto real dentro do sistema. [5].

1.1 Objetivos da ferramenta proposta

Este trabalho objetiva construir uma IDE para o COMPGEN que permita a edição de linguagens de forma mais produtiva e integrada e o desenvolvimento de gramáticas COMPGEN para novas linguagens. Neste contexto, destacam-se as funções básicas de um editor de linguagens, isto é, edição sensível a linguagem, auxílio à documentação, code completion e suporte à navegação [6].

1.2 Metodologia

1.2.1 Materiais e Métodos

Como requisitos para a IDE COMPGEN foram definidos os recursos de: coloração de sintaxe, auto-complete folding e hyperlinks. Para a análise e projeto da IDE COMPGEN foi utilizada a ferramenta ASTAH UML para a definição dos diagramas de classes UML que apresentam as classes utilizadas em cada uma das funcionalidades implementadas. Além disso, para o desenvolvimento utilizou-se o Eclipse PDE (Plugin Development Environment) para definir o plugin do editor COMPGEN.

1.2.2 Resultados e Impactos Esperados

A principal contribuição do desenvolvimento de uma IDE para o COMPGEN é fornecer uma maior assistência na criação de outras linguagens a partir da implementação de um editor para a linguagem assim como incrementar algumas funcionalidades para a mesma.

Espera-se que, como resultado, obtenha-se um plugin capaz de de ser integrado à plataforma Eclipse que fornecerá um ambiente integrado de desenvolvimento para o COMPGEN. Além disso, será possível definir uma

linguagem em um ambiente construído com recursos para o desenvolvimento mais ágil e amigável.

Deseja-se ainda que a ferramenta desenvolvida possa ser incubada no projeto Eclipse, como contribuição no estudo dos geradores de parsers , e estar disponível para toda a comunidade.

1.2.3 Organização do Trabalho

Este trabalho está estruturado da seguinte forma: o capítulo 2 apresenta a fundamentação teórica, que objetiva contextualizar a base na qual o projeto irá se desenvolver (a plataforma eclipse e alguns de seus recursos essenciais) como também apresentar como funcionam algumas funcionalidades presentes na maioria dos editores de código fonte conhecidos atualmente. O capítulo 3 descreve os materiais e métodos utilizados construção do plugin COMPGEN. O capítulo 4 apresenta a construção do plugin Eclipse que define a IDE COMPGEN, a partir da implementação dos recursos descritos na seção 3. Por fim, a seção 5 discute as conclusões deste trabalho.

Capítulo 2

Fundamentação Teórica

2.1 Editores de código fonte

Editores de código fonte são editores de texto com funcionalidades adicionais para facilitar a produção de código fonte. Estes servem como um auxílio aos desenvolvedores nas tarefas de construção e manutenção de projetos e podem contar com alguns recursos, que visam maior produtividade em sua execução, tais como codificação, compilação e depuração.

2.2 Ambientes de desenvolvimento integrado

Como mencionado, os Ambientes Integrados de Desenvolvimento são fundamentais para medir resultados, fazer a verificação e correção de erros, além de integrar as diversas tecnologias usadas no desenvolvimento de um projeto de software. Exemplos de ambientes de desenvolvimento integrado que se tornaram referência e padrões mundiais podem ser encontrados tanto como projetos abertos e de código livre, como o Eclipse; quanto em projetos privados como o Microsoft Visual Studio. Esses ambientes fornecem ferramentas para todas as etapas de um processo de desenvolvimento de software, incluindo a edição de código fonte, além de possibilitar a criação e integração de novas ferramentas usando mecanismos como o de plug-ins, ou seja, componentes que podem ser adicionados pelo próprio usuário e podem ser fornecidos por terceiros.

2.3 O Eclipse

O Eclipse é uma plataforma de desenvolvimento de software livre extensível, baseada em Java. Por si só, é simplesmente uma estrutura e um conjunto de serviços para desenvolvimento de aplicativos de componentes de plug-in. Felizmente, o Eclipse vem com um conjunto padrão de plug-ins, incluindo as amplamente conhecidas Ferramentas de Desenvolvimento Java (JDT) [7]. A plataforma contém a funcionalidade IDE e é construída com componentes que criam aplicativos usando subconjuntos de componentes. Os desenvolvedores criam, compartilham e editam projetos e arquivos genéricos na plataforma, enquanto participam de um repositório de ambiente de desenvolvimento de várias equipes.

As raízes do Eclipse remontam a 2001. A base de código inicial foi fornecida pela IBM. Em novembro de 2001, um consórcio foi formado para apoiar o desenvolvimento do Eclipse como software de código aberto. Este consórcio foi controlado pela IBM. Em 2004, tornou-se a Fundação Eclipse, que é uma base neutra de fornecedores, onde nenhuma empresa controla a direção [3].

A Fundação Eclipse foi lançada em 2 de fevereiro de 2004, pouco mais de 2 anos após o Consórcio, e trouxe uma corporação sem fins lucrativos, com um corpo de diretores e um ecossistema independentes, para suportar parceiros estratégicos e fornecedores de extensões/plugins. Na época do lançamento do Eclipse 3.1 em 2005 a Fundação já contava com 100 organizações participantes [8].

O Eclipse fornece alguns editores básicos, como editores de texto e de código fonte Java, juntamente com alguns editores de várias páginas mais complexos, como o editor de plug-in manifest. Produtos que precisam apresentar seus próprios editores podem usar os mesmos pontos de extensões usados pelos editores internos do Eclipse [9].

2.4 Construção de uma IDE

Desenvolver uma IDE não é um trabalho difícil, porém tende a ser complexo pois envolve diferentes componentes, ferramentas e recursos em seu desenvolvimento. Tendo isso em vista, seu projeto de construção pode ser fatorado em pequenas partes, para que torne os processos de desenvolvimento, manutenção e integração mais fáceis.

Esse tipo de editor pode conter vários recursos em sua construção; alguns deles essenciais, como por exemplo um editor de código fonte, uma configuração para compilação do que foi escrito e algum tipo de visualização dos resultados obtidos. Outros recursos podem ser chamados de extras, tais como edição sensível à linguagem, suporte à navegação, auxílio à documentação, etc. A utilização dessas ferramentas torna o trabalho de criação e gerenciamento dos projetos mais otimizados, uma vez que os processos como análise e execução dos artefatos criados tornam-se mais ágeis.

2.4.1 Realce de Sintaxe

Recurso presente na maioria das IDEs permite que o código fonte escrito seja destacado baseado na linguagem em que foi escrito, facilitando sua leitura.

```
package compgen_project.scanner;

import org.eclipse.jface.text.rules.IWordDetector;

public class IdDetector implements IWordDetector {

    @Override
    public boolean isWordPart(char arg0) {
        return Character.isLetter(arg0) || Character.isDigit(arg0);
    }

    @Override
    public boolean isWordStart(char arg0) {
        return Character.isLetter(arg0);
    }
}
```

Figura 1. Realce de sintaxe na plataforma Eclipse.

Fonte: Próprio autor.

2.4.2 Completação de Código

Completação de código, mais conhecido como autocomplete é o recurso responsável por apresentar sugestões de complementos na escrita do código fonte, pelo usuário. As sugestões podem ser feitas em referência às variáveis, aos métodos ou às palavras chaves da linguagem. Em suma, podemos dizer que fica à critério do desenvolvedor do recurso implementar a robustez necessária para atender as necessidades de quem utiliza o plugin.

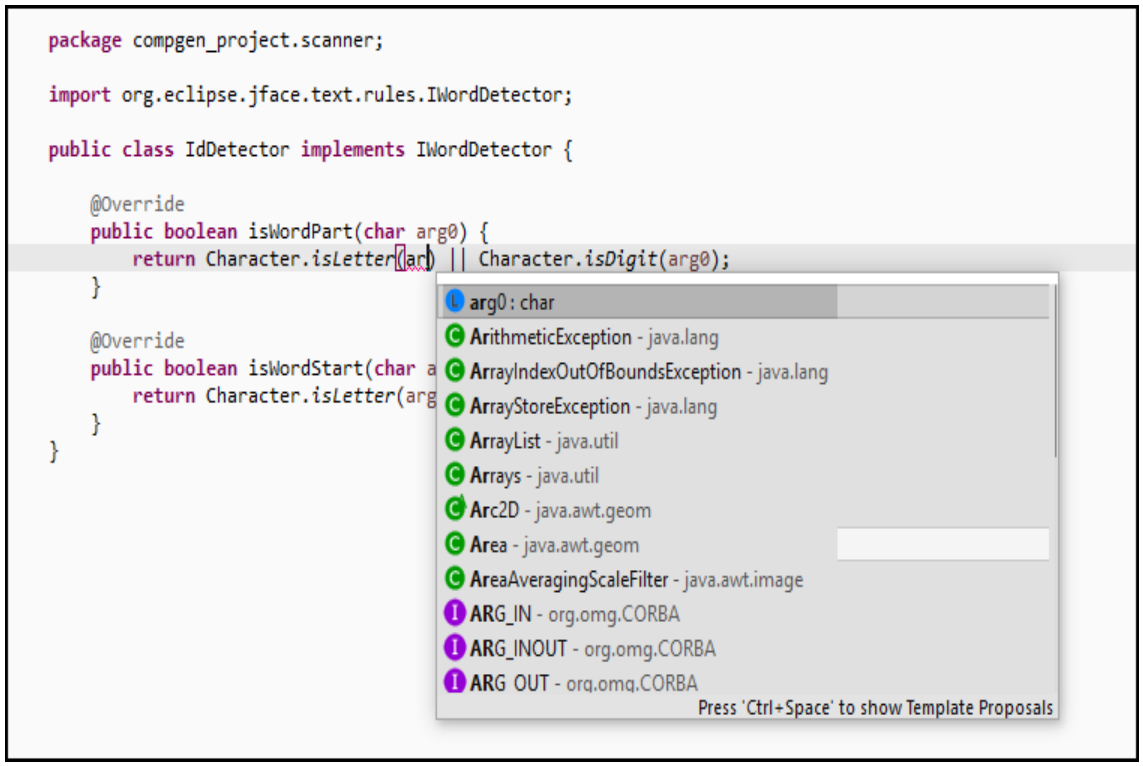


Figura 2. *Autocomplete* na plataforma Eclipse.

Fonte: Próprio autor.

2.4.3 Folding

Folding é um recurso no qual o usuário revela e esconde trechos de código no editor de sua linguagem, dentro do projeto. Esse mecanismo é bastante útil para uma melhor visualização e estruturação da escrita do código fonte.

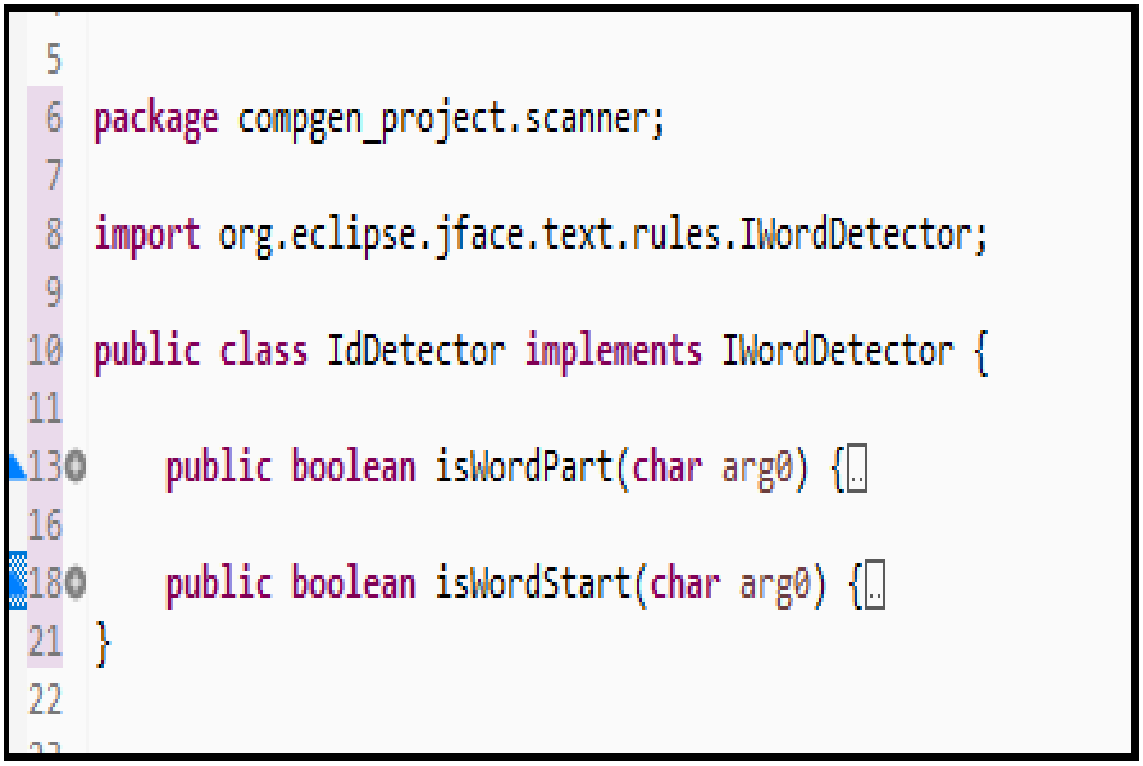
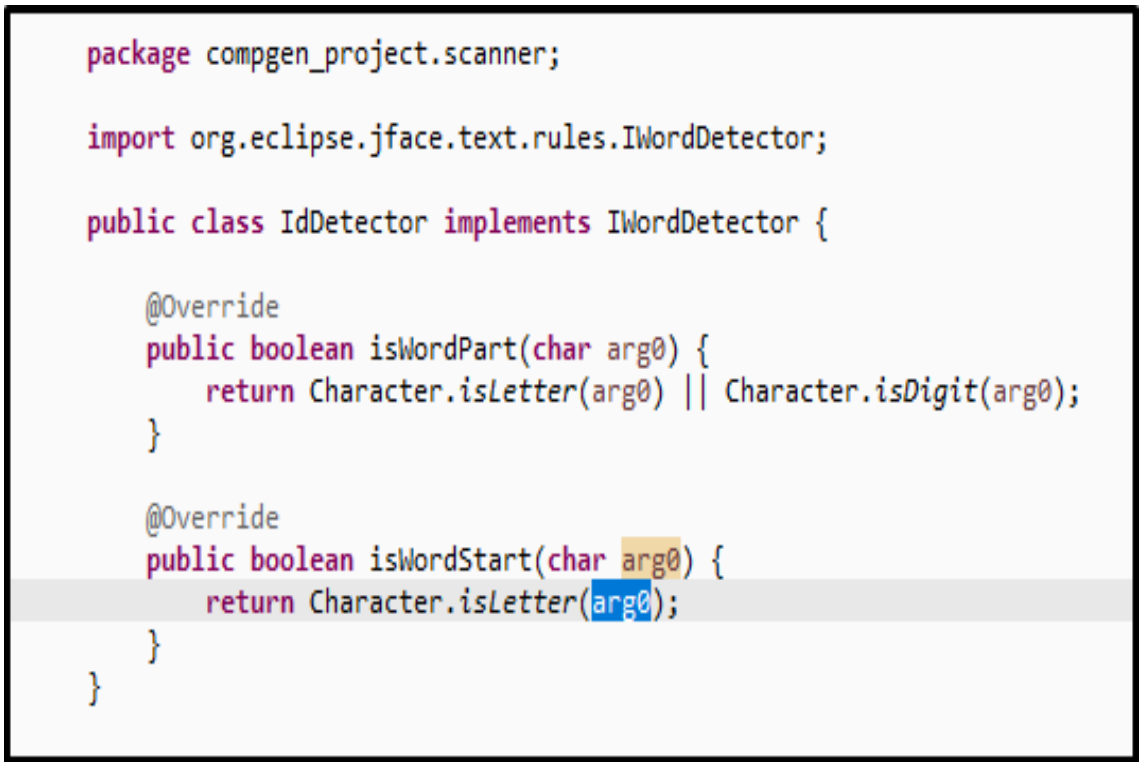


Figura 3. *Folding* na plataforma Eclipse.

Fonte: Próprio autor.

2.4.4 *Hyperlinks*

Recurso responsável por permitir a navegação dentro do código fonte escrito no projeto, auxiliando na busca de declarações ou documentações do projeto.



```
package compgen_project.scanner;

import org.eclipse.jface.text.rules.IWordDetector;

public class IdDetector implements IWordDetector {

    @Override
    public boolean isWordPart(char arg0) {
        return Character.isLetter(arg0) || Character.isDigit(arg0);
    }

    @Override
    public boolean isWordStart(char arg0) {
        return Character.isLetter(arg0);
    }
}
```

Figura 4. *Hyperlinks* na plataforma Eclipse.

Fonte: Próprio autor.

Capítulo 3

Construção do PLUGIN COMPGEN

Nesta seção serão contempladas as etapas da implementação do editor para o gerador de parser COMPGEN utilizando o Eclipse PDE.

3.1 COMPGEN Realce de Sintaxe

Este traço comum a várias IDEs faz referência à ideia discutida na seção 2.4.1. Este recurso é responsável realizar a coloração de diferentes partes do código, respeitando as regras de escolha das áreas desejadas a serem destacadas, tais como palavras reservadas, etc. A figura 5 mostra o diagrama de classes que mostra atributos e métodos das entidades relacionadas na função de coloração no IDE.

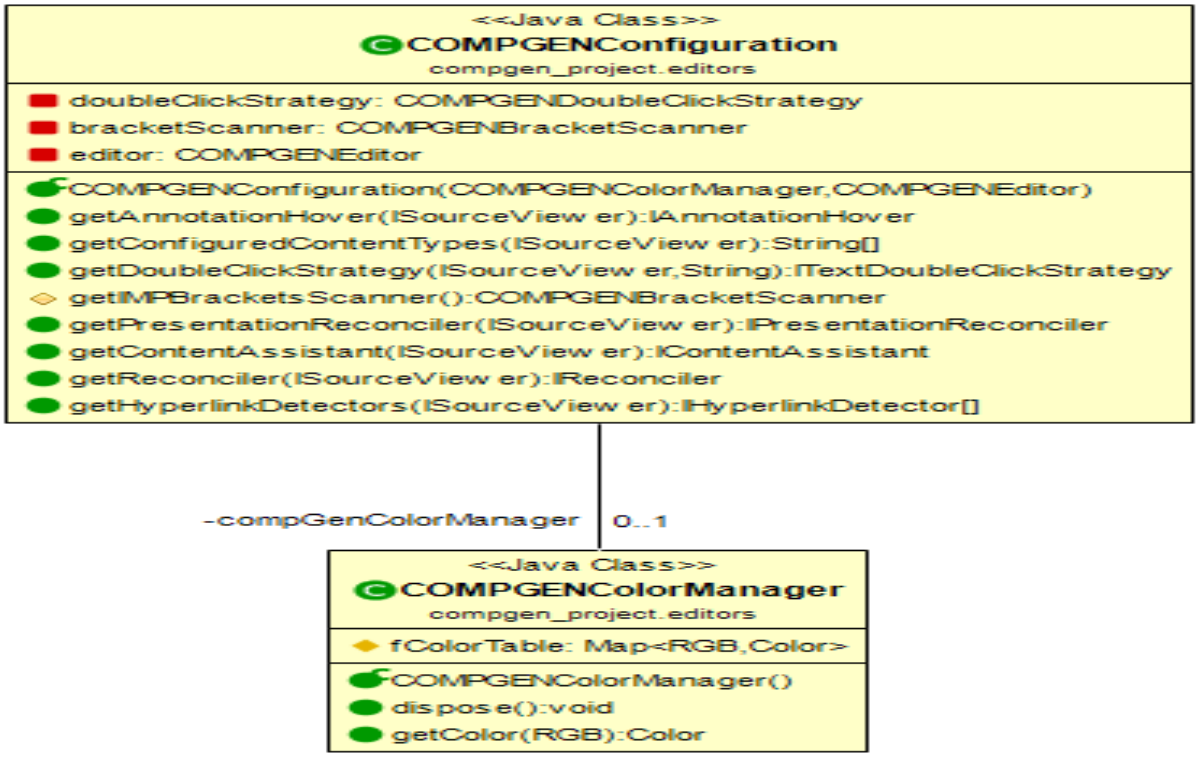


Figura 5. Diagrama de classes das entidades responsáveis pelo realce de sintaxe.

Fonte: Próprio autor.

3.2 COMPGEN Completação de Código

Este recurso faz referência ao que foi abordado na seção 2.4.2. Esta funcionalidade faz com que o sistema dê ao usuário a possibilidade de completar palavras à medida em que vão sendo escritas no editor de linguagem. A figura 6 mostra o diagrama de classes dos componentes envolvidos neste recurso.

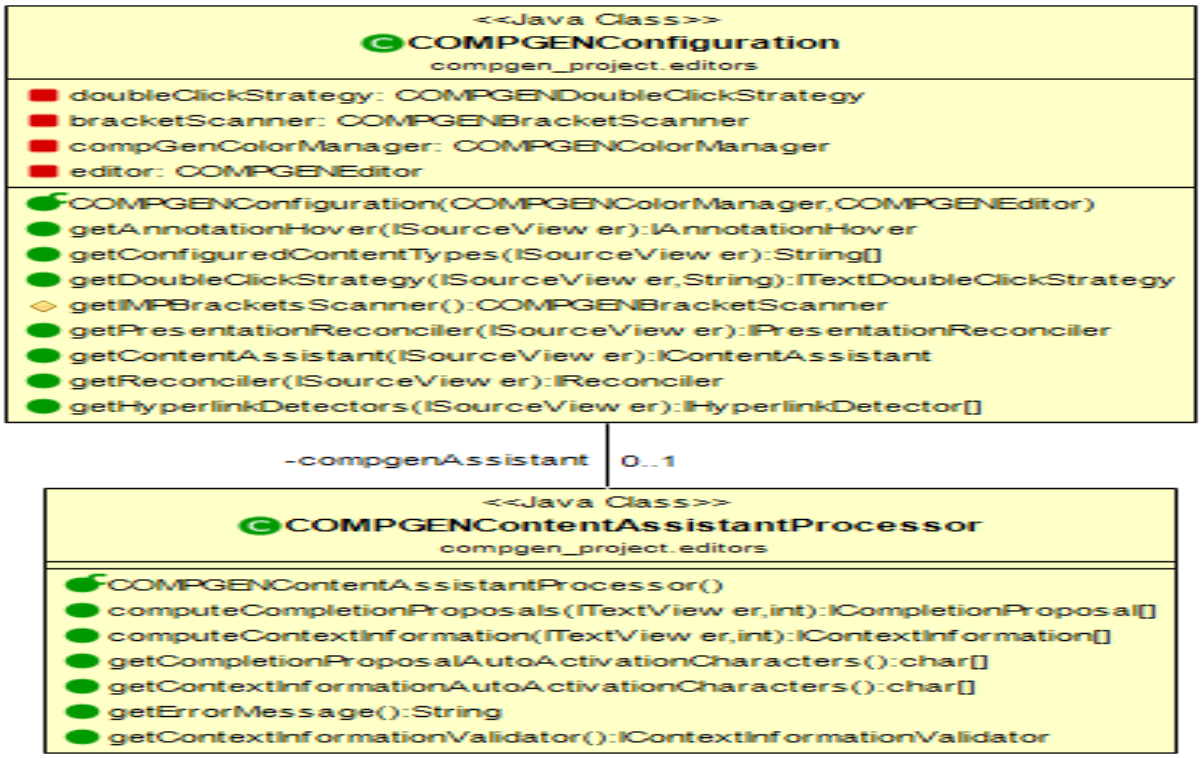


Figura 6. Diagrama de classes das entidades responsáveis pela completação de código.

Fonte: Próprio autor.

3.3 COMPGEN *Folding*

Este tipo de preferência permite que o usuário esconda ou mostre trechos de códigos nos quais não se está trabalhando no momento. Esse tipo de recurso representa um ganho no que se diz respeito ao desenvolvedor ter que lidar com códigos extensos em um espaço limitado. A figura 7 ilustra o diagrama de classes que envolve os agentes responsáveis por este recurso, no COMPGEN.

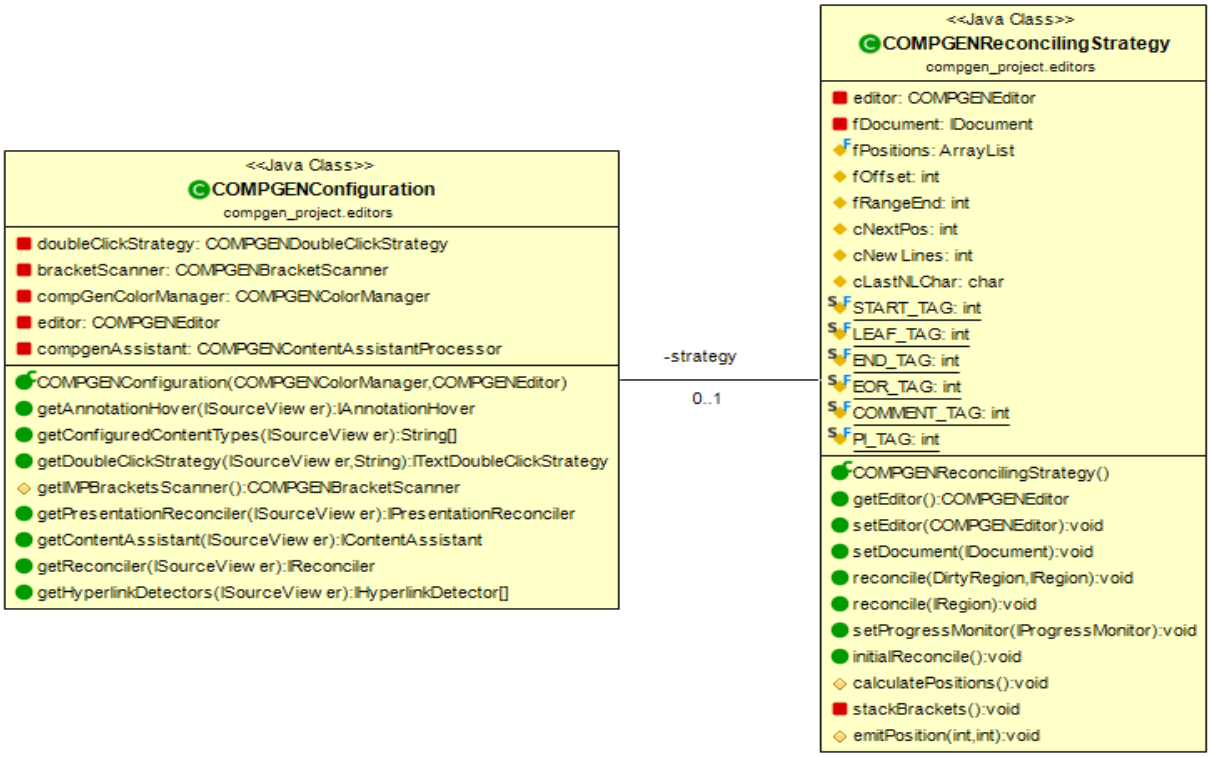


Figura 7. Diagrama de classes das entidades responsáveis pelo *folding.*

Fonte: Próprio autor.

3.4 COMPGEN *Hyperlinks*

Este recurso permite que o desenvolvedor do código navegue entre as definições dos termos existentes em sua gramática a partir de um clique em qualquer elemento da linguagem. O plugin COMPGEN implementa este tipo de recurso na classe `COMPGENConfiguration`, através do método

```
IHyperlinkDetector[] getHyperlinkDetectors(ISourceViewer sourceViewer).
```

Além disso, é necessário criar uma estrutura capaz de obter os tokens do nosso gerador de linguagens, a fim de armazenar suas ocorrências no documento além da região em que foram encontradas suas declarações.

A classe `COMPGENHyperLinkDetector` implementa a interface IHyperlinkDetector que nos permite construir o método

`IHyperlink[] detectHyperlinks(ITextViewer textViewer, IRegion region, boolean canShowMultipleHyperlinks).`

A fim de identificar os links na gramática, utilizamos o método split da classe String, em Java, que nos permite realizar a quebra do código fonte escrito a partir da passagem de uma expressão regular. A partir daí, criamos um mapeamento entre os tokens obtidos pelo nosso gerador de linguagens e a região em que se encontram, no documento. Por fim, o método identifica para qual a região do código o recurso deverá nos redirecionar, ou seja, para qual parte do documento o usuário irá navegar no momento do clique de um elemento específico no código. A figura 8 ilustra a construção do método que realiza esse mapeamento em nosso plugin.

```
private Hashtable<String, Region> getTokensFromDocument(String pText, String pRule) {

    Hashtable<String, Region> vHashTokens = new Hashtable<String, Region>();

    String[] tokensEncontrados = pText.split(pRule);
    ArrayList<String> Tokens = new ArrayList<String>(Arrays.asList(tokensEncontrados));
    Tokens.removeAll(Arrays.asList("", null));

    int vOffSet = -1;
    for (String vTokenAtual : Tokens) {
        if (vHashTokens.get(vTokenAtual) == null) {
            vOffSet = pText.indexOf(vTokenAtual, vOffSet + 1);
            vHashTokens.put(vTokenAtual, new Region(vOffSet, vTokenAtual.length()));
        }
    }

    return vHashTokens;
}
```

Figura 8. Mapeamento entre os tokens do gerador de linguagens e suas regiões no código fonte.

Fonte: Próprio autor.

As expressões regulares que permitem a identificação dos links na gramática foram construídas utilizando os moldes de implementação de um analisador léxico, ou seja, é necessário reconhecer todos os símbolos válidos de nossa linguagem. A figura 9 mostra as expressões regulares criadas para obter todos os termos do documento.

```
private final String aDelimiterKeyWords = "|language|class|compile|extends|syntax|this|val|eval|print|asm|forEach|nextLabel|opCodeOf"
        + "|toString|lexical|whitespace|start|var";


private final String aDelimiterArithmeticOperators = "|\\+|\\-|\\*|\\/|%|\\+\\+|--";


private final String aDelimiterRelationalOperators = "|(!=|==|<=|>=|>|<)";


private final String aDelimiterLogicalOperators = "|&&|\\|\\||<=|>=|>|<";


private final String aDelimiterAssignmentOperators = "|=|\\+=|-=|\\*=|/=|%=";


private final String aDelimiterSeparators = "|\\(|\\)|\\{|\\}|\\[|\\];|\\.|'(.*?)'|:";


private final String aDelimiterStringLiterals = "|\\\"([^\\\\\\\"]|\\\\.)*\\\"";
```

Figura 9. Expressões regulares utilizadas na quebra do documento para obtenção dos elementos da linguagem.
Fonte: Próprio autor.

A classe `COMPGENHyperLink` implementa a interface `IHyperlink` que, durante a redefinição do método open, destaca a região alvo do link selecionado. O diagrama de classes da figura 10 torna mais compreensível o relacionamento das classes envolvidas na implementação deste recurso.

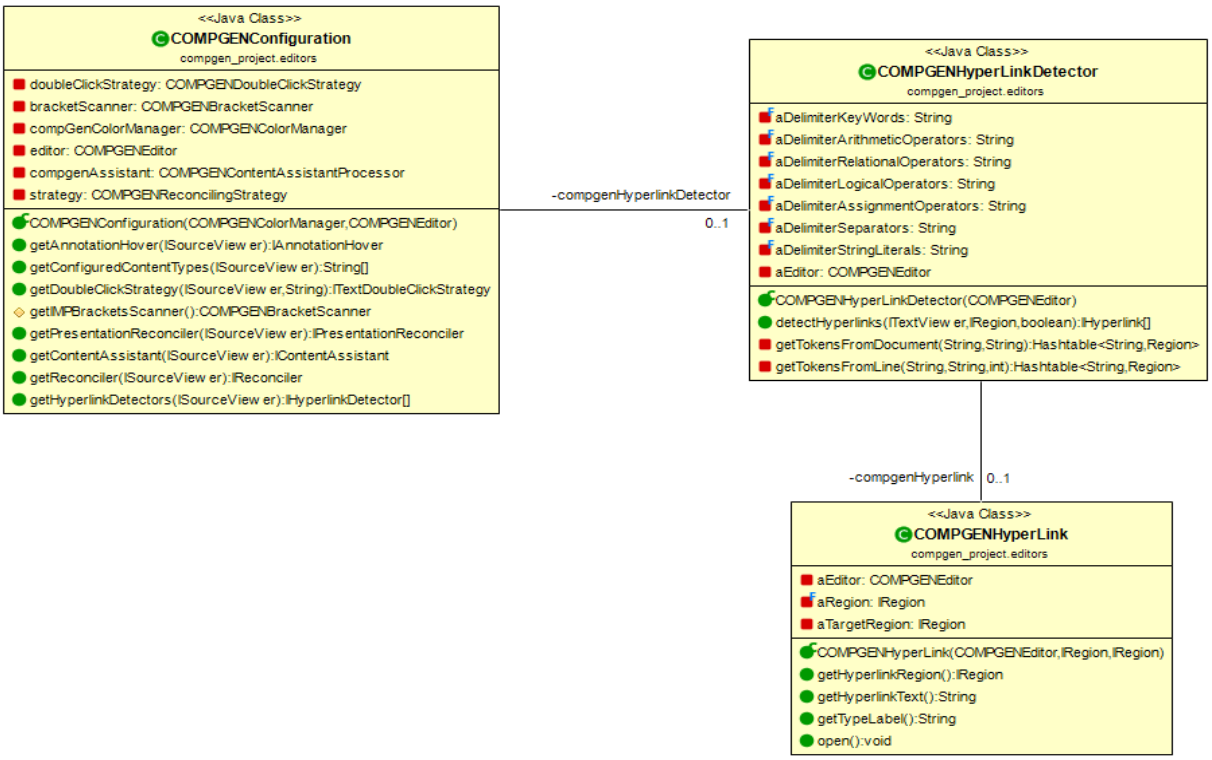


Figura 10. Diagrama de classes das entidades responsáveis pelo *hyperlinks.*

Fonte: Próprio autor.

Capítulo 4

Estudo de Caso

Este capítulo tem como objetivo apresentar um exemplo de utilização da IDE desenvolvida neste trabalho através da criação de uma linguagem hipotética intitulada Imp, através do COMPGEN.

4.1 IMP

A figura 11 mostra como foi definida as regras para esta linguagem.

```
language Imp {


    class Com {

       compile(dest) {
       }
    }

    class Assign extends Com {
       syntax [[ id:ID ':=' val:Exp ';' ]]

       compile(asm) {
            this.val.eval(asm);
            asm.print("store " + this.id);
       }
    }

    class Block extends Com {
       syntax [[ "{" coms:Com* '}' ]]
       compile(asm) {
           this.coms.forEach(c => c.compile(asm));
       }
    }
    class While extends Com {
       syntax [[ "while" test:Exp "do" c:Com ]]

       compile(asm) {
          var l1 = asm.nextLabel();
          var l2 = asm.nextLabel();
          asm.print(l1+":");
          this.test.eval(asm);
          asm.print("jumpf "+l2);
          this.c.compile(asm);
          asm.print("jump "+l1);
          asm.print(l2+":");
       }
    }

    class Exp {
    }

    class Id extends Exp {
       syntax [[ id:ID ]]

       eval(dest) { dest.print("load " +this.id); }
    }

    class Op extends Exp {
       syntax [[ left:Exp o:OP right:Exp ]]

       eval(asm) {
          this.left.eval(asm);
          this.right.eval(asm);
          asm.opCodeOf(this.o.toString());
       }
    }

    class Group extends Exp {
       syntax [[ '(' e:Exp ')' ]]

       eval(asm) { this.e.eval(asm); }
    }
    class Num extends Exp {
       syntax [[ val:NUM ]]
       eval(dest) {
          dest.print("push " + this.val);
       }
    }
    lexical {
       OP '\+|\-|\*|/|(==)|(!=)|(>=)|>|<|(<=)' '';
       NUM '[0-9]+' 'color=green';
       ID  '[a-z]+' 'color=blue';
    }


    whitespace = '[ \n\t]+';
    start = Com;
}
```

Figura 11. Trecho da gramática do Imp.

Fonte: Próprio autor.

4.2 Projeto Imp no COMPGEN IDE

Primeiramente, iniciaremos um projeto genérico para que possamos armazenar arquivos de escrita de novas gramática, como mostrado na figura 12.

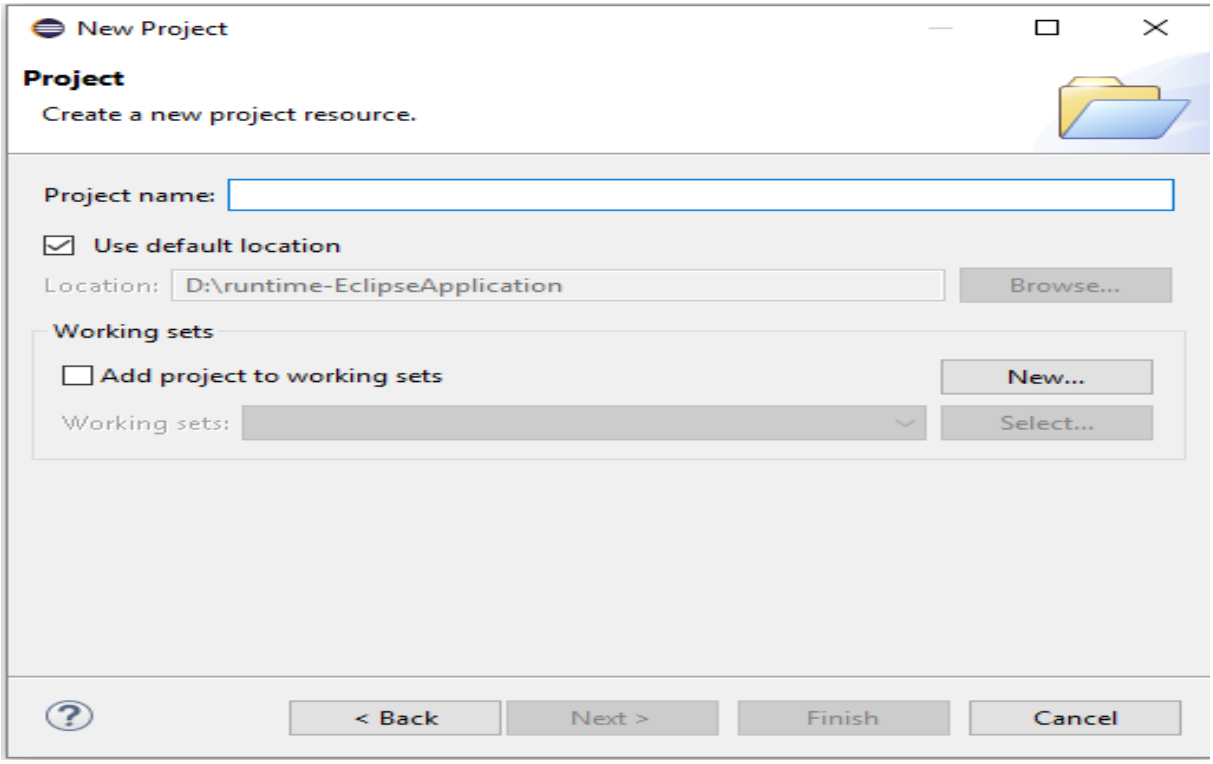


Figura 12. Tela de criação de um novo projeto genérico.

Fonte: Próprio autor.

Em seguida, criaremos um arquivo de extensão .cpg para que os recursos criados para este editor de código fonte possam ser utilizados. O processo de criação deste arquivo está ilustrado na figura 13.

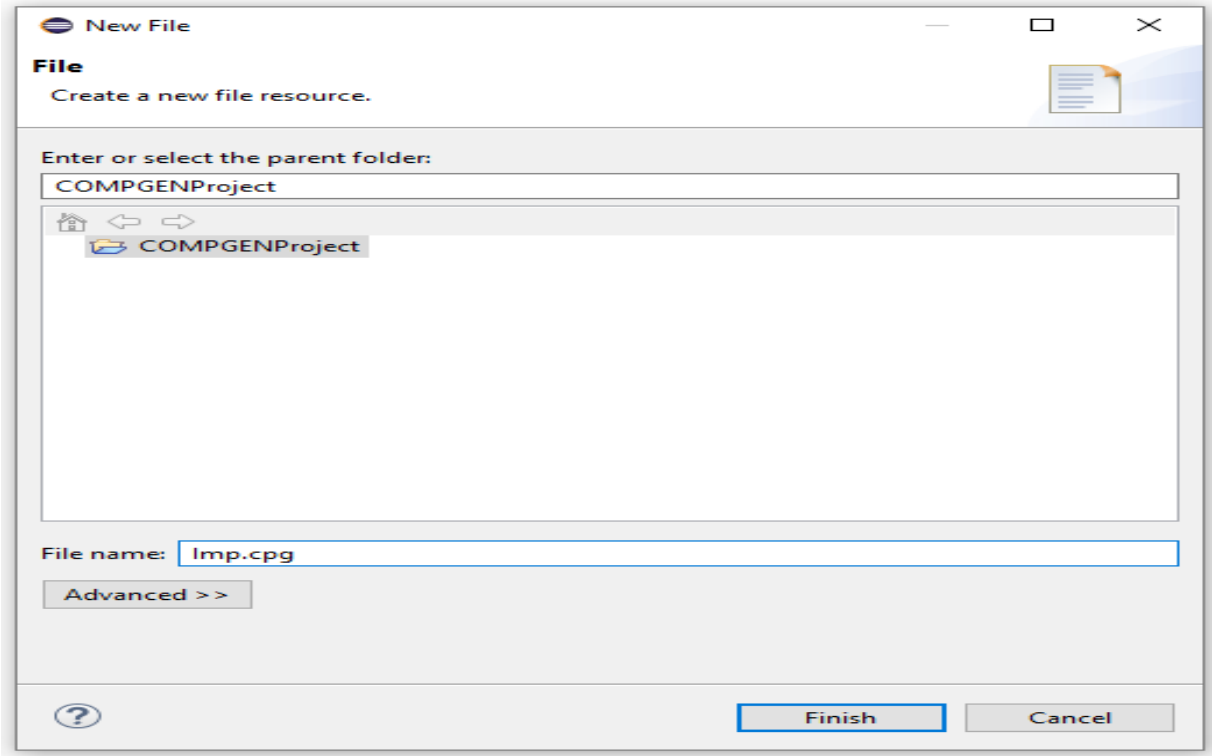


Figura 13. Tela de criação de um novo arquivo de extensão *.cpg* para criação da linguagem Imp.
Fonte: Próprio autor.

Na etapa seguinte, manipularemos o código de geração da linguagem Imp no editor propriamente dito. Um dos primeiros recursos que podemos notar em sua utilização é o realce de sintaxe, uma das características mais importantes em uma IDE, descrito na seção 2.4.1. Além disso, observar a utilização do recurso folding já que o código fonte é extenso e a IDE possui a habilidade de dobrar trechos de código, conforme descrito na seção 2.4.3. A figura 14 ilustra de forma mais clara a utilização de ambos os recursos citados anteriormente.

Imp.cpg

```
language Imp {

    class Com {..
    }..

    class Assign extends Com {
        syntax [[ id:ID ':=' val:Exp ';' ]]

        compile(asm) {
            this.val.eval(asm);
            asm.print("store " + this.id);
        }
    }


    class Block extends Com {
        syntax [[ "{" coms:Com* '}' ]]
        compile(asm) {
            this.coms.forEach(c => c.compile(asm));
        }
    }

    class While extends Com {..
    }..

    class Exp { ..
    }..

    class Id extends Exp {..
    }..

    class Op extends Exp {..
    }..

    class Group extends Exp {..
    }..

    class Num extends Exp {..
    }..

    lexical {..
    }..

    whitespace = '[ \n\t]+';
    start = Com;

}
```

Figura 14. Demonstração dos recursos de realce de sintaxe e *folding*.

Fonte: Próprio autor.

A seção 2.4.2 destaca a utilização do recurso de completação de código presente em várias IDEs. Com ele implementado no editor para COMPGEN, é possível automatizar a escrita de palavras chave e assim ser capaz de aumentar a produtividade e diminuir o esforço gasto pela repetição. A figura 15 mostra de forma mais objetiva a utilização deste recurso.

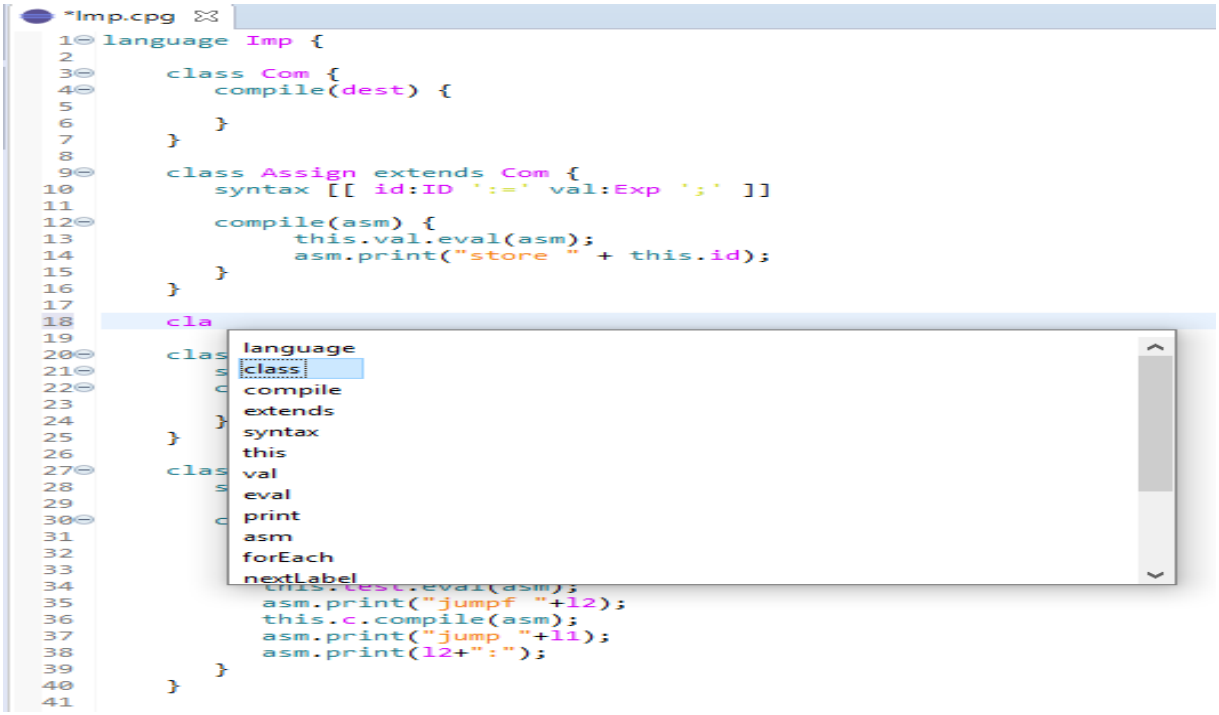


Figura 15. Demonstração do recurso de completação de código.

Fonte: Próprio autor.

Por fim, como descrito na seção 2.4.4, o recurso de *hyperlinks* permite que o usuário navegue através do código escrito até a declaração de algum token da linguagem do COMPGEN. Caso o elemento clicado seja um termo válido encontrado no documento, então o editor permitirá fazer daquela região uma ponte para um outro lugar na gramática. A figura 16 ilustra a utilização deste recurso, onde o elemento Com na linha 9, o usuário é levado à declaração deste termo na linha 3.

```
3     class Com {
4        compile(dest) {
5
6        }
7     }
8
9     class Assign extends Com {
10       syntax [[ id:ID ':=' val:Exp ';' ]]
11
12       compile(asm) {
13            this.val.eval(asm);
14            asm.print("store " + this.id);
15       }
16    }
17
18
19    class Block extends Com {
20       syntax [[ "{" coms:Com* '}' ]]
21       compile(asm) {
22           this.coms.forEach(c => c.compile(asm));
23       }
24    }
```

Figura 16. Demonstração do recurso de hyperlinks.

Fonte: Próprio autor.

Capítulo 5
Conclusão

O desenvolvimento deste estudo nos permitiu mostrar a importância da utilização de um ambiente de desenvolvimento integrado no trabalho de criação, gerenciamento e execução de um projeto e de que forma a utilização dos recursos essenciais e não essenciais pode agilizar essas atividades.

Diante da necessidade da existência de um IDE para geração de linguagens através do COMPGEN, o plugin elaborado neste projeto contribui de forma positiva não só ao desenvolvedor, que irá contar com alguns recursos implementados na ferramenta, mas também à comunidade Eclipse, que possui um ambiente aberto para colaborações de trabalhos deste tipo.

Bibliografia

[1] D. COOPER, Keith;TORCZON, Linda. Engineering a Compiler. 2. ed. Estados Unidos: Elsevier, 2012.ALEKSANDER, I. e MORTON, H. **An Introduction to Neural Computing.** International Thomson Computer Press, 1995. 490 p.

[2] AHO, ALFRED V.; SETHI, RAVI; ULLMAN, JEFFREY D. Compilers Principles, Techniques, and Tools. Tradução: DANIEL DE ARIOSTO PINTO. 2. ed. Estados Unidos: LTC - Livros Técnicos e Científicos Editora S.A, 1995. 344 p.

[3] VOGEL, Lars. Contributing to the Eclipse IDE Project - Principles, Plug-ins and Gerrit Code Review. 1 ed. 2015.

[4] OMASSETTI, Federico. EBNF: How to Describe the Grammar of a Language. [S. l.], 1 ago. 2017. Disponível em: https://tomassetti.me/ebnf/#what. Acesso em: 26 jun. 2019.

[5] NEGRESIOLO, Leticia. Tudo o que você precisa (e deveria) saber sobre Programação Orientada a Objetos. [S. l.], 22 jan. 2019. Disponível em: https://gizmodo.uol.com.br/tudo-sobre-programacao-orientada-a-objetos/. Acesso em: 26 jun. 2019.

[6] TEXT Editors for Programmeres - Programming Tools. [S. l.], [20--]. Disponível em: https://www.cprogramming.com/texteditors.html. Acesso em: 26 jun. 2019.

[7] ANISZCZYK, Chris; GALLARDO, David. Introdução à Plataforma Eclipse. [S. l.], 6 fev. 2012. Disponível em: https://www.ibm.com/developerworks/br/library/os-eclipse-platform/index.html. Acesso em: 2 jun. 2019.

[8] BLEWITT, Alex. 10 Anos de Eclipse. [S. l.], 21 nov. 2011. Disponível em: https://www.infoq.com/br/news/2011/11/eclipse-10/. Acesso em: 1 jun. 2019.

[9] CLAYBERG, Eric; RUBEL, Dan. Eclipse Plug-ins. [S. l.: s. n.], 2008. E-book.