3 IMPLEMENTATION

A Domain Specific Language is a programming language with a higher level of abstraction optimized for a specific class of problems [4]. Since this chapter aims to present the implementation of this DSL, it will be analysed the grammar and the lexer and parser.

3.1 Grammar

The syntax of a programming language is the set of rules that define which arrangements of symbols comprise structurally legal programs. Grammar is defined by four elements in n order of $G = (V_N, V_T, P, S)$. The meaning of the elements is as follows:

- V_N - set of nonterminal symbols.
- V_T - set of tokens or terminal symbols.
- P - set of production rules.
- S - start symbol.

For further understanding, in the Tab. 3.1 are listed the meta notations used throughout this paper.

Symbol	Meaning
<abc>	A nonterminal symbol
abc	A terminal symbol
x^*	Zero or more occurrences of x
x^+	One or more occurrences of x
$x^?$	Zero or one occurrence of x
|	Separates alternatives

Table 3.1 Meta Notations

For the project in question, the elements were defined as follows.

V_N = { <call_method>, <method_body>, <method_body_string>, <method_name>, <method_parameter>, <begin_method>, <end_method>, <parameter>, <f_name_parameter>, <string_parameter>, <extension_parameter>, <number_parameter>, <image_parameter>, <link_parameter>, <type_doc_parameter>, <text>, <text_char>, <Lcase_letter>, <Ucase_letter>, <number>, <img_extension_name>, <symbol>, <round_bracket_Left>, <round_bracket_Right>, <colon>, <comma>, <low_line>, <quotation_mark>, }

V_T = {[0-9], [a-z], [A-Z], ({ }) : " , _ ? / - . " " {^ ^} , report, research, docx, pdf, jpeg, png, jpg, ε}

S = {<call_method>}

P = {<call_method> → <method_name> <round_bracket_Left> <method_parameter>*
<round_bracket_Right> | <method_name> <round_bracket_Left>

<method_parameter>* <round_bracket_Right> <colon> <begin_method>
<method_body> <end_method>

<method_body> → <text>+ <call_method>*

<method_name> → <Lcase_letter>+ | <method_name> <low_line> <method_name>

<method_parameter> → <parameter> | <parameter> <comma> <method_parameter>

<parameter> → <f_name_parameter> | <string_parameter> | <extension_parameter> |
<number_parameter>|<image_parameter>|<link_parameter> |
<type_doc_parameter>

<f_name_parameter> → <text_char>+ <number>*

<string_parameter> → <quotation_mark> <text>+ <quotation_mark>

<extension_parameter>→ **pdf** | **docx**

<number_parameter> → <number>+

<image_parameter> → <string_parameter> **,** <text> **.** <img_extension_name>

<link_parameter> → <text>

<type_doc_parameter> → **report** | **research**

<text> → <text_char>+ <number>* <symbol>* <text>*

<text_char> → **a** | **b** | … | **z** | **A** | **B** | … | **Z** | **ε**

<Lcase_letter> → **a** | **b** | … | **z**

<Ucase_letter> → **A** | **B** | … | **Z**

<number> → **0** | **1** | … | **9**

<doc_extension_name> → **docx** | **pdf**

<img_extension_name> → **jpeg** | **jpg** | **png**

<begin_method> → **{^**

<end_method> → **^}**

<symbol> → (| { | } |) | : | " | , | _ | ? | / | - | .

<round_bracket_Left> → **(**

<round_bracket_Right> → **)**

<comma> → **,**

<colon> → **:**

<low_line> → **_**

<quotation_mark> → **“** | **”**

}

The grammar is further used in the lexer and parser.

3.2 Implementation example

A derivation tree is a graphical representation for the derivation of the given production rules of the context free grammar (CFG). It is a way to show how the derivation can be done to obtain some string from a given set of production rules [5].

Hence, an example of a derivation tree is used with the purpose of a better understanding of how the grammar works in generating correct instructions. Considering Fig. 3.1 it can be seen how choosing repeatedly some rules from the set P generates a valid instruction for defining a new chapter.

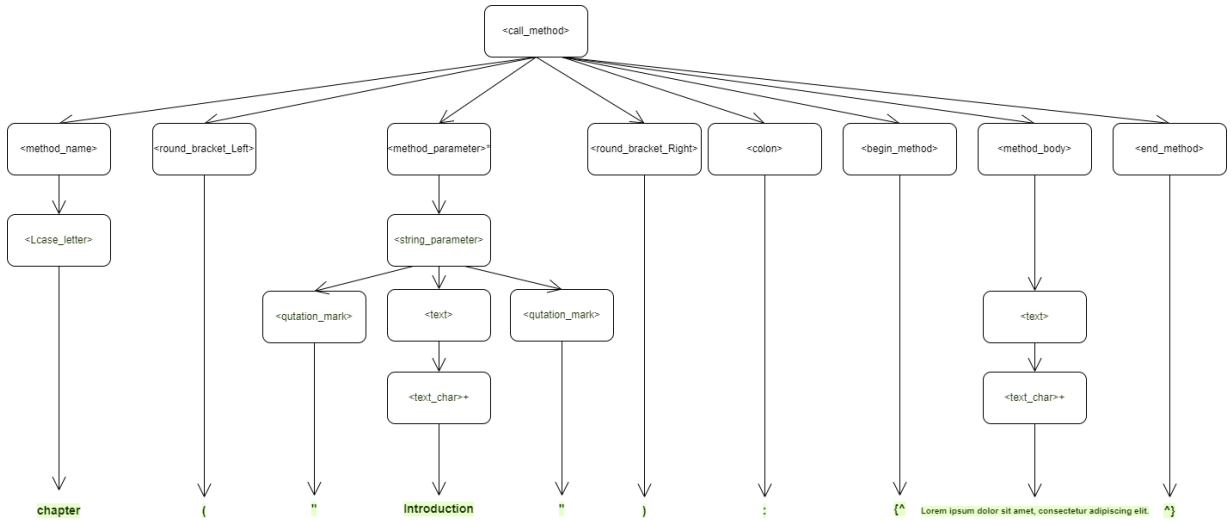

Figure 3.1 Derivation tree example

In the same way, a user generates code and the DSL analyses it by the grammar it was defined on. For a better perspective of what the DSL does, it was considered the following code generated by a user:

```
document( report, Docx)
title("Report Example")
subject("Formal Languages")
author("Gîlca Constantina")

table_of_contents(default)

chapter("Introduction"):
{^
      Lorem ipsum dolor sit amet, consectetur adipiscing elit. In a    nisl
enim. Ut semper, velit hendrerit gravida volutpat, odio neque    malesuada orci, et
imperdiet mi augue in dolor. Etiam efficitur   ultricies risus nec posuere.
Vestibulum accumsan venenatis mauris ac  tincidunt. Mauris ut massa    quam.

      subchapter("Problem Analysis"):
      {^
            Ut condimentum dignissim augue, at bibendum nunc blandit   eu.
Vivamus augue mauris, scelerisque et venenatis et, tincidunt ut  nisi. Suspendisse
interdum massa ut porta condimentum. Proin lorem     nibh, pretium at diam et,
ultrices semper arcu. Donec accumsan dolor     enim, ac luctus mi     fringilla ut.
Integer accumsan lectus accumsan   semper aliquam. Suspendisse scelerisque sem vitae
```

```
libero fermentum, in   consectetur enim sollicitudin. Duis vestibulum gravida
augue, eu   rhoncus diam pulvinar a.

                table_name("Statistics")
                table_row("Year", "Company", "Mean")
                table_row("2001", "Sony", "4.1")

                image(“Face”,/face.png)

                list("For example", arabic_numbers):
                {^
                      item("Item 1")
                      item("Item 2")
                      item("Item 3")
                ^}

          ^}
    ^}
```

After its compilation, the DSL generated the file in Fig. 3.2, which presents the title page, the contents table and the other content generated by the user.

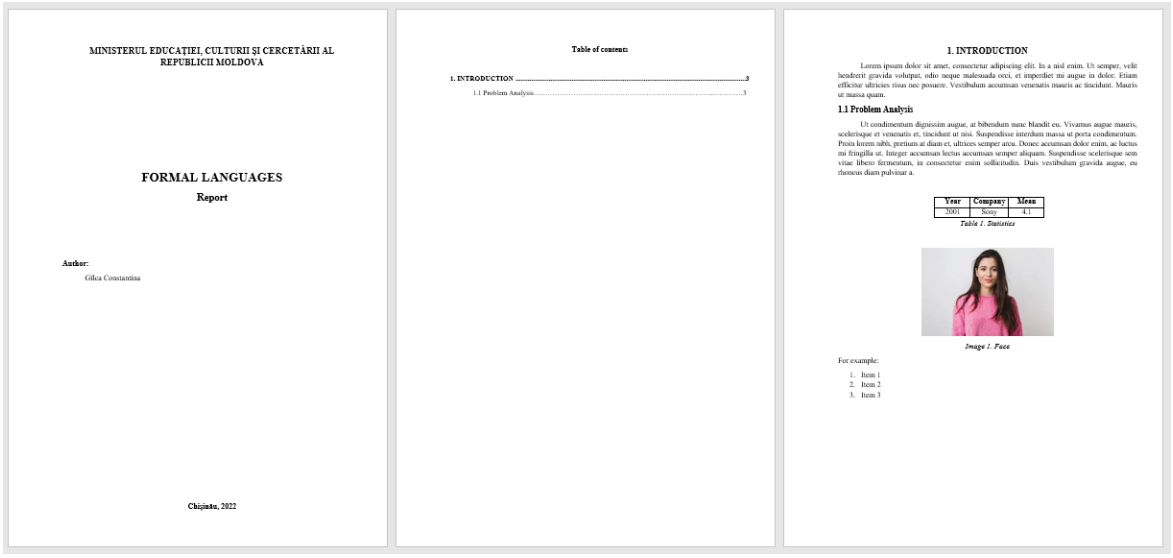

Figure 3.2. A generated document by DSL

The DSL first of all checks the code generated by the grammatic standards it was defined on. Then, for each keyword or word, it generates tokens that are sent to the parser. After that, the parser transforms the tokens into compatible elements for a programming language to compute them and the result is generated.