DOMAIN SPECIFIC LANGUAGE IN THE FIELD OF TAXES CALCULATION

Vasile BOAGHI, Nadejda CÎRNAȚ, Elizabet GALAJU, Mihai ȘCEBEC

Technical University of Moldova, Faculty of Computers, Informatics and Microelectronics

Abstract: *This article represents an analysis of the implementation of a Domain Specific Language in Taxes Calculation. Analyzing the both technical and non-technical subjects, the article has the goal to describe in depth, step-by-step the process of the implementation of the DSL, as well as pointing the priorities and rules. Pointing out the necessities and the branches that should be contained in the language, there were listed the basic features of the DSL, as well as written the basic semantics rules and lexicon of the grammar for the DSL. With the help of ANTLR tool, it was elaborated the parse tree that thoroughly shows all the configuration of the DSL. The new DSL for taxes calculation has the aim of assuring an easy approach, a new vision and methods of utilization of a language.*

Keywords: *DSL, language, grammar, semantics, syntax, taxes, calculations.*

INTRODUCTION

Programming offers a wide range of possibilities that enables skilled people create features, solving everyday problems or routines. Domain Specific Language (DSL) comes to represent the scientific part of the relation between day by day concerns and their programming solution. With the extent of the DSL concept, it is easier now to dive into diverse fields of interest and try to make a solution that would somehow automate a manually done work, like calculation, text generation, micro controllers managing and others. Domain-specific languages (DSLs) are tailored to a specific application domain, having a close relation with a particular niche and solving a specific problem, that cannot have relation to programming, engineering or science at all [1].

DSL offers substantial gains in ease of use compared with general-purpose programming languages in their domain of application. DSL development is hard, requiring both knowledge of the particular domain and language development expertise and skills. Not surprisingly, the decision to develop a DSL in taxes calculation requires a deep understanding of the financial part of the subject, as well an analysis of how it should be implemented from the programming point of view.

Although plenty of articles have been written on the development of DSL, the subject remains to be treated as an abstract one, having many solutions from different perspectives. To identify patterns in the decision, analysis, design, and implementation phases of DSL development, it is important to focus on both correctness of implementation of DSL, not forgetting about looking from the potential users' perspective. Audience needs targeting, uniqueness proposal and quality - the factors which will determine the future success in the DSL utilization [2-3].

1. DOMAIN DESCRIPTION

The domain that was analyzed about taxes calculation has some main specific fields, after which are calculated all types of taxes. These are: Social Fund Category, Individual contribution to the Pension Fund, Medical Insurance Employer, Medical Insurance Employee, Employ IT Sector, Syndicate Contribution, Personal exemption, Exemption for Disabilities, Exemption for the wife/husband. Some of them have more specific fields, such as the Social Fund Category, shown in the following partitions:

Contribution rate	**Category**
18%	For employers of the private sector, higher education institutions and health care institutions;
2. 3%	For employers of budgetary authorities / institutions and public authorities / institutions to self-management, except for higher education institutions and health care institutions;
26%	For employers of the private sector, higher education institutions and medical institutions (persons working under special working conditions);
33%	For employers of budgetary authorities / institutions and public authorities / institutions to self-management, except for higher education institutions and sanitary institutions (persons working under special working conditions);

For a better understanding, was represented the grammar for this specific language according to a very simple and textual program. Through it, was shown in details each feature of grammar.

A grammar of a domain specific language is characterized by four parameters:

G = { V_N, V_T , *P*, *S* },
where:
- V_N represents the set of non – terminal symbols;
- V_T represents the set of terminal symbol;
- *S* start point of our program;
- *P* represents productions.

Below was represented the grammar for the following code snippet:

```
Am 1 servicii {
   Serviciu_1 {
      Fond_Social – Sector_Privat ;
      Salariu_Brut = 25k;
      Cotizatie_Sindicate - OFF;
     }
   }
Calculeaza_Taxa_Pentru (Serviciu_1, lunar);
```

S = { sourceCode}

$\boldsymbol{V_T}$ = {Am, Serviciu, servicii, anual,Sector_Privat, Salariu_Brut, Cotizatie_Sindicate, Calculeaza_Taxa_Pentru, …, ON, k, 0, 1, 2 , 3, 4 , 5, 6, 7, 8, 9, - , =, : , ; , , , (,), {, }, _, . }

$\boldsymbol{V_N}$ = {<sourceCode>, <program>, <jobsDeclaration>, <serviceDetails>, <functionCall>, <symbol>, <keyword>, <identifier>, <block>, <statementsBlock>, <socialCategory>, < socialCategoryTypes>, <statements>, <state>, <salaryType>, <syndicationContribution>, <grossSalary>, < privateSector >, <syndicationContribution>, <dash>, <underLine>, <equal>, <semicolon>, <comma>, <roundBracketRight>, <roundBracketLeft>, <curlyBracketRight>, <curlyBracketLeft>, <digit> }

P = {

< sourceCode > → <program>, <program> → <jobsDeclaration> <symbol> <serviceDetails> <symbol> <functionCall>, <jobsDeclaration>→<keyWord> <identifier> <keyWord>,
<serviceDetails>→<block> | <block> < serviceDetails>, <block>→<statementsBlock>,
<statementsBlock>→<serviceBlockDeclaration> <symbol> <statements> <symbol>,
<serviceBlockDeclaration>→<keyWord> <symbol> <identifier>,
<statements>→<socialCategory><symbol><socialCategoryTypes>
<symbol><statements>→<salaryType><symbol><identifier> **k** <symbol>
< statements>|<syndicationContribution> < symbol > <state> < symbol >, <state>→ **ON** | **OFF**, <socialCategory>→ **Fond_Social,**
<socialCategoryTypes >→<firstCategory>,<firstCategory>→<privateSector>,
<salaryType>→<grossSalary >, < grossSalary >→ **Salariu_Brut**,
< privateSector >→**Sector Privat,** < syndicationContribution>→**Cotizatie_Sindicate**,
<keyWord>→**Serviciu** | **Servicii** | **Am** | **Calculeaza_Taxa_pentru**,
<symbol>→ <dash> | <underLine>| <equal>| <semicolon>| <comma>| <roundBracketRight>| <roundBracketLeft>| <curlyBracketRight> <curlyBracketLeft>,<identifier>→<digit>,<digit>→**0**|**1**| **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**, <functionCall>→**Calculeaza_Taxa_pentru** <symbol> <serviceBlockDeclaration> <symbol> <period> <symbol> <symbol>, <period>→ **lunar**

}

2. SEMANTIC AND LEXICON

The program will be constructed of two fields. The first one is the particular service declaration, where the user declares how many services they need to have and the details about each job, if they have more than one job. The second part consist of a function call, or method invocation, where the user asks the display of the results, in dependence of parameters that were previously introduced. The method should have at least two parameters.

Speaking about the lexical side, keywords are both lowercase and uppercase, in dependence of position. Thus, if the keyword is at the start of line, it means that the sentence is stating, so it should begin with uppercase, but if the keyword is in the middle or at the end of sentence, it must begin with lowercase.

Instead of white spaces between words, there must be the underline symbol (_), for example: HIGHER_EDUCATION_SECTOR. In the proposed grammar, it will be represented as one string or identifier. This rule is applied to non-terminal symbol, which derive terminal ones only and besides to this, is applied just for those non – terminal symbols that contain more than one word in that string.

For terminal symbols the rule is almost the same, the only difference is that there are not all letter uppercase, but just the beginning of the word (e.g. Sector_Privat). Other non-terminal symbols (except those that derive terminal symbols) follow the rule that first word starts with lowercase letter, and the next ones they must start with uppercase letter. The new DS language is case sensitive. Yet another point here, each block of the main program part must begin with curly left bracket "{" and end with curly right bracket "}". In the initialized field, when there is an assignment, the keyword must be followed by one of two type of assignment "– " or " = ". Each line within the block must end with semicolon symbol ";", as it means end of statement.

3. PARSE TREE GENERATED BY ANTLR

A parse tree or parsing tree or derivation tree or concrete syntax tree is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar. The term *parse tree* itself is used primarily in computational linguistic. In theoretical syntax, the term *syntax tree* is more common.

For the following code snippet, was generated the parse tree (Figure 1):

```
Am 1 servicii {
   Serviciu_1 {
      Fond_Social - Institutii_Bugetare ;
      Salariu_Total = 28k;
      Cotizatie_Sindicate - ON;
     }  }
Calculeaza_Taxa_Pentru (Serviciu_1, lunar);
```

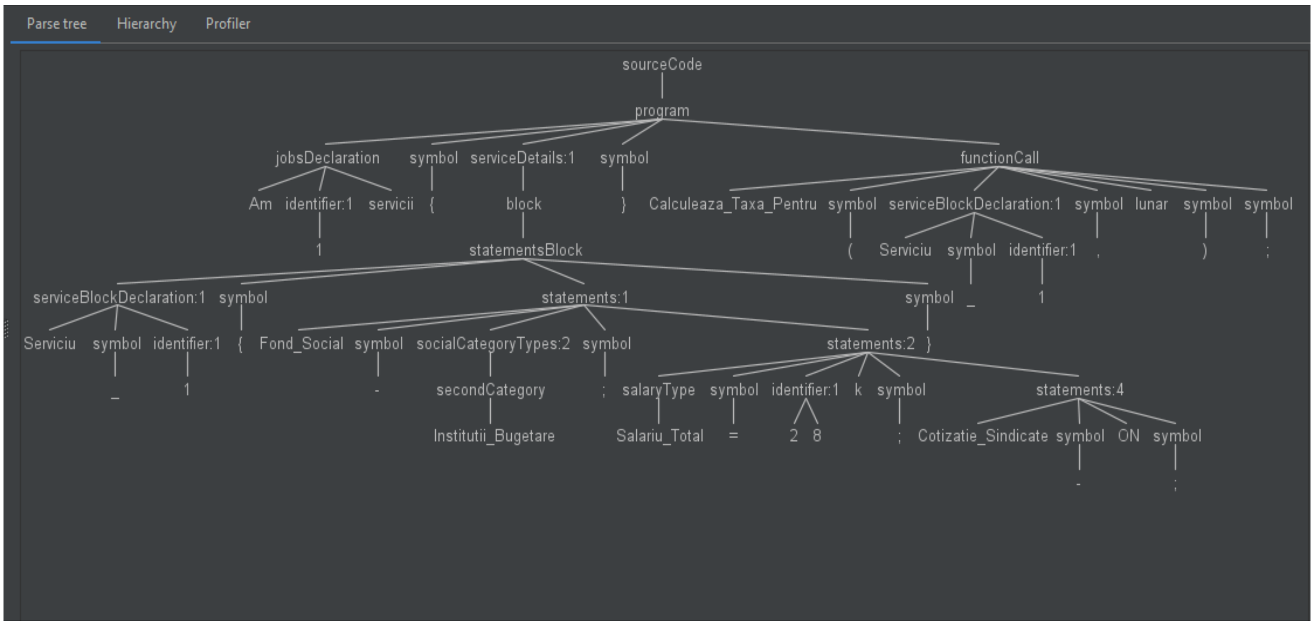
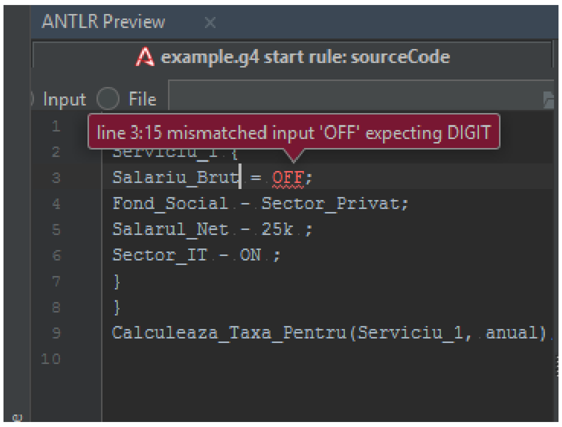


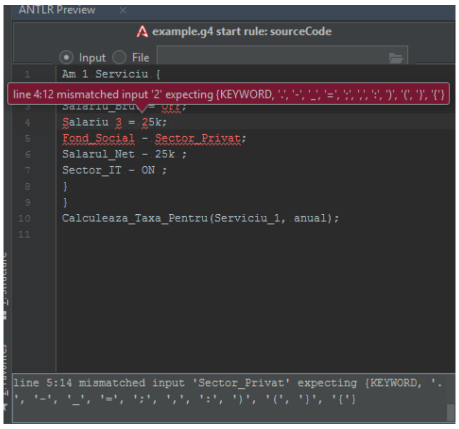
Figure 1. Parse Tree.

In the following example is represented a program, an incorrect one, meaning that it doesn't respect the rules of our grammar, therefore the scanner is able to recognize incorrect inputs:



line 3:15 mismatched input 'OFF' expecting DIGIT

```
Serviciu_1 {
Salariu_Brut = OFF;
Fond_Social - Sector_Privat;
Salarul_Net - 25k ;
Sector_IT - ON ;
}
}
Calculeaza_Taxa_Pentru(Serviciu_1, anual)
```



line 4:12 mismatched input '2' expecting {KEYWORD, '.', '-', '_', '=', ';', ',', ':', ')', '(', '}', '{'}

```
Am 1 Serviciu {
Salariu_Bru = OFF;
Salariu 3 = 25k;
Fond_Social - Sector_Privat;
Salarul_Net - 25k ;
Sector_IT - ON ;
}
}
Calculeaza_Taxa_Pentru(Serviciu_1, anual);
```

```
line 5:14 mismatched input 'Sector_Privat' expecting {KEYWORD, '.
', '-', '_', '=', ';', ',', ':', ')', '(', '}', '{'}
```

Figure 2 Mismatched input Error1. Figure 3 Mismatched input Error2.

Conclusion

Overall the process of analysis took an essential part in this particular research. Deepening the knowledge in the domain of taxes calculations allowed to understand what it should be changed and made otherwise as the already existent solutions. The approach of choosing the Romanian language in describing the syntax of the new DSL can be one of the points that make the DSL unique and user-friendly. From the other side, the baby steps made in the discovery mission of DSL implementation allowed to put some programming basics on all the abstraction. What is new? The target niche is very narrow and specific, having relation just on employees and takes in consideration the taxes paid to the government based on the salary.

Finally, it should be mentioned the advantage of the performed work, namely the contribution to the financial domain in our country, and also the technical approach of performing the calculations of taxes, the automatic process that is the base of the digitized future coming to us.

References

1. Voelter, M., Benz, S., Dietrich, Ch., Engelmann, B., Helander, M., Kats, L., Visser, E., Wachsmuth, G., et al. *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. CreateSpace Independent Publishing Platform, 2013.
2. Harrison, Michael A. *Introduction to Formal Language Theory*. Addison Wesley, 1978.
3. Aho, Alfred V., and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling*. Prentice-Hall, 1973.