DOMAIN SPECIFIC LANGUAGE FOR GEOMETRIC FIGURES AND BODIES REPRESENTATION

Mihai MOGLAN[1], Daniela MAZUR[1], Victor BALAN[1],
Antonina OSMĂTESCU[1*], Mihai AȘTIFENI[1]

[1] Technical University of Moldova, Faculty of Computers, Informatics and Microelectronics, Software Engineering, Group FAF-192, Chișinău, The Republic of Moldova

*Correspondent author: Antonina Osmătescu, osmatescu.antonina@isa.utm.md

Abstract: *This article represents an analysis of the implementation of a Domain Specific Language for Geometric figures and bodies representation. Analyzing both the technical and non-technical subjects, the article has the goal to describe in depth, step-by-step the process of 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 semantic rules and lexicon of the grammar for the DSL.*

Keywords: *DSL, language, grammar, semantics, syntax, geometric figures and bodies, sketching.*

Introduction

Domain-Specific Languages (DSLs) are becoming more and more important in software engineering. Tools are becoming better as well, so DSLs can be developed with relatively little effort [1].

Creating a domain-specific language, rather than reusing an existing language, can be worthwhile if the language allows a particular type of problem or solution to be expressed more clearly than an existing language would allow and the type of problem in question reappears sufficiently often. Pragmatically, a DSL may be specialized to a particular problem domain, a particular problem representation technique, a particular solution technique, or other aspects of a domain [1, 2].

Programming offers a wide range of possibilities that enable skilled people to 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 management 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 be unrelated to programming, engineering or science at all [3].

1. Domain description

This article presents the development of a specific language for representation of geometric figures and bodies. This idea was inspired by the fact that many pupils and students face difficulties with exact representation of difficult 2D or 3D geometrical figures from geometry problems.

The solution, for the problem described above, is building a domain specific language easy to use by students and teachers that will help both parties in the educational process. The old, ineffective way of visualizing geometrical shapes will change, making possible to see the shape from different perspectives.

It will be easier to use a DSL rather that a GPL from two reasons:

- Errors that a user can encounter using a GPL are more general and hard to understand by a person that has no experience in programming. While errors that can be generated by a DSL are built for a specific domain and are easier for domain experts to understand.
- Tokens used by a Domain Specific Language are intentionally chosen to be easily understandable by humans that are working in that domain. Therefore, the code will be readable and will require less lines to obtain the same result compared with time, logic and number of lines that are needed to obtain the same result with a General Purpose Language.

2. Grammar

For a better understanding, further is 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.

The DSL design includes several stages. First of all, definition of the programming language grammar $L(\mathrm{G}) = (\mathrm{S}, \mathrm{P}, \mathrm{V_N}, \mathrm{V_T})$:

- S - is the start symbol;
- P – is a finite set of production of rules;
- $\mathrm{V_N}$ – is a finite set of non-terminal symbol;
- $\mathrm{V_T}$ - is a finite set of terminal symbols.

In Tab. 1 are represented meta-notation used for specifying the grammar.

Table 1

Meta notation

Notation (symbol)	**Meaning**
<foo>	means foo is a nonterminal
foo	foo in bold means foo is a terminal
x*	zero or more occurrences of x
|	separates alternatives
→	derives
//	comment section

Below is represented the grammar for Domain Specific Language for geometric figures and bodies representation:

S = {<source code>}

$\mathrm{V_T}$ = {0.9, A.Z, a.z, true, false, Point, Line, Segment, Triangle, Square, Rectangle, Parallelogram, Trapezoid, Rhombus, Circle, Ellipse, Cube, Sphere, Cylinder, Cone, Pyramid, sketch, length, angle, radius, diagonal, median, bisector, vertex_name, angle_name, ., , , :, (,), _, “, ”, /}

$\mathrm{V_N}$ = {<source code>, <method name>, <methods invocation>, <decimal numeral>, <floating-point>, <digits>, <non zero digit>, <boolean literal>, <characters>, <string>, <string characters>, <identifier>, <type>, <numeric type>, <variable declaration>, <variables declaration>, <method invocation>, <argument list>, <expression>, <comments>, <comment>}

P = {

<source code> → <variables declaration><methods invocation>*<comments>*

<variables declaration> → <variable declaration>|<variables declaration ><variable declaration >

<variable declaration> → <type><identifier>

<type> → **Point | Line| Segment | Triangle | Square | Rectangle | Parallelogram | Trapezoid | Rhombus | Circle | Ellipse | Cube | Sphere | Cylinder | Cone | Pyramid | …**

<identifier> → (<character> | _) (<character> | <digits> | _) *

<character> → **a** | **b** | **c** |…| **A** | **B** | **C** | . | **Z**

<digits> → <digit> | <digits> <digit>

<digit> → **0** | <non zero digit>

<non zero digit> → **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

<methods invocations> → <method invocation>|<methods invocation><method invocation>

<method invocation> → <identifier>.<method name>(<argument list>*)

<method name> → **sketch** | **length** | **angle** | **radius** | **diagonal** | **median** | **bisector** | **vertex_name** | **angle_name** |**…**

<argument list> → <expression>|<argument list>,<expression>

<expression> → <numeric type> | <string> | <boolean literal>

<numeric type> → <decimal numeral> | <floating-point>

<decimal numeral> → **0** | <non zero digit> <digits>*

<floating-point> → <decimal numeral>.<decimal numeral>

<string> → "<string characters>*"

<string characters> → <characters>*<non zero digit>*

<characters> → **a** | **b** | **c** |…| **A** | **B** | **C** | . | **Z**

<boolean literal> → **true** | **false**

<comments> → <comment>|<comments><comment>

<comment> → // <string>

}

3. Semantic and lexicon

The program will be constructed from two fields. The first one is the particular variable declaration, where the user declares variables name and type. The second part consists of a method invocation, where the user asks to display the 2D or 3D figure, in dependence of the parameters that were previously introduced. The method may or may not have any parameters, for example *sketch()* method will have no parameters.

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

There are two numeric types in the DSL – decimal numeral and floating-point. The program differentiates between decimal and float by '.' (dot symbol).

Statements are executed one after another from top to bottom, similar to the scripting language.

4. Parse tree

A parsing 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 linguistics. In theoretical syntax, the term syntax tree is more common.

For the following code snippet, the corresponding parse tree was generated (Fig. 1):

```
Triangle var
var .setParameters (1, 4.5, 6)
var.draw(true)
```

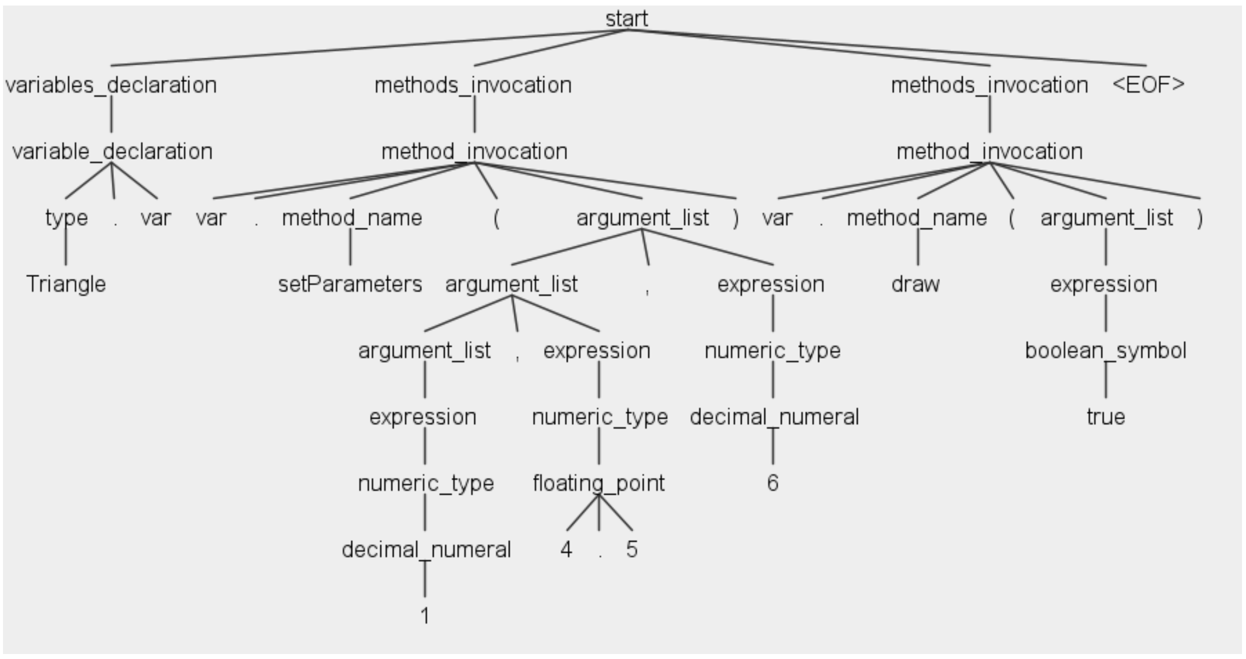

Figure 1. Parse Tree

Conclusion

This article meant to show the use of a DSL for Geometric figures and bodies representation, which will be designed in a way that will ease the process of drawing or sketching difficult geometrical figures. The DSL is designed to turn lines of code into geometric shapes. Compared to other similar languages, the product is directed towards students and teachers who may not be very familiar with programming. Only having variables and methods makes the language as basic as possible.

Finally, it should be mentioned the advantage of the performed work, because visualization of geometric figures and bodies is a serious problem at least in our country. This make students to hate geometry, even mathematics. Therefore, that simple language designed for this purpose will make geometry much more understandable and easy.

References

1. Domain-specific language, [accessed on 02.02.2021] Available: https://en.wikipedia.org/wiki/Domain-specific_language
2. MARTIN FOWLER, REBECCA PARSONS, *Domain Specific Languages*,2010 [accessed on 06.02.2021] Available: https://martinfowler.com/books/dsl.html
3. MARKUS VOELTER, *DSL Engineering Designing, Implementing and Using Domain-Specific Languages*, 2010-2013