DOMAIN SPECIFIC LANGUAGE FOR GEOMETRIC CALCULATIONS

Ion ROTARU[1], Alexandru GRAMA[1*], Ahmed AL HAJ[1], Mihai CORETCHI[1], Maxim ZADAROJNII[1]

[1]Technical University of Moldova, Faculty of Computers, Informatics and Microelectronics, Software Engineering and Automation Group FAF-211, Chisinau, The Republic of Moldova

*Corresponding author: Alexandru Grama

Coordinator: Catruc Mariana

Abstract: *This article analyzes the use of a domain-specific language for mathematics using geometric shapes and bodies. The paper analyzes both technical and non-technical topics with the intention of describing the Domain Specific-Language implementation process in detail, step-by-step, and highlighting the priorities and regulations. The fundamental characteristics of the Domain Specific-Language, as well as the fundamental semantic rules and vocabulary of the grammar for the Domain Specific-Language, were stated while highlighting the prerequisites and branches that should be included in the language.*

Keywords: *Domain-Specific Language, language, grammar, semantics, syntax, geometric calculations.*

Introduction

A Domain-Specific Language (DSL) is a computer language that's targeted to a particular kind of problem, rather than a general purpose language that's aimed at any kind of software problem. Domain-specific languages have been talked about, and used for almost as long as computing has been done [1].

A Domain specific language is usually less complex than a general-purpose language, such as Java, C, or Ruby. Generally, DSLs are developed in close coordination with the experts in the field for which the DSL is being designed. In many cases, DSLs are intended to be used not by software people, but instead by non-programmers who are fluent in the domain the DSL addresses.

There are two fundamentally different ways of how traditional code and DSL code can be integrated. The first one keeps DSL code and regular code in separate files. The DSL code is then transformed into programming language code by an automated code generator, or alternatively the program loads the domain-specific code and executes it. This first approach, with separated General Purpose Language (GPL) and DSL code is termed external DSLs. Think of SQL, MATLAB as an example of an external DSL [2].

1. Domain description and analysis

A Domain-Specific Language for geometry could address a wide range of domains related to geometry, including: Computer-Aided Design, Computer Graphics and Animation, Robotics, Architecture and Construction and Computational Geometry. The proposed DSL will address Computational Geometry, which means that it could be used to solve mathematical problems related to geometry, such as calculating intersections, distances, or areas between shapes, or performing geometric transformations. This could include tools for solving geometric algorithms or defining geometric primitives.

A programming language called a Domain-Specific Language is created to handle a particular issue domain. A DSL might be developed to aid in the description and manipulation of geometric forms in the case of geometry, enabling more succinct and expressive code.

Here are some particular ways that a DSL may help geometry:

- Code that is clear and easy to read: a geometry DSL might utilize a syntax that is designed specifically for the description of geometric forms.
- Higher-level abstractions: a geometry DSL might offer more expressive and understandable code by providing higher-level abstractions for frequent geometric notions like points, lines, circles, and polygons.
- Greater type checking: a geometry DSL should include tighter type checking to avoid frequent errors like adding points to lines or computing the intersection of forms that are not intersecting.
- Code reuse: a geometry DSL might offer reusable parts for routine tasks like calculating a shape's area or locating the closest point on a line.
- Integration with additional tools: a geometry DSL may be used with additional tools, such as visualization libraries or computer-aided design (CAD) software, to provide smooth integration across various phases of a geometric modeling pipeline.

In conclusion, a geometry DSL may offer a more effective and expressive approach to interact with geometric forms, making it simpler to build, modify, and examine complicated models.

A DSL for geometry can help solve several problems related to expressing and manipulating geometric concepts in a more intuitive and efficient way. Some of these problems include:

- Expressing geometric concepts: a DSL for geometry can provide a more natural and intuitive way to express geometric concepts, such as points, lines, curves, and surfaces. This can make it easier for users to create and manipulate geometric shapes and models.
- Improving accuracy: geometry can be complex, and small errors in geometric calculations can lead to significant inaccuracies in the final results. A DSL for geometry can help improve the accuracy of geometric calculations by providing built-in functions and methods for common geometric operations.
- Increasing productivity: a DSL for geometry can help users be more productive by automating repetitive tasks and reducing the need for manual calculations. This can save time and reduce the risk of errors.
- Facilitating collaboration: a common language for expressing geometric concepts can facilitate collaboration between different stakeholders, such as engineers, designers, and architects. This can help ensure that everyone is on the same page and working towards the same goals.

Overall, a DSL for geometry can help solve several problems related to expressing and manipulating geometric concepts, improving accuracy, increasing productivity, and facilitating collaboration

DSL in geometry can be used by various users who need to express geometric concepts or perform geometric computations in a domain-specific context. Some potential users of DSL in geometry are:

- Mathematicians: mathematicians who work on geometry or related fields can use DSL in geometry to formalize geometric concepts and theorems.
- Engineers: engineers who work on fields like computer-aided design (CAD), computer graphics, or robotics can use DSL in geometry to define and manipulate 2D or 3D geometric models.
- Architects: architects can use DSL in geometry to specify and manipulate geometric models of buildings, including their shapes, dimensions, and orientations.
- Designers: product designers and industrial designers can use DSL in geometry to create and manipulate 3D models of their designs.
- Scientists: scientists who work on fields like physics or chemistry can use DSL in geometry to represent and simulate molecular structures and their interactions.
- Educators: educators can use DSL in geometry to teach geometry in a more interactive and engaging way, by allowing students to create and manipulate geometric models.

These are just some examples of potential users of DSL in geometry, but there could be many other users depending on the specific domain or application.

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 detail each feature of grammar.

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

V_N – is a finite set of non-terminal symbol;

V_T - is a finite set of terminal symbols.

P – is a finite set of production of rules;

S - is the start symbol;

In Table 1 are meta-notations 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

S = {<source code>}

V_T = {START, 0.9, A…Z, a...z, true, false, Point, Line, Segment, Triangle, Square, Rectangle, Parallelogram, Trapezoid, Rhombus, Circle, Ellipse, Cube, Sphere, Cylinder, Cone, Pyramid, length, angle, radius, diagonal, median, bisector, vertex_name, angle_name, area, perimeter, volume, ., , , :, (,), _, “, ”, /, +, -, *, ^, sin, cos, ctg, tg, END }

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>, <expression>, <comments>, <comment>}

P = {<source code> → **START***<variables declaration><methods invocation>*<comments>***END**

<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** | **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> → **length** | **angle** | **radius** | **diagonal** | **median** | **bisector** | **vertex_name** | **angle_name** | **area** | **perimeter** | **volume** | **…**
<argument list> → <expression>|<argument list>,<expression>
<expression> → <numeric type> | <string> | <boolean literal>
<numeric type> → <decimal numeral> | <floating-point>
<floating-point> → <decimal numeral>.<decimal numeral>
<decimal numeral> → <digits>*
<string> → "<string characters>*"
<string characters> → <characters>*<digit>*
<boolean literal> → **true** | **false**
<comments> → <comment>|<comments> <comment>
<comment> → // <string>
}

3. Semantic and lexicon

The program will be split into two sections. The user defines the name and type of the variables in the first step, the private variable declaration. The second section consists of a method invocation in which the user requests that specific parameters, such as an object's area or the volume of a 2D or 3D figure, be calculated based on other values that have already been input.

There must be an underscore (_) in place of white space between words, such as in vertex_A. This will be shown in the suggested grammar as a string or identifier. This rule only applies to those non-terminal and terminal symbols that include more than one string word. It also only applies to non-terminal symbols that stem from terminal ones.

In DSL, there are two different sorts of numbers: floating point and decimal. The software is used to distinguish between decimal and float by "." (dot symbol). Similar to linguistic scripts, instructions are carried out sequentially from top to bottom.

4. Parse Tree

A parsing tree or concrete syntax tree is an ordered, rooted tree that describes the syntactic structure of a string according to a context-free grammar. Computational linguistics is the main field in which the term "parse tree" is used.

The phrase "syntax tree" is more prevalent in theoretical syntax. The matching parse tree for the following sample of code was created (Fig. 1):

START
Square ABCD
ABCD.setParameters (4)
ABCD.perimeter()
ABCD.area()
END

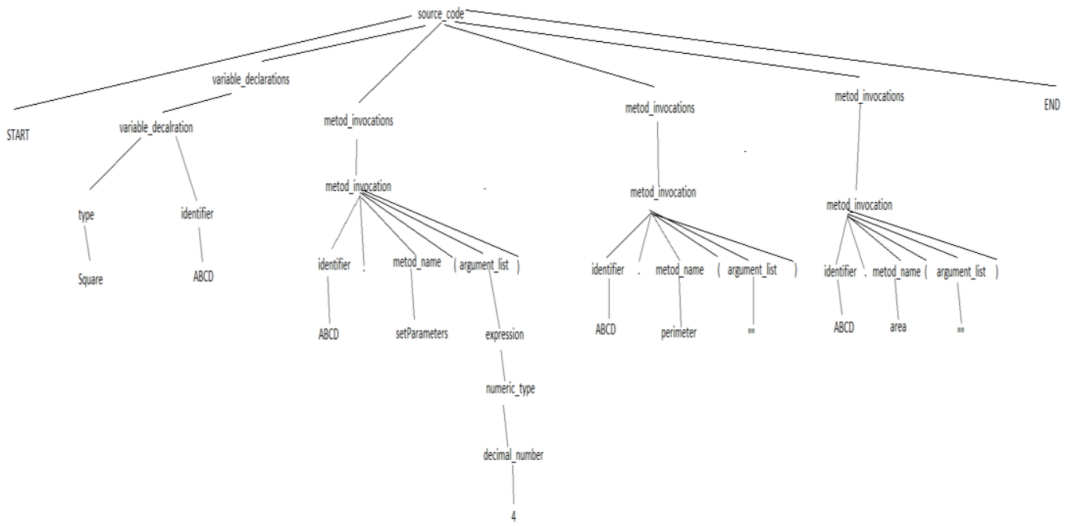

Figure 1. Parse Tree

Conclusion

This article's goal was to demonstrate how to utilize a DSL for geometric calculations. The DSL will permit to make the easier measurement and calculations of area, perimeter, volume or other geometric numbers. Lines of code may be converted into geometric computations using DSL. The product is intended for students, professors, and engineers who may not be highly experienced with programming, in contrast to other languages of a similar nature. Just having variables and methods makes language sound as simple as possible.

As the measurement and computation of geometric figures and bodies is a significant issue, the benefit of the task must be stated last. Students begin to despise math and geometry as a result. Hence, geometry will be more simpler to learn and easy thanks to the straightforward language created for this aim.

References

1. MARTIN FOWLER, *Domain-Specific Languages Guide*, 28 Aug 2019 [accessed on 15.02.2023] Available: DSL Guide (martinfowler.com)
2. Domain-Specific Languages, [accessed on 15.02.2023] Available: What are Domain-Specific Languages (DSL) | MPS by JetBrains