DOMAIN SPECIFIC LANGUAGE FOR CHEMISTRY

Darinela ANDRONOVICI[1*], Daniel CORMAN[1], Dumitraș MĂMĂLIGĂ[1], Iuliana TROFIM[1], Oleg VOLVOV[1]

[1]Technical University of Moldova, Faculty of Computers, Informatics and Microelectronics, Department of Software Engineering and Automatics, group FAF-191, Chisinau, Moldova

*Corresponding author: Darinela Andronovici, andronovici.darinela@isa.utm.md

Abstract: *This article presents the grammar of a domain-specific language (DSL) that is made for chemical equations. Furthermore, this paper explains how the DSL, which is being developed, will work, what functions will be implemented and how this language will ease the studying process for pupils from high school or for those who are interested in chemistry.*

Keywords: *domain-specific language, grammar, chemical equations.*

Introduction

Due to the fact that domain-specific languages are created for supporting a particular set of tasks from a specific domain, their importance in software engineering has grown in the last years. A domain-specific language (DSL) is a programming language with a higher level of abstraction optimized for a specific class of problems [1]. Using a DSL can bring benefits in term of reduced time to market, productivity, quality assurance, development cost. Well-designed domain-specific languages enable the easy expression of problems, the application of domain-specific optimizations, and dramatic improvements in productivity for their users.

The article aims to develop a graphical language for balancing and visualising chemical equations.

Chemistry is a very important and widely used domain in modern society. Some of the most important products that we are using in our life, things that helped us to overcome mass deaths because of epidemics and different kind of illnesses are medical drugs and medicines, developed in a domain called pharmaceutics would have been impossible without chemistry. Due to its importance, the chemistry course was introduced in middle school and high school program.

Pupils meet many problems during the educational process, because they have a lack of experience that leads to the appearance of many mechanical mistakes. In order to solve a chemical equation, pupils must equal the coefficients of the substances that are entering in reaction, which most of the time, is a challenging process.

The main purpose of a domain-specific language for chemical equations is to make the educational process easier and more pleasant than it was before. By using this DSL, their work will become easier and they will be certain that the answers are correct without having to verify them. The main parts that define a domain-specific language are: grammar, data types, semantic rules, types of assignment, scope rules and how to invoke a method [2].

Specifications of the DSL

The basic functionalities of the domain specified language are:

- Print an equation.
- Balance an equation.
- Solve an equation.
- Visualize an equation.

Considering the functionalities, in the DSL, the user will be able to do four commands: PRINT, BALANCE, SOLVE, SHOW and the arguments of the specified commands are variables.

Reference grammar

For a better understanding of the grammar are used special notations. (Table 1. Meta-notations).

Table 1.

Meta-notations

<foo>	means that foo is a nonterminal
foo	(in **bold** font) means that foo is a terminal i.e., a token or a part of a token
x?	means zero or one occurrence of x
x*	means zero or more occurrences of x
x^+	means one or more occurrences of x
|	separates alternatives

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

- S – is a start symbol;
- P – is a finite set of production of rules;
- V_N – is a finite set of non-terminal symbols;
- V_T – is a finite set of terminal symbols [3].

V_N = {<commandStatemenetList>, <commandStatemenet>, <printCommand>, <balanceCommand>, <solveCommand>, <showCommand>, <varCommand>, <declReaction>, <declPartial>, <varName>, <EQUAL>, <PLUS>, <reaction>, <partialReaction>, <side>, <molecule>, <element>, <DIGIT>, <DIGITS>, <LOWERCASE>, <UPPERCASE>, <number>}

V_T = {PRINT, BALANCE, SOLVE, SHOW, VAR, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 , {, }, (,) , + , =}

S = {<program>}

P = {<program> → <commandStatemenetList>

<commandStatemenetList> → <commandStatement>$^+$

<commandStatement> →<printCommand> | <balanceCommand> | <solveCommand> | <showCommand> | <varCommand>

<printCommand> → **PRINT (**<varName >**)**

<balanceCommand> → **BALANCE (**<varName >**)**

<solveCommand> → **SOLVE(**<varName >**)**

<showCommand> → **SHOW(**<varName >**)**

<varCommand> → <declReaction> | declPartial

<declReaction> → **VAR** <varName> <EQUAL> **{** <reaction> **}**

<declPartial> → **VAR** <varName> <EQUAL> **{** <partialReaction> **}**

<varName> → <LOWERCASE>$^+$

<reaction> → <side> <EQUAL> <side>

<partialReaction> → <side>

<side> → <side> <PLUS> <side> | <number>? <molecule>

<molecule> → (<element> <number>?)$^+$ | <molecule> **(** <molecule> **)** <number>

<element> → <UPPERCASE> <LOWERCASE>?

<number> → <DIGIT> <DIGITS>*

<UPPERCASE> → **A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z**

<LOWERCASE>→ **a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |q | r |s | t | u | v | w | x | y | z**
<DIGIT> → **1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
<DIGITS> → **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
<EQUAL> → **=**
<PLUS> → **+**

Table 2.

Example of code

Input	**Output**

VAR a = {Ni(OH)2+HNO3=Ni(NO3)2 + H2O}
BALANCE(a)
	$Ni(OH)_2 + 2\ HNO_2 \rightarrow Ni(NO_3)_2 + 2\ H_2O$

The rule to start a language grammar is engage the parser first. Any rule in grammar can act as a start rule for the following parser. Start rules do not necessarily consume all of the input, because they consume only as much input as needed to match an alternative of the rule.

Semantic Rules

This set of rules place additional constraints on the set of valid programs besides the constraints implied by the grammar. The compiler will explicitly check each of these rules and as well as other languages, our DSL will use the console to communicate with the user, therefore all the errors will be specified in the console or by some error fields within the code where it occurred.

A program in the domain-specified language will consist of chemical equations. Each equation has at least one side, that if formed from molecules delimited by the equal sign. If the chemical reaction is balanced, in front of the molecule can be positioned a number (the coefficient). A molecule is a group of two or more elements, also it represents a structure that contains information about elements from the periodic table followed by a number, which is optional. Respectively, an element has to consist of at least an upper case letter, that can be followed of a lower case letter, because in the periodic table, there are two types of symbols: uppercase letter or uppercase and lowercase letters.

Data types

In this DSL, string is the only one main data type, due to the fact that a chemical equation consists of numbers, letters and symbols.

Scope Rules

The DSL has very simple scope rules:
- a variable used in future calculations must be declared first;
- variables may be declared at the beginning of the program/everywhere in the program;
- only predefined methods can be used;

Type of assignment

The symbol for the assignment is “=” and it is used for declaring variables following the rule from the grammar of the domain specified language: <varName> = { <reaction> } or <varName> = { <partialReaction> }.

Method Invocation

In order to verify if the grammar was written correctly and the invoked methods show the needed result, ANTLR was used. ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. From a grammar, ANTLR generates a parser that can build and walk parse tree [4]. In Figure1 is presented the graphical form of the parse tree for the input specified in Table2, which has <program> as start symbol and then it derives in <commandStatementList> and so on.

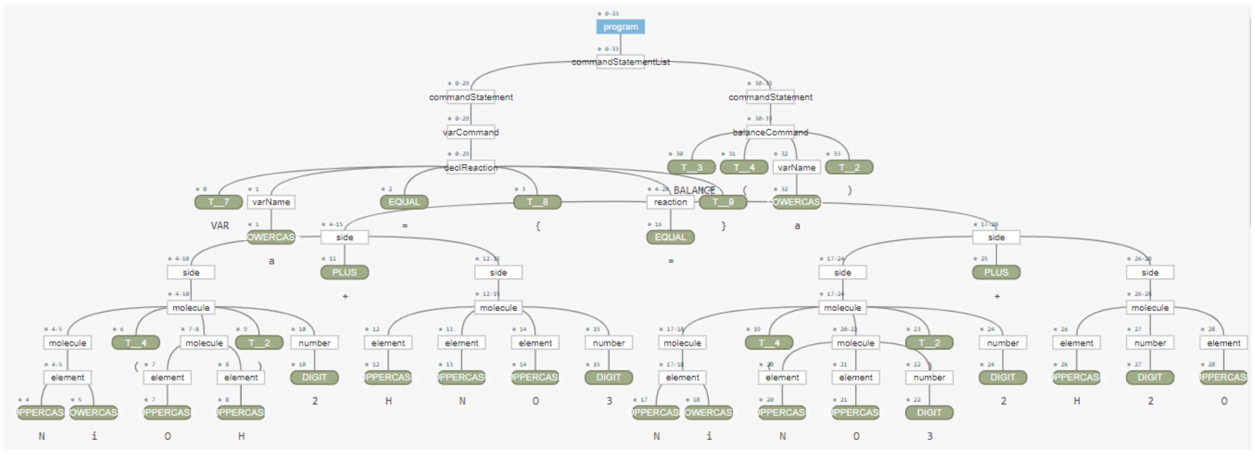

Figure 1. The parse tree of the input from Table 2

Conclusions

For developing a new DSL is necessary to follow some rules: to have grammar, sematic, parse rules and others. This paper intended to show the use of a domain specified language for chemistry, which will be designed in a way that will ease the process of solving and balancing chemical equations. The target niche is very narrow and specific, having relation just on the pupils from middle and high school. Taking into considerations the points mentioned above, it is a great timesaving tool and easy to handle, which is an advantage for those who are facing problems in solving chemical equations. As a consequence of this research, the conclusion made was that the biggest advantage of the performed work, namely the contribution to the educational process.

References:

1. JET BRAINS, *Domain-Specific Languages* [online] [accessed 22.02.2021] Available: https://www.jetbrains.com/mps/concepts/domain-specific-languages/.
2. MARKUS VOELTER, *DSL Engineering. Designing, Implementing and Using Domain-Specific Languages* [online] [accessed 22.02.2021] Available: http://www.voelter.de/dslbook/markusvoelter-dslengineering-1.0.pdf.
3. RANALD CLOUSTON, *Non-Deterministic Finite Automata and Grammars* [online] [accessed 23.02.2021] Available: https://cs.anu.edu.au/courses/comp2600/2013/lectures/NFA.pdf.
4. *What is ANTLR?* [online] [accessed 24.02.2021] Available: https://www.antlr.org/.