DEVELOPING A DOMAIN-SPECIFIC LANGUAGE FOR AN AUTOMATION SYSTEM

Margareta DIACENCO, Mariana ONICA, Cristofor FIȘTIC, Andrei LUPAȘCU

Technical University of Moldova

Abstract: *This article presents the grammar and the lexical parser for a domain specific language that is made for the automation system. Additionally, this paper explains how the DSL, which is being developed, will work, what functions will be implemented and how this language will help people interact with the automation systems.*

Keywords: *Domain-specific language, programming language, context-free grammar, parser, semantic, routines, syntax, Raspberry Pi, automation system*

1. Introduction

The starting point is analysing the classical programming languages. The idea of such languages (like C, Java and Python) is widely known not only in the technical community but also in every activity domain. They play an inevitable role in the process of software development. "A programming language (PL) is an artificial computer language designated to express computations that can be performed by a machine. PLs can be used to create programs that control the behaviour of a machine, to express algorithms precisely, or as a mode of human communication " [5].

The major part of the developed programming languages describe computation in an imperative style, mostly as a sequence of commands and support the object-oriented paradigm of programming (OOP) which is the definition of general-purpose programming languages, meaning you can code software applications for many various application domains. In order to be able to be processed by automation systems, they need to have their syntax and semantics specified. The syntax of programming languages is usually specified by means of context-free grammars or from context-free grammar derived Backus-Naur forms (BNFs). The next step is to parse the resulting grammar by a corresponding type of parser. The role of a parser (or syntactic analyser) is to determine whether the program is syntactically correct and to build a parse tree of the program. Parsers call lexical analysers (or scanners), which recognize lexical elements (tokens) such as keywords, identifiers, constants, operators and separators.

As the topic of this project is based on developing a DSL for automatic systems, the next step is to define this specific system.

Automated system operations (ASO) is the set of software and hardware that allows computer systems, network devices or machines to function without any manual intervention. ASOs allow computer systems to work without a human operator physically located at the site where the system is installed. Automated system operations are a part of the automatic system control where the processes are completely automated with the help of control loops and special logic. Automated system operations are also known as lights-out operations. Automated system operations are a combination of both software and hardware that is designed and programmed to work automatically without the need for a human operator to provide inputs and instructions for each operation. Automated operations reduce the complexity of labour-intensive tasks. Some of the most widely used applications that incorporate ASO include scheduling, management of console messages, backup and recovery, printing services, performance tuning, network monitoring and bug detection [6].

2. Basic concepts of a domain-specific language

A domain-specific language (DSL) is a computer language designed to be used in a particular field to solve a specific class of problems. They can be developed to be *programming languages dedicated to a particular problem domain* or *specification languages dedicated to a particular problem domain*.

The main idea of a DSL is to offer means which would allow the specialist of a particular domain to compute the solution using idioms and terms that he operates with. The automation system described in this article uses such keywords as *temperature*, *altitude*, *left*, *right*, etc. Other examples of widely-used DSLs are: Mathematica, a language used for mathematical manipulations, SQL - for relational database queries and manipulations, etc.

The representations of a DSL can be both textual and graphical, but the latter one is more popular due to having more tools for its designing, such as Generic Eclipse Modeling System or Microsoft Visual Studio DSL; the textual representation is more productive.

DSLs also can be *internal* or *external*. Internal ones are extensions of currently existing general-purpose programming languages, for example, sets of functions, data structures and conventions. Oppositely, external languages are being entirely created to serve a specific purpose. It is in the format of a text file, which is then interpreted by a matching system or compilator.

A well-designed DSL has several advantages:

- it is *human-readable* - which means it is understandable by humans and easier to use (in comparison with XML languages, sometimes);
- it is *machine-processable* because it has a defined syntax and semantics.

DSLs are mainly used in software engineering to help experts from different domains to have a solution t their particular problem, without them needing to know programming. It is not an easy task, because the DSL developer must have knowledge of both the field in question and of computer languages and language processors, compilers and interpreters [3].

3. The DSL for an automation system

The Domain-specific language is created for an automation system that is built on Raspberry Pi. First, is created the grammar for the specified language: G = {S, V_T, V_N, P}. The grammar ”G” of the language for automation system includes the start point of the application ”S”, terminal nodes ”V_T”, non-terminal nodes ”V_N” and the production ”P” where all the nodes were combined in a correct logic.

S = {<Program>}

V_T = {FollowLine, Temperature, Altitude, Pressure, Start, Forward, Right, Left, Back, Sleep, If, for, while, then, int, (,), [,], , , 0, 1,…, 9, A, B,...Z, a, b...,z}

V_N = {<Program>, <Set of commands>, <Command>, <Arrow movement>, <Environment data>, <condition>, <initialization>, <incrementation>, <operation>, <variable>, <alpha_num>, <variable>, <alpha>, <number>, <digit>, <boolean>}

P ={ **<Program>** → <Set of commands>

<Set of commands> → <Environment data><Set of commands>* |<Arrow movement><Set of commands>* |<Command><Set of commands>*

<Environment data> → <Temperature><Data temperature>* |<Altitude><Data altitude>* |<Pressure> <Data pressure>*| <Temperature><Data temperature>*, <Altitude><Data altitude>*, <Pressure><Data pressure>*

<Arrow movement> → Forward<Arrow movement>*| Right<Arrow movement>*| Left<Arrow movement>* |Back<Arrow movement>*

<Command> → FollowLine <Command>*| Temperature <Command>*| Altitude <Command>*| Pressure <Command>*| If (<condition>) then <Set of commands> <Command>*| for “(“<initialization> <condition> <incrementation>”)” ”{“<Set of commands>*”}” | while “(“<condition>”)” “{<Set of commands>* <incrementation>”}”| Start <Command>*| Forward <Command>*| Right <Command>*| Left <Command>* | Back <Command>* | Sleep<Command>*

<condition> → [<Environment data> |<variable |<number>] <operation> [<Averagedata> |<variable> |<number>] <operation>* [<Average data>|<variable>|<number]*

<**initialization**> → <variable>”=”<variable>

<**incrementation>** →<variable>”++” |<variable>”--” |<variable>”+”<digit> |<variable>”-”<digit>

<Data temperature> → ‘(‘C’)’ | ‘(‘K’)’

<Data altitude> → ’(‘m’)’ | ‘(‘km ‘)’

<Data pressure> → ’(‘Pa’)’ | bar’)’

<number> → <digit>|<digit>*

<boolean> → true | false

}

<operation> → <operationrealtional> | <conditionaloperation>

<operationrealtional> → > | < |<= | >= |!= | =

<conditionaloperation> → &&| ||

<variable> → <alpha> | <alpha_num>*

<alpha_num> → <alpha>|<digit>

<alpha> → A|B … Z|a|b … z|_

<digit> → 0|1|2|3|4|5|6|7|8|9

This DSL is based on a deterministic context-free grammar in order for it to be easier to be parsed. The syntax is based on natural language constructions commonly used in a particular context in the problem area. Otherwise, it will be a risk that the user would not understand how to use this DSL. For that reason, having knowledge of the problem domain is very important. The automation system will be easier to control with DSL as users will have access to the built-in ready to use functions. This system has a function like **Pressure**, that will show the user data from the pressure sensor in the system and a command **FollowLine**, that makes the automatic system to follow a line. Also, the DSL offers users the possibility to make triggers with the **If** command to check some conditions. Then, the automatic system needs to do something. In the Domain-specific language, we provide loop instructions like **while** and **for**, so the automation system is able to generate the same actions without human assist until some condition is met. This will allow the user to automate his activities. The parsing strategy is a top-down tree which is supported by ANTLR4. So, after the created grammar, a parser, lexical analyser and semantic routines are made, all written in Python, as Raspberry Pi supports the Python language.

To explain how this DSL works, the user will take a command, for example, **Temperature(k).** This command shows the user data from the temperature sensor in Kelvin degrees. In the grammar, it is explained in productions how it is able to see the data from this command. To test the correct logic of the grammar, it was introduced in the ANTLR4 command line: Fig. 1 ”grun ProjectGrammar program -tree Temperature(k)” to see the parsing tree of this instruction Fig. 2 and be sure that the application concept works well.

```
C:\Grammar>grun ProjectGrammar program -tree
Temperature(k)
^Z
(program (setofcommands (environmentdata Temperature (k))))

C:\Grammar>
```

Fig. 1. Instruction wrote in ANTLR4 that parses the grammar to display the path for Temperature. command.

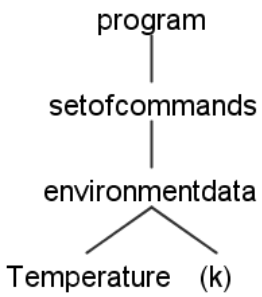

Fig. 2. The parse tree for Temperature(k) command.

In the next step, the system receives tokens that activate the temperature functionality for it, which returns data from the sensor and only after that activates the Kelvin function, which will transform the data received from the precedent function in Kelvins and will return the converted data in a human-readable format to the command-line, so the user will be able to see his desired information. The user must know programming in order to create these types of functions, but with the help of this DSL, he will be able to just type the desired command, which will give that intuitive feature of the DSL.

Conclusion

The intention of this paper was to show how the concepts of a DSL in automation helped to understand how to create such an intelligent system using the grammar, which explains all the necessary components for the application and how DSL can help people to use high technology utilities in their work without having programming skills. The ANTLR4 is a powerful parser generator aimed to correctly present the grammar for the created automation language. The role of the programmer in relation to a programming language can be analogical to the role of a trained specialist in an industry in relation to a DSL. That said, the language will be made as simple and as intuitive as possible. With the developed DSL, it will be feasible to create a language that will be easy to be used by specialists in the problematic domains.

References

1. Markus Volter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart Kats, Eelco Visser, Guido Wachsmuth, *DSL Engineering.* Designing, Implementing and Using Domain-specific languages, http://dslbook.org, 2010-2013.
2. Boris Sedov, Sergey Pakharev, Alexey Syschikov, Vera Ivanova, *Domain-Specific Approach to Software Development for Microcontrollers* Proceeding of the 17th conference of FRUCT association, 2015.
3. Ľubomír Dedera, *Domain-specific languages for control systems*, 2010, p. 41-42.
4. *Conferința Tehnico-Ştiinţifică a Colaboratorilor, Doctoranzilor şi Studenţilor din 16-18 noiembrie 2017*, Vol. 1, 547 p., 2017.
5. https://en.wikipedia.org/wiki/Programming_language
6. https://www.techopedia.com/definition/31065/automated-system-operations-aso
7. https://tomassetti.me/antlr-mega-tutorial/#creating-a-grammar
8. https://code.visualstudio.com/
9. https://tomassetti.me/antlr-mega-tutorial/#starting-with-python
10. https://tomassetti.me/antlr-mega-tutorial/#the-python-way
11. https://tomassetti.me/antlr-mega-tutorial/#testing-with-python
12. https://tomassetti.me/antlr-mega-tutorial/#tips-and-tricks