ROAD MARKUP DOMAIN SPECIFIC LANGUAGE

Maxim NICHIFOROV[1], Ion ȚURCANU[1], Nicolae BASSO[1*], Marinela BRÂNZAEANU[1], Nichita SOROCHIN[1]

[1] Technical University of Moldova, Faculty of Computers, Informatics and Microelectronics, Software Engineering, Group FAF-191, Chișinău, Republic of Moldova

[*]Corresponding author: Nicolae Basso, basso.nicolae@isa.utm.md

Abstract: *This article describes a Domain Specific Language for helping to create a road marking plan. Subsequent, this paper has the purpose of explaining that by creating a domain-specific language (DSL) which is being focused on a certain task, some of the previous actions will become automatized and will consume much less time to be implemented. Further, this material clarifies how the DSL will work, what are its base functionalities and how it is built.*

Key words: *road, markings, domain-specific language, grammar, parse tree.*

Introduction

A domain-specific language (DSL) is a high-level software implementation language that supports concepts and abstractions that are related to a particular (application) domain [1]. The effort needed by an end-user to rapidly write correct programs using the produced DSL is the main factor that causes them to be so popular and widely implemented [2]. Traffic rules are a major safety factor that require all the humans to be in charge of. Sometimes they are well determined and intuitive but there are times when they are confused and not obvious at first sight. This is why an idea for constructing a language that can intuitively help mark roads and set the priorities into an intersection has appeared. Our DSL's main purpose is to make some base functions for planning the traffic rules on the streets. These functions may potentially be used by AI autopilot to predict markings and set of rules where the road does not have them at all or by traffic police to ease their work [3]. It is considered, that by dint of this kind of language, it will be easier for road builders and designers to do their work, and in the same way, this will increase quality and speed of projects implementation in future. Most of the road-marking job is routine, that can be automated and people doing it can shift the responsibility about trivial things on computers. Thereby, in further workers will focus on other more important things, which can be missed during the routine implementation. In addition, the DSL can be standardized, thus, the projects implemented with it will be in one general standard, and these projects can be easily saved, stored and transmitted to everybody who needs it.

Reference grammar

Road Markup DSL grammar's definition:

L(G) = (S, P, VN, VT), where:

- S – start symbol;
- P – finite set of production of rules;
- VN – finite set of non-terminal symbols;
- VT – finite set of terminal symbols.

S = {<program>}

VN = {<program>, <statement>, <createElement>, <invokeProcedure>, <createRoad>, <createIntersection>, <createSign>, <finalize>, <setter>, <draw>, <autoMarkup>, <setter>, <dimensional>, <restrictional>, <setWidth>, <setLength>, <setCrossingAngle>, <setLongtitudinalMarking>, <setPriority>, <setIntersectionType>, <setMovementType>, <createRoad>, <createIntersection>, <createSign> }

VT = {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 , {, }, (,) , +, =, ;, _ ,\n, \t, <SET_WIDTH>, <SET_LENGTH>, <SET_CROSSING_ANGLE>, <SET_PRIORITY>, <SET_INTERSECTION_TYPE>, <SET_MOVEMENT_TYPE>, <DRAW>, <AUTO_MARKUP>, <ROAD>, <INTERSECTION>, <SIGN>, <NEWLINE>, <STMTEND>, <INTERSECTION>, <SIGN>, <INT>,<LOWERCASE>, <UPPERCASE>, <DIGIT>, <LETTER>, <CHAR>, <WORD>, <WHITESPACE>, <TEXT>, <VARNAME>, <ROAD_NAME>, <INTERSECTION_NAME>, <SIGN_NAME>, <SEMICOLOR>, <MARKING_TYPE>, <MOVEMENT_TYPE>, <INTERSECTION_TYPE>, <PRIORITY> }

P = { <program> → <statement> <(STMTEND)+>
<statement> → < createElement > | < invokeProcedure >
<createElement> → < createRoad > | < createIntersection > | < createSign >
<invokeProcedure> → <finalize> | <setter>
<finalize> → <draw> | <autoMarkup>
<setter> → <dimensional> | <restrictional>
<dimensional> → <setWidth> | <setLength> | <setCrossingAngle>
<restrictional> → <setLongtitudinalMarking> | <setPriority> | <setIntersectionType> | <setMovementType>
<createRoad> → <ROAD>
<createIntersection> → <INTERSECTION>
<createSign> → <SIGN> }

The full example of the grammar and production rules is in references [4].

Semantics and semantic rules

Semantic rules come as an addition above the language rules, which are going to check the code user provides our DSL with for logical integrity.

Data types

There are 3 main data types in the "Road Construction" DSL, which are:

- ROAD;
- INTERSECTION;
- SIGN;

"ROAD" responds for roads, "INTERSECTION" for intersection, and "SIGN" for all types of road signs.

Lexical analysis

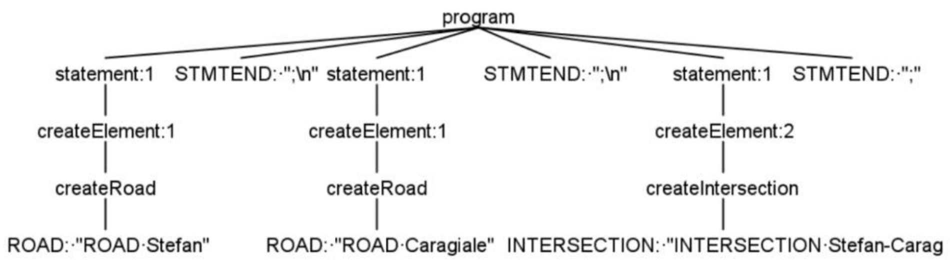
Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code [5].

Basic control structures

Our core control structure is built using top-down or tree flow: complex expressions are decomposed and, thus, simplified till they reach the "atomic" logical size.

Example of script and parsed tree

```
ROAD Stefan;
ROAD Caragiale;
INTERSECTION Stefan-Caragiale;
SetPriority(Stefan, Stefan-Caragiale, Main);
AutoMarkup();
Draw();
```



(A fuller, more expressible parse tree and script sample can be found at [6,7])

Figure 1. Parse tree

Specifications of DSL

The main functionalities of the DSL are:

- Creating elements such as: Road, Intersection and Sign.
- Automated road and markup properties' generation.
- Drawing given project.
- Setting properties of the objects (such, as length, width or priority (for roads)).

Conclusion

The purpose of this paper is to show the concept of the DSL directed to make road marking easier. In addition, this DSL can help with digitalization of the road marking projects, so they can be built, stored and transmitted with certain and global standard, which can be used by every user having and computer and knowing this DSL syntax. Our team was making researches in this domain to understand with what kind of problems face people that are involved in process of designing the road marking and we tried to generate intuitive language than will not require thorough and deep learning and can be easily read and written.

In general, the domain of urbanistic is wide field of activity and there is much work to do about automation despite it is not popular about creating certain software for it. We think that this language can be great start for developing software in this domain in a more global context than it is at this moment.

References:

1. VISSER, E. WebDSL: A Case Study in Domain-Specific Language Engineering. *In: R. Laemmel, J. Saraiva, and J. Visser, editors, Generative and Transformational Techniques in Software Engineering (GTTSE 2007) Volume 5235 of Lecture Notes in Computer Science*, p.2.
2. KOSAR., PABLO E. A preliminary study on various implementation approaches of domain-specific language. In: Information and Software Technology, Volume 50, Issue 5, April 2008, p.390.
3. PAUL K., JURGEN V. A Domain Specific Language for Source Code Analysis and Manipulation. In: ACM SIGPLAN Notices 25(6), June 2000, pp.26-36.
4. Grammar in .g4 format, https://bitbucket.org/pbl-elsd/road-language/src/dev/ANTLR/src/ANTLR -grammars/RoadConstruction.g4
5. Lexer, https://bitbucket.org/pbl-elsd/road-language/src/dev/ANTLR/src/base/gen/RoadConstruction/RoadConstructionLexer.java
6. GitBucket reporitory of the project: DSL Script Sample, https://bitbucket.org/pbl-elsd/road-language/src/dev/ANTLR/src/tmp/ScriptExample.txt
7. A fuller parse tree example, https://bitbucket.org/pbl-elsd/road-language/raw/91bebfc9cb22559b91164fd4655dff550b706cb6/ANTLR/gen/images/parseTree.png