CHAT MESSAGE MANIPULATION LANGUAGE

Paula POPESCU[1*], Victor CARAGIU[1], Alexandr BOICO[1], Ion GAVRILIȚA[1], Octavian GRECU[1]

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

*Corresponding author: Paula Popescu, popescu.paula@isa.utm.md

Abstract. *In this article has described a Domain Specific Language for message manipulation. The Chat Message Manipulation Language has the purpose to offer an alternative way to manage messages. This DSL will create more interactive communication and will reduce boring texting. In addition, it will learn people basic code syntax. The grammar of this domain specific language is simple so that it cannot confuse the user with many different functions and tricks. This language is focused on gathering input function from the user, then on the server it analyses the syntax and semantics and server application produce a JSON data structure to be sent as an output.*

Key words: *communication, mark-up language, domain specific language (DSL), parse tree, lexer.*

Introduction

The most important domain in the human life is the communication one. Humans are social person and they need to speak with their neighbourhoods every day [1]. In the 21st century when all activities transferred to online platforms, communication also transferred on online format. People use many messengers to speck with friends, neighbourhoods and even with teachers. The Chat Message Manipulation Language has the scope to improve and make more interactive online communication throw messages. It also focusses on style and interaction related functionalities, such as defining message type. There are many message applications, but there is no one widely used which support writing scripts in the message section to manipulate messages. This language is meant to offer tools for text editing and generation for your chat application.

Reference grammar

The DSL design includes some important steps. First of all, definition of the programming language grammar L(G) = (S, P, V_N, V_T), [2] where:

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

S = {program}.

V_N = {<program>, <statement>, <manipulation>, <delimiter>, <method>, <object>, <btn>, <qstn>, <gft>, <domain>, <transform>, <change>, <two_parameters>, <parameter>, <id>, <number>, <letters>, <string>}.

V_T = {+, button, question, gift, birthday, newyear, christmas, upper, lower, repeat, rightcut, leftcut, edit, replace, a.z, 0.9, ,, ., “, ”, -, !, ?, ‘, ’}.

P = {

<program> → <statement>$^{+}$,

<statement> → <string> <delimiter> <string> | <manipulation> | <manipulation> <delimiter> <string> | <string> <delimiter> <manipulation> | <object>,

<manipulation> → <string>.<method> | <id>.<method>,

<delimiter> → +,

<method> → transform>() | <change>(<number>) | edit(<string>) | replace(<two_parameters>),
<object> → <btn> | <qstn> | <gft>,
<btn> → button([<string>] <string>),
<qstn> → question(<string>)[<string>][<string>]{ [<parameter>]<parameter> },
<gft> → gift(<domain>),
<domain> → birthday | newyear | christmas,
<transform> → upper | lower,
<change> → repeat | rightcut| leftcut,
<two_parameters> → {[<parameter>] <parameter>}$^+$,
<parameter> → <statement> | <string>,
<id> → # <number>,
<number> → 0.9$^+$,
<letters> → a.z$^+$,
<string> → { a.z | 0.9 | , | . | ? | ! | " | " | ' | ' | - }$^+$
}

Semantics and semantic rules

This DSL in made for non-programming users too, the semantics have very few limitations and rules. One of the rules that the user should follow is to write parameters of specific data types, accordingly to those specified in the grammar.

The program can start with a string or an object. If it starts with a string, it can be directly sent to the second user or transformed using the pre-defined functions.

In case the user wants to use an object, he can start describing only the object, without any more messages outside the object. If there are any, they will be omitted. The functionality the objects offer is that different or same type of particular objects can be nested, according to the grammar. This language is case-sensitive and all the spaces are taken in the account.

Data types

There are two data types: int and enum. The enum data type is used for available domains classification. In addition, there are two data structures: string and JSON. The data is created by the keyboard input, or by referencing and updating old data. The data manipulation is done by the built-in functions, which are called by the user input. The data of JSON type is generating from the parse tree and is used to represent the components and their content.

Lexical analysis

As the lexical analysis require, [3] further is specify the following details: how to handle comments, strings, errors and other specific characteristics of this lexer:

The CMML language doesn't provide the possibility to write comments, as there is no need to. Every character is treated as a part of the string which represents the message, until a method is identified. The delimiter's meaning is the start of the string's section, meant to be modified by the method, when the user doesn't want to apply the method for all the string.

The errors are sent to the user as a pop-up, in the upper part of the chat section, and the message is not posted. If the script doesn't have errors it is posted, for the user to check if it is working as intended. In order to send the messages or apply the modifications, the user have to also click the *send* button.

Basic control structures

The control structure which is capable of branching the execution flow is the question object: question(<string>)[<string>][<string>]{[<parameter>]<parameter>}. This component is able to change the output, depending on the second user's input.

Example of script and parsed tree

The following script represents a nested object inside another one of the same type, this script will be write by the user in the input field of the messenger app. In the C language, the nested function can access all the variables of the containing function that are visible at the point of its definition, which is called lexical scoping [4].

```
question(Let’s go for a walk!)
[Accept][Decline]{
  [question(In the park?)[Accept][Decline]{
      [super…]   Where do you want?
  }]
ok.}
```

Table1.

Generated tokens by the lexer from the source-code

Token	Lexeme
qstn	question
string	Let’s go for a walk!, In the park?
answer_1	Accept
answer_2	Decline
parameter_1	super…
parameter_2	ok., Where do you want?
identifier	{, }, [,].

The code is tokenized and converted into tokens through the process of lexical analysis in Table 1 and into the parse tree through the process of syntactic analysis [5]. After conversions, the translator’s output is a data structure, which keeps the tree’s structure for the interpreter.

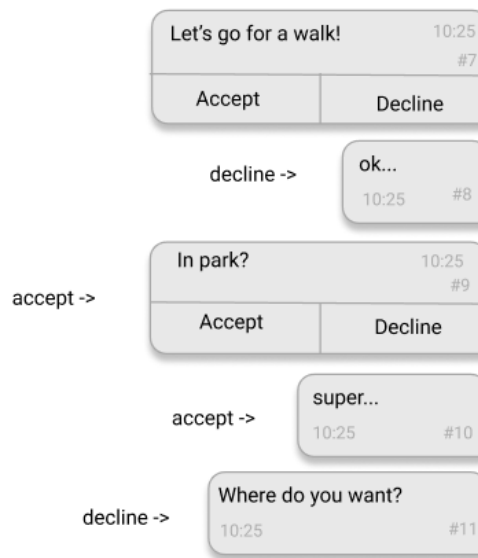

Figure 1. The resulted scenarios of the conversation with components specified by the script.

In Figure 2 below is presented the parse tree of the script presented above. This ordered, rooted tree represent the syntactic structure of the script [6] according to the grammar of CMML. It reflect the syntax of the input. In this way it is easier to visualize the way this language acts. The parse tree is constructed according the relations defined in the grammar.

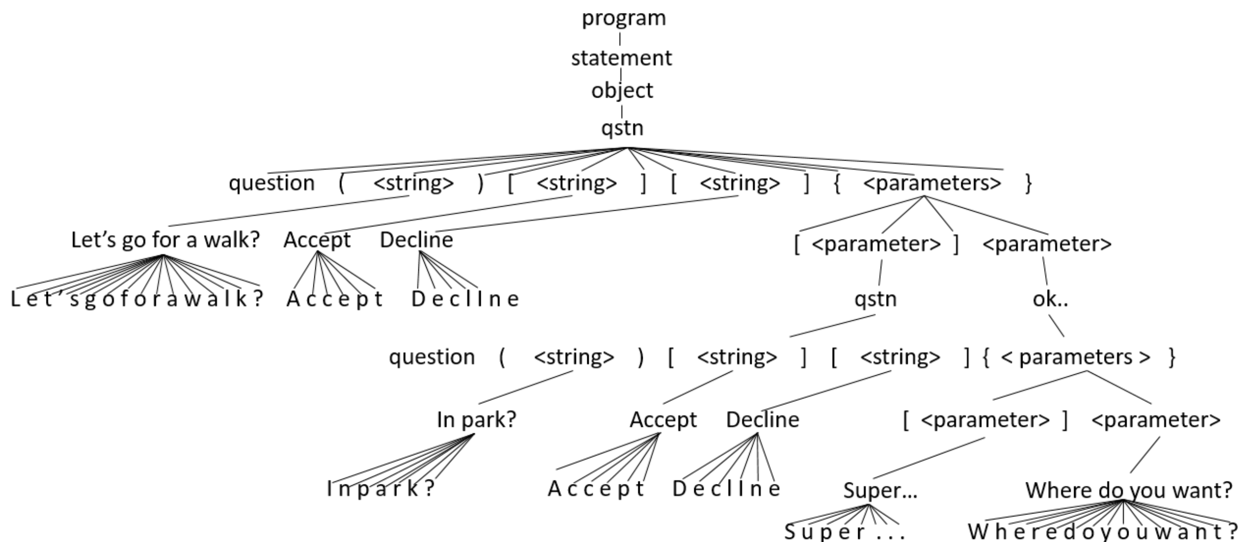

Figure 2. Parse tree

Conclusion

The intention of this paper was to show how the concepts of Domain Specific Languages can help to improve and make more interactive online communication throw messages. Also, how 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 ordinary life without having programming skills. The syntax of the language is quite simple, so even a child can introduce himself in it and to begin to learn programming from an early age.

To top it off, this technology is here to transform the way people have ever looked at message applications and create a different and interactive way of communication.

References

1. MATEI, S., BALL-ROKEACH, S. J., *Real and Virtual Social Ties: Connections in the Everyday Lives of Seven Ethnic Neighborhoods* [online]. 2021, 03 [access 02.03.2021]. URL: https://journals.sagepub.com/doi/abs/10.1177/0002764201045003012
2. JOHN, E. H. and RAJEEV, M. J. D. U. *Introduction to Automata Theory, Languages, and Computation*, 2001, p. 169.
3. APPEL, A. W. *Modern Compiler Implementation in C*, 1998
4. Nested Functions, [online]. [access: 27.02.2021]. URL: https://gcc.gnu.org/onlinedocs/gcc/Nested-Functions.html
5. Tokenization and Abstract Syntax Tree. [access: 27.02.2021]. URL: https://geekeefy.wordpress.com/2017/06/07/powershell-tokenization-and-abstract-syntax-tree/
6. AHO, V., ULLMAN, D. *The Theory of Parsing, Translation and Compiling Parsing of Series in Automatic Computation*, vol. 1, Prentice-Hall, 1972