IMAGE MANIPULATION DSL

Marius BÎTCĂ[1*], Anastasia GAVRILIȚĂ[1], Ilcenco EUGENIU[1], Daniel BRIȚCHI[1], Nicolae EVSTAFIEV[1]

[1]*Technical University of Moldova, Faculty of computers, informatics and microelectronics, Department of Software Engeneering and Automatics, FAF-191, Chisinau, Moldova*

[*]Corresponding author: Bîtcă Marius, bitca.marius@isa.utm.md

Abstract. *Image manipulation is not a hard task to perform because of all the existent tools. However, when having a specific number of transformations to be performed on a group of files, the task is harder to perform. Therefore, this paper will further describe a domain specific language to help perform some particular actions on multiple images with an easy to understand and read syntax.*

Keywords: *grammar, processing, syntax, parser, lexer, ANTLR.*

Introduction

Nowadays, in the age of rapidly developing technologies, there is growing need to use a field-specific programming language. Unlike general-purpose languages, a domain-specific language consists of elements and relationships that directly represent the logic of the problem space. The benefits of the DSL are that it can be understood by their customers, that the code generated from it is reliable, and that the system can be rapidly updated if the customers' requirements change.

Moreover, this interpretation of programming language will help to focus the ideas and logical power on the specific problems that DSL is created for. Therefore, one of the biggest issues to find a domain that will give all spectrum of disponible action to cover all necessities.

Finite Automata

Finite Automata is an abstract machine that can be in exactly one of a finite number of states at any given time. The Finite Automata can change from one state to another in response to some external inputs, the change from one state to another is called a transition. A transition is a set of actions to be executed when a condition is fulfilled or when an event is received. The job of an FA is to accept or reject an input depending on whether the pattern defined by the FA occurs in the input [1].

External DSL and image processing?

- An external DSL is a language that is parsed independently of the host general-purpose language: good examples include regular expressions and CSS.
- Image processing is a method to perform some operations on an image, to get an enhanced image or to extract some useful information from it. Digital image processing techniques help in manipulation of the digital images with computers.

Why image-manipulation DSL?

Domain-specific languages increase the level of abstraction. The proposed DSL shall be as close to the photo-editing domain as possible. Unlike fully detailed programming languages, the proposed DSL will bring several commands together, which will make a smaller, easy to write and easy to maintain language. It shall not require anything other than basic image-editing knowledge – no previous programming skills or advanced Photoshop experience. Users will be able to apply changes to multiple images at a time, by mentioning them altogether and by writing the commands only once.

The proposed DSL enables users to easily select multiple photos from the device to work with and perform basic, quick manipulations without accessing complex interfaces.

Reference Grammar

The proposed DSL's grammar is described in Fig. 1: terminal and non-terminal symbols to be used, data types and data structures, actions and semantic rules etc.

- S – the start symbol
- P – a finite set of production of rules
- V_n – a finite set of non-terminal symbols
- V_t – a finite set of terminal symbols

<foo>	means foo is a nonterminal.
foo	(in **bold** font) means that **foo** is a terminal.
[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.

V_n = { <program>, <declaration>, <action>, <export>, <open_file>, <open_folder>, <id>, <file_path>, <folder_path>, <action_type>, <export_action>, <image_type>, <int_literal>, <digit>, <alpha>, <alpha_num> }

V_t = { ‘,’, ‘open’, ‘[‘, ‘]’, ‘.’, ‘;’, ‘(‘, ‘)’, ‘crop’, ‘rcrop’, ‘rotate’, ‘flip_x’, ‘flip_y’, ‘resize’, ‘brightness’, ‘contrast’, ‘saturation’ }

S = { <program> }

```
      <program>   →   <declaration>+ <action>* <export>

  <declaration>   →   <open_file> | <open_folder>

    <open_file>   →   <id> = open ( <file_path> ) ;

  <open_folder>   →   <id> `[` `]` = open ( <folder_path> ) ;

       <action>   →   <id> . <action_type> ;

       <export>   →   <id> . <export_action> ( [ <file_path> ] ) ;

  <action_type>   →   crop ( <int_literal>, <int_literal>, <int_literal>, <int_literal> )
                  |   rcrop ( <int_literal>, <int_literal>, <int_literal>, <int_literal> )
                  |   rotate ( <int_literal> )
                  |   flip_x ( )
                  |   flip_y ( )
                  |   resize ( <int_literal> [, <int_literal> ] )
                  |   brightness ( <int_literal> )
                  |   contrast ( <int_literal> )
                  |   saturation ( <int_literal> )

<export_action>   →   save

    <file_path>   →   “ <alpha>+ . <image_type> “

   <image_type>   →   png
                  |   jpg

  <folder_path>   →   “ <alpha>* “

  <int_literal>   →   <digit>+

          <id>    →   <alpha> [ <alpha> | <digit> ]*

       <alpha>    →   a | b | … | z | A | B | … | Z

       <digit>    →   0 | 1 | 2 | … | 9
```

Figure 1. Detailed description of the proposed DLS's grammar

Computational model

The proposed image-processing DSL will interpret code line-by-line and execute the commands. In other words, each line represents a separate command for the DSL to execute when open. Images or the sets of images to be processed will be declared at the beginning of each program, with the help of variables.

Data types

In programming, data types are used to define what values or variables should be considered for several actions.

At the low level, the image editor DSL's users will operate on images using strings (for names, links and assignment) and numbers (decimals or integers) as the actions' parameters (resolution, RGB values etc.)

Data structures: image or array of images

The user creates a variable of type image which points towards the image(s) to be open, then they would perform actions on the images with the help of the line-commands.

Functions

Functions, also called actions in the proposed DSL, are all the image manipulations you can perform on the object that was declared.
Here are some actions and a short description to be implemented:

- crop – crop a part of the image;
- rotate – rotate the image with a positive number of degrees;
- flip – flip the image on the x or y axis;
- resize – resize the image with given dimensions;
- brightness – increase or decrease the image brightness in percentages;
- contrast – increase or decrease the image contrast in percentages.

There is another type of actions that are created for image declaration and for image export, this are: *open* and *save*, in the Fig. 2 and Fig. 3 one can see how these actions will be used.

```
img = open("image.png");
img.resize(500, 200);
img.save("new_image.png");
```

Figure 2. Syntactically correct program.

To demonstrate that this is indeed a syntactically correct example that can parse a line of code with the created grammar (Fig. 1).

```
<declaration> → <open_file>

→ <id> = open ( <file_path> ) ;

→ img = open ( <file_path> ) ;

→ img = open ( " <char>* . <image_type> " ) ;

→ img = open ( " image.png " ) ;
```

Figure 3. Parsing the declaration part for the program.

Lexer and Parser

After deciding on the final grammar, the next step was to translate it into a parser and lexer generator, called ANTLR. Its function is to take a grammar that specifies a language as input and generate the source code for a recognizer of that language.

The chosen base language in which all the image manipulation functions were created is Golang, as a modern and flexible language.

The grammar defined in ANTLR generated parse trees for different examples. Fig. 5 depicts the parse tree for the program in Fig. 4.

```
img = open("image.png");
img.resize(500, 200, 300);
img.save("new_image.png");
```

Figure 4. Syntactically incorrect program.

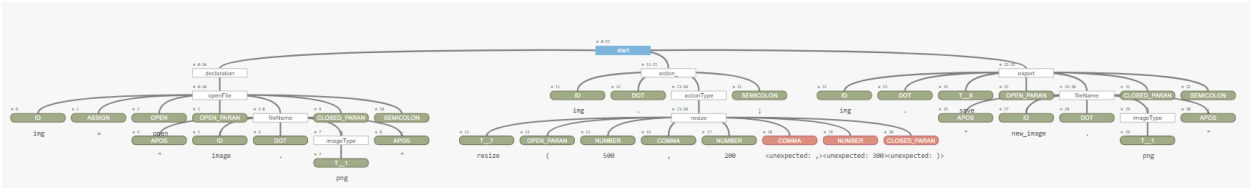

Figure 5. Parse tree with unexpected nodes.

In Fig. 5 there are 3 red nodes, this means that the program is not syntactically correct because of a misspelling of the grammar in the resize action. The grammar definition of resize action allows only 1 or 2 arguments in the parenthesis, but the one in the image has 3.

Conclusions

There are no other external DSLs specifically designed for processing and editing images. However, there are some general-purpose DSLs for web-pages, like the well-known CSS. The proposed DSL will include only the commands that address image manipulations, nothing more than that. Therefore, not only will it represent a convenient tool for quick actions, but also, this means that no special courses will be needed to learn how to use the proposed DSL. Clients can discover its functionalities as they use it, for each command will be as close to the English language as Possible. On the other hand, possessing some basic image-editing knowledge will be a great advantage for the users of this DSL.

References

1. RANDAL, C., *Computation and Formal Systems*. [online]. [accessed 15.02.2021] Available: https:/www.cs.rochester.edu/u/nelson/courses/csc_173/fa/fa.html