DOMAIN SPECIFIC LANGUAGE FOR AUTOMATIC DOCUMENT PROCESSING

Maxim BUGĂESCU, Dan CEBAN, Dinu CROITORU, Andreea COVALEVSCHI*, Nicolae GHERMAN

[1]*Department of Software Engineering and Automatics, group FAF-202, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chișinău, Moldova*

*Corresponding author: Andreea Covalevschi, andreea.covalevschi@isa.utm.md

Abstract: *This article presents the grammar of a domain-specific language (DSL) for automatic document processing. It is described the way in which the DSL being developed will work and how it will facilitate the process of working with documents with image file extensions which contain unstructured data, and the process of visualizing data.*

Keywords*: domain-specific language, grammar, document, data.*

Introduction

A domain specific language (DSL) is a programming language that is created with a specific purpose for a specific need [1]. A DSL may be developed to be used on a particular platform, to solve a specific software problem, or business challenge that cannot be productively addressed by using general Programing Languages.

The article's purpose is to develop a language for processing data from documents with .jpeg or .png extension.

At the current time of living, the information keeping and processing has taken a major step, data being converted into digital formats. This article brings an alternative for an extant problem, extracting and manipulating unstructured data in big quantities. It is consuming a lot of time and follows a monotonous pattern, therefore, it fits for a job for a computer. An approach would be to let the representatives of these branches use a language that would extract certain data from digital photo formats and would allow to perform operations on the obtained data and to visualize it. There are different benefits included in automated information extraction which makes this activity more manageable as well as easier to work in specific domains.

The quantity of data is developing violently and helpful outcomes are appearing every day and would increase in future [2]. The main purpose of a DSL for automatic document processing is to simplify the process make it less time consuming. By using this DSL, people wouldn't have to copy manually the desired data from an image or from a collection of images. Grammar, types of assignment, data types, semantic and scope rules and how to invoke a method – are some of the most important properties which define a DSL.

Language overview

The basic computations that the proposed DSL performs consists of scanning the data from images or collections of images, abstracting from the general-purpose language for using which the user needs knowledge of programming principles, arranging the unstructured data in form of structured data - tables, performing computations on those tables – such as computing the mean value, standard deviation, mode, sum, minimum element value, max. element value and sum of a specific column. Also, plotting the dataset in different ways and saving the obtained dataset in .csv or .xls formats

The main data structures are the images (or folders of images). The DSL doesn't include an explicit data declaration that will affect the computation of the main program.

The DSL supports Sequence control structure, by which the lines of code are executed in the order of appearance in the program. But selection and repetition would also be supported in case the user doesn't want to limit himself using only the built-in functions of our DSL. In these cases, for

delimiting the code inside conditionals and loops, indentation would be used. The first line of code cannot have indentation. Indentation is mandatory to define blocks of statements. The number of spaces must be uniform in a block of code.

Grammar Design

Table 1

Meta Notations

Notation	Meaning
<symbol>	symbol is a nonterminal
symbol	**symbol** is a terminal
[expression]	The expression is optional (when the brackets are bold, they are part of terminal symbols!)
{expression}	expression is iterative
|	Separates alternatives

The grammar of the proposed DSL in short form:

P = { <program> → <set of affirmations>
<set of affirmations> → <affirmation>
| <set of affirmations><affirmation>
<affirmation> → <assignment>
| <plot>
| <save>
<assignment> → <variable> = <data extraction>
|<variable> = <computation>
<plot> → **lmplot** <df>
| **scatterplot** <df>
| **displot** <df>
| **barplot** <df>
| **countplot** <df>
<data extraction> → **extract** <string list> **from** <image>
|**extract** <string list> **from** <folder>
<computation> → **compute** <measure> **on** <df> [**each** <int> **rows**]
| <num> **percent** <range> **rows** <df>
<save> → **save** <df> **as** <format>
<image> → **image** <string>
<folder> → **folder** <string>
<measure> → **mean** | **stdev** | **max** | **median** | **mode** | **sum**
<format> → **csv** | **xls**
<df> → <variable>
| <variable><string list>
<range> → **[**<int> **to** <int>**]**
| **[**<int>**]**
<string list> → **[**<string items>**]**
<string items> → <string>
| <string>**,** <string items>
<string> → [<char>]
<variable> → <alpha>
| <alpha>{<digit>}
| <alpha><variable>
| <df>
<num> → <float>

		| <int>
<int>	→	<digit>{<digit>}
<float>	→	<int>.<int>
<char>	→	**a** | **b** | ... | **z** | **A** | **B** | ... | **Z** | **0** | **1** | **2** |…| **9** |…| **-** | **_** | **+** | **!** | **.** | **,** | **$** | **%** | **&** | **(** | **)**
<alpha>	→	**a** | **b** | ... | **z** | **A** | **B** | ... | **Z**
<digit>	→	**0** | **1** | **2** | **3** |…| **9**

}

Example of parsing a program:

a = **extract [“**name**”] from image “**xyz.jpg**”**
compute mean on a
lmplot a
save a **as csv**

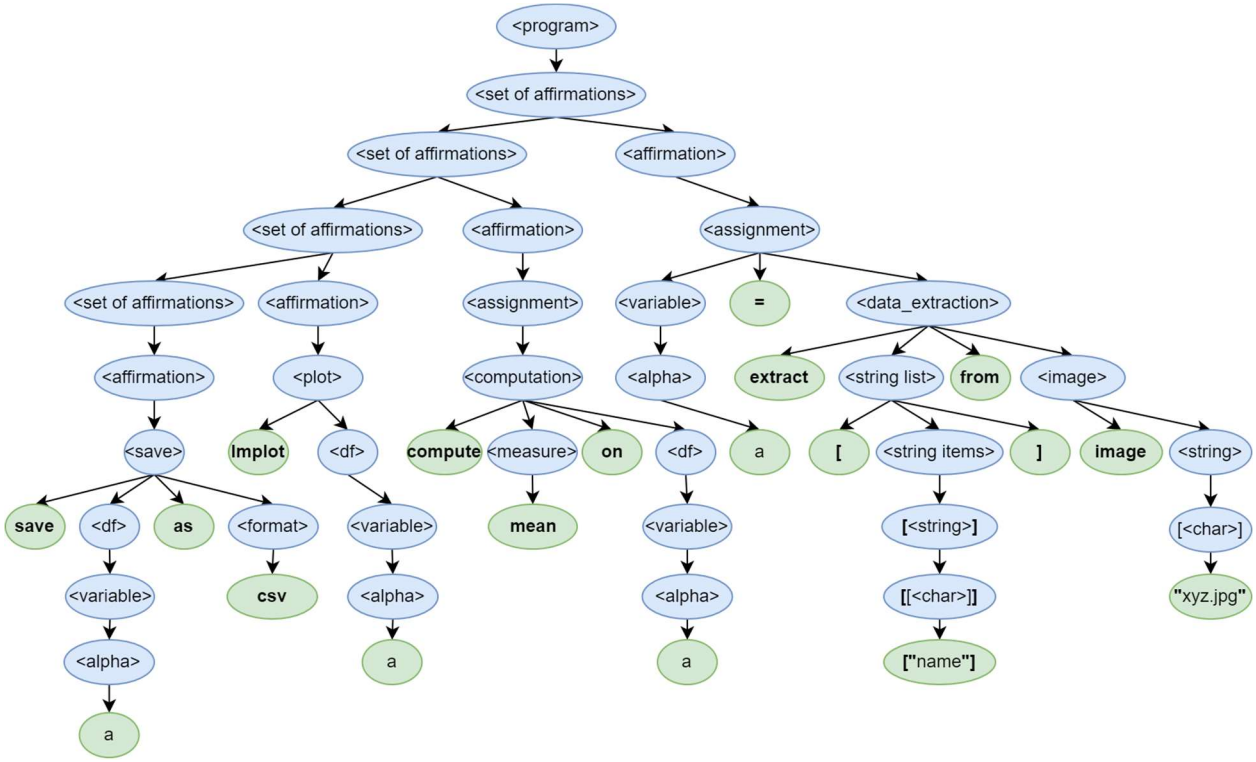

Figure 1. Parsing tree

Conclusion

A DSL makes doing specific tasks a lot easier. It has simple routes to implement the parts of that domain that are ‘common process’. This paper introduces a DSL concept which is centered around automatic processing of documents in image format. The target are the employees of small to medium businesses which have to deal with lots of physical documents with unstructured data. Taking into consideration everything said above, it is a time saving tool and easy to use. After having defining the grammar and language specifications, the next step would be using the real tools to build the DSL itself.

References:

1. MANAGOLI, G. *What developers need to know about domain-specific languages* [online], 2020. [accessed 14.02.2022]. Disponible: https://opensource.com/article/20/2/domain-specific-languages
2. CRISTEA, M. *Digital archiving reduces companies’ costs with more than 20%* [online], 2021. [accessed 16.02.2015]. Disponibil: https://business-review.eu/business/digital-archiving-reduces-companies-costs-with-more-than-20-218376