DOMAIN SPECIFIC LANGUAGE FOR DOCUMENT AUTOMATION

Irina TIORA*, Liviu MOCANU, Nicolae ȘEVCENCO, George VRAGALEV, Nicu SAVA

Department of Software Engineering and Automatics, group FAF-203, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chișinău, Moldova

*Correspondent author : Tiora Irina irina.tiora@isa.utm.md

Abstract. *This article describes the design of a domain-specific language (DSL) that targets document automation field. Additionally, this text explains how the DSL will accomplish its main functionality, the implementations steps and how this language will ease the document autocompletion process in different fields. Subsequently, this paper delves into the syntaxis, functionality and creation of the language.*

Key words: *domain-specific language(DSL), grammar, document automation, autocompletion, form, template function*

Introduction

A domain-specific language (DSL) is a specific programming language that has higher abstraction level and is specifically optimised for a specific field of problems [1]. Domain-specific languages support a narrow set of tasks in a chosen domain. In the last years, their significance in software engineering has grown, because of the evolvement of different kinds of programming languages. DSL brings such advantages to the economical field as well as daily work schedule such as higher productivity, assurance of quality, lower development cost. The user that apply in their tasks efficient domain-specific languages can easier express and solve their problems, take advantage of certain domain optimizations and observe enormous improvements in productivity.

The article aims to describe the development process of a domain specific language for document automation.

Document automation is about creating systems and workflows that help in the creation of electronic documents and it is a very popular topic these days [2]. It develops logic-based systems that use pre-existing segments of text and necessary data to construct a new version of the document. This technology is popularly applied in such industries to construct legal documents, letters and contracts. Document automation technology is used to allow automation of possible conditional text as well as variable text, and data commonly contained within a map of documents.

Automation systems that target documents domain enable different companies to substantially minimize data entry amount, reduce the proofreading time, and reduce the risks associated with human error [3]. Additional benefits proved to be financial savings due to less paper handling, document loading and paper waste [4]. Today, everyone fills out dozens, hundreds of forms. And often the same type of information is always entered. Whether it's at the doctor's office or at the police station, paperwork is always filled out.

The main purpose of a domain-specific language for document automation is to make the process of filling out documents easier for absolutely any domain. This DSL will ease the process of completing a document which already has a standard template. Thus the main purpose is to automate the process of completing documents – this being the reason why it is alternatively called document autocompletion. The user can create template functions which will represent the general draft of the document. Inside the template function, the user can set the parameters that must be submitted to the template. Furthermore, it is possible to have conditional text, error messages, and loops. Another feature which was considered important for this DSL, is the ability of the language to position text, and to adjust text design.

Input

This DSL has the following types of inputs: plain text files(.txt, .doc,.docx), PDF files and CSV files(or XLSX).

Plain text files(.txt, .doc,.docx, .pdf) - will be used for creating a template from already an existing document. This is the case when the internal template parameters should be marked with the '#' symbol in the readily made input text file. This is a fast method for creating templates and easier to understood by people who do not possess all the technical skills for understanding the DSL's syntax.

CSV files (or XLSX) - the spreadsheets can be used as input data for templates. For example a user has a spreadsheet full of data of ten people and he wants to create contracts for them. He can use this data as input to the template function. As a result ten contracts will be generated filled with the data from the spreadsheet.

Command line inputs - command line inputs can be used to manually fill in data of a template. It can be used for testing purposes, to check input/output connection, to avoid PDF export for wrong input. It offers more of a command line application for users to submit data to the template functions.

Output

The program can produce two types of outputs:

Command line output - the user can command line output to show error information and log the status of a running program in case it has errors.

PDF/Docs - the DSL's main type of output is a PDF file or a Docs file. The user can create user-made templates to later use that template to produce a PDF file that is filled with data and is ready for export.

Reference grammar

The DSL design includes several stages. First of all, definition of the programming language grammar L(G) = (S, P, VN, VT):

S - is the 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.

V_N = { <program>, <import_statement>,<template>,<actions>,<identifier> ,<char>, <nums_char>,<digit> , <starting_digit>, <template_body>, <params_declaration>, <main_template_body> , <variable_declaration>, <variable_type> , <num> , <text>,<count> ,<date> , <money>, <day> , <delimeter>, <month>, <delimeter>, <year>,<flow_control>, <text_action>, <until_statement>, <if_statement>,<expression>, <statements>, <method_fun>, <bin_op>, <method_op>, <arithm_op>, <relational_op>, <equal_op>, <conditional_op>, <text_design> , <color_name>, <method_call>, <method_name>, <font_type> , <actions>, <list_of_actions> , <file_name>
};

V_T = {use , 0,1….9, a | b | ... | z | A | B | ... | Z |, create template, [,], (,) , . , : , bool, words , chars ,10 ,11 , 12,“, ”, - , / , . ,end , template , until, if , else ,do, not , length , begin , end , error ,input , + , - , * , / |,% , < , > , <= , >=, == , != , and, or, pdf ,\u , \i , \center , \b, \color, \line, \space, \t red, blue, black, green, magenta, yellow, brown, gray , center, fontsize, font, right, left, merge, split, replace, Times New Roman, Arial, Georgia, actions, Open , createTemplate, make, createPack, sentences, global, contains, doc, docx, html }

Table 1

Meta notation

Notation (symbol)	**Meaning**
<word>	nonterminal
word	terminal
[x]	x is optional, zero or one occurrence of x, **‘[‘ ‘]’** are terminals
x$^+$	one or more occurrence of x
x*	zero or more occurrence of x
|	an alternative separation
‘{‘ ‘}’	used for grouping, { } are terminals

P = {<program> → <import_statement>$^+$ <template>$^+$ <actions>

<import_statement> → **use** <identifier>

<identifier>→ <char> <nums_char>*

<char>→ a | b | ... | z | A | B | ... | Z |

<nums_char>→ <char> | <digit> | - | _

<digit>→ **0** | <starting_digit>

<starting_digit> → 1 | 2 | ... | 9

<template> → **create template** <identifier> **:** <template_body>

<template_body> → [<params_declaration>] <main_template_body>

<params_declaration>→ **params ‘[‘** <variable_declaration>$^+$ **‘]‘**

<variable_declaration>→ <identifier> **:** <variable_type>
|**global** <identifier> **:** <variable_type> = <identifier>

<variable_type>→ <num> | <text>[**’[‘** <starting_digit> <digit>* <count> **‘]’]** | <date>
| <money> **| bool**

<count>→**words | chars | sentences**

<num>→ <starting_digit> <digit>* | **0.**<digit>$^+$

<text>→ **”** <char>* **”**

<date> → <day> <delimeter> <month><delimeter> <year>

<year> → <digit><digit><digit><digit>

<month> → **0**<digit> | **10** |**11** |**12**

<day> → <starting_digit><digit>

<delimeter> → **- | / | .**

<money>→ <num> $ | <num> <text>

<main_template_body>→<flow_control>* <text_action>$^+$ **end template**

<flow_control>→<until_statement>|<if_statement>

<until_statement>→
until <expression>**{**
<statements>
}
|
do {
<statements>
} until <expression>

<if_statement>→ **if** <expression> <statement>$^+$
| **if** <expression> <statement>$^+$ **else** <statement>$^+$

<expression> → <method_fun>
| <identifier>
| <expression><bin_op><expression>
| **not** <expression>
| (<expression>)
<method_fun>→ <identifier>**.**<method_op> [**(**<identifier>$^{+}$| <num> |<text>**)]**
<method_op>→**length |begin |end | contains |**
<bin_op> → <arithm_op> | <relational_op> | <equal_op> | <conditional_op>
<arithm_op> → **+ | - | * | / | %**
<relational_op>→ **< | > | <= | >=**
<equal_op> → **== | !=**
<conditional_op>→**and** | **or**
<statement> →<identifier> **.** <method_call>$^{+}$
| <identifier>=<expression>
|**error(**<text>**)**
|**input(**<identifier>**)**
<text_action> → <identifier> = { <text> }
|<identifier> = { ‘{‘<text_design> <text> \ ‘}’*
|‘{‘<text> # <identifier> ‘}’ * }
|**pdf(** <identifier>$^{+}$ **)**
|<identifier> = { ‘{‘<text> # <identifier> ‘}’ * }
|<identifier> **.** <method_call>$^{+}$
<text_design> → **\u | \i | \center | \b | \color=**<color_name> **| \line *** <num>
| \space * <num> **| \t *** <num>
<color_name>→**red | blue | black | green | magenta | yellow | brown | gray**
<method_call>→ <method_name> [**(**<identifier>$^{+}$| <num> |<text>**)]**
<method_name> → **center | fontsize | font(**<font_type>**) | right | left |merge | split**
| replace
<font_type> → **Times New Roman | Arial | Georgia**
<actions> → **actions :** <list_of_actions>$^{+}$
<list_of_actions> → <identifier> = **open(**<file_name>**)**
|<identifier> = **createTemplate(**<identifier> **)**
|**make** <identifier> **[doc, docx, html]**
|<identifier>**.(‘{‘** <text>**,** ’ }’***)**
|< identifier>**.(** <identifier>**)**
|< identifier>**.**< identifier> = <text>
|<identifier> = **createPack(**<identifier>$^{+}$ **)**
|<identifier> = <identifier>.<method_call>$^{+}$
<file_name> -><text>**.**{pdf | csv | docx}

Example of code and Derivation tree

In the following example of code a basic template named “test” is described. The template has two parameters , abc1 of type num, and abc2 of type money. Then the text is positioned using such keywords as center,and stylised using \b. Also the place of the variables in text is marked with #. Then in action part of the subprogram, the pdf is generated using make keyword. In this case the input will be taken from the console. Additionally, the source code is parsed by an abstract syntax tree. The result of parsing is presented in Figure 1.

```
create template test : params [
abc1 : num
abc2 : money ]
title = { \centre User Credentials \}
main_text = {
\b User\ identified by ID: # abc1
, has the following amount of money: # abc2
}
pdf(title,main_text)
end template
actions : make test
```

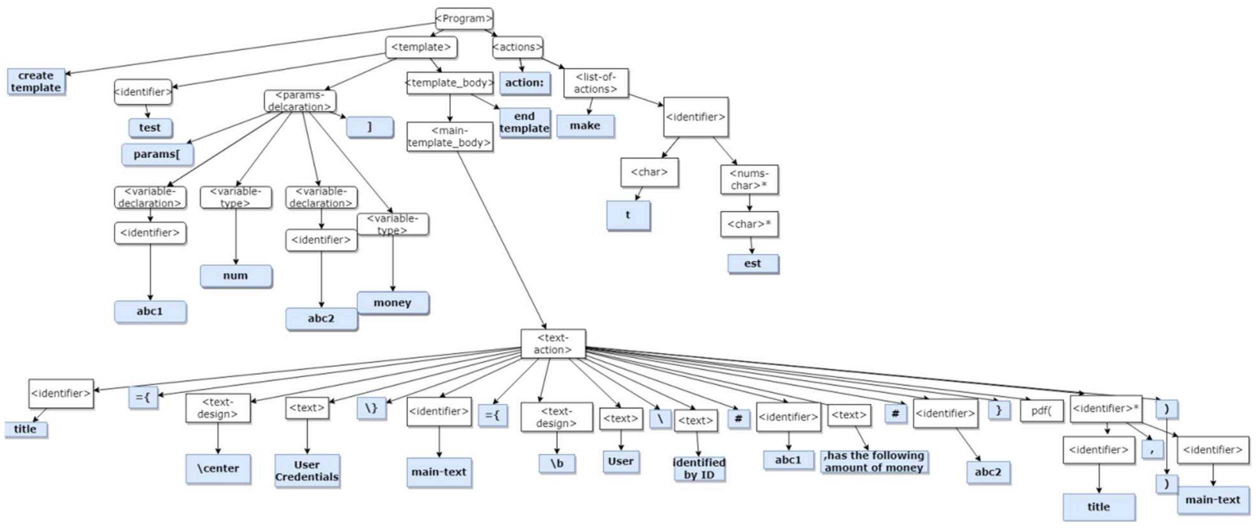

Figure 1. Parsing tree

Conclusion

The intention of this paper is to show that the concepts of Domain Specific Languages can improve and make faster the process of document completing. This DSL's easy syntax will allow people from the domain to easily grasp the language and use it in their daily tasks. This language will eliminate the need for any proofreading of the document as well as manual writing of the same document structure. The main users of our Document Automation DSL will primarily be specialists in the legal field, financial services, logistics, supply chain management.

To top it off, this DSL combines both flexibility of a general purpose language, as well as it is easily understood for non-technical people.

References

1. FOWLER, M. *Domain-Specific Language*s. Boston: Addison-Wesley Professional, 2010.
2. Document automation. [online]. [accesat 09.02.2022]. https://en.wikipedia.org/wiki/Document_automation
3. Legal Document Automation : Why are you waiting? By :"Affinity consulting group". [online]. [accesat 07.02.2022]. https://www.youtube.com/watch?v=eXi8m3iERUE
4. What is document automation? [online]. [accesat 07.02.2022]. https://www.bigtincan.com/what-is-document-automation/