MARKUP LANGUAGE FOR LABEL DESIGN

Călin RADU[1], Maxim COJOCARI-GONCEAR[1*], Sorin IAȚCO[1], Nichita CHIHAI[1], Andrei BARDIER[1]

[1]*Department of Software Engineering and Automatics, FAF-213, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chișinău, Republic of Moldova*

*Corresponding author: Maxim Cojocari-Goncear, maxim.cojocari-goncear@isa.utm.md

Scientific advisor/coordinator: Vasili BRAGA, university assistant, DISA, FCIM

Abstract. *This article is about Domain Specific Language (DSL) for label design, its architecture and conceptual implementation. Labels are crucial for branding and marketing, and a well-designed label can capture consumers' attention and convey important information about a product. Markup language, which refers to the visual and textual elements that comprise a label's design, plays a significant role in creating a label's overall aesthetic and communicating the desired message to consumers. This article discusses the various components of markup language, including color, typography, imagery, and layout, and how they can be used to create effective label designs.*

Keywords*: markup language, label design, DSL, grammar*

Introduction

A domain-specific language (DSL) is a language designed for usage within a given domain. A domain might be either a business or an application setting (e.g., an app extension, a plugin etc.). A DSL does not try to please everyone. Instead, it is designed for a certain set of issues and solutions, but it is powerful enough to express and handle those problems and answers. To exemplify, HTML is a DSL for web application development [1].

Generally speaking, a *label* is a little piece of cloth, paper, or plastic that is attached to an object. A label typically includes branding elements such as the company logo and colors, as well as product information such as the product name, description, ingredients, and nutritional information [2]. Labeling products not only provides crucial information and instructions to consumers but can also help your product stand out. In fact, 85% of shoppers say that their decision to buy a product is informed by reading a product's packaging while they're shopping [3].

Label design is the process of creating visual and textual elements that are printed or affixed to a product or package in order to identify and provide information about the product. The design of a label can vary depending on the product, brand, and target audience. Sometimes the label design process can be challenging, laborious and a long process. A DSL in this case, can become a handy solution to those who are looking for a specialized tool and want to save time and material resources in order to create an appropriate label. This tool can also help professionals work more efficiently, accurately, and flexibly, while also promoting standardization and collaboration within an organization or industry.

Language overview

A flexible, fully extendable, modular and user-friendly language, can be counted as the summary for DSL which we want to develop. This is a unique tool, in the field of languages, seen from the AI generated content perspective.

It is very easy to get to know the language, as it is similar to other markup languages. It has two main sections, in the *document* section: *meta* and *data*.

In the *meta section* the user defines all the auxiliary bits of information that will be bound to the output file, like the author of the file, the creation date or some short description. The document is created by the components shown in the *data section*. In each section we have a tag, with some

parameters, which dictates the component's behavior. The developed DSL is just a markup language for the label design, there are no data structures available, nor is there a way to manipulate, copy or produce data. The output will be a file, either in PDF format, or as a series of photos in .png format.

There will be no complex runtime error handling or checking, nor will there be a strict and well defined error checking system. This markup language has very easy rules, and we believe that the user gets a hand of it without much effort, and without the need of being guided by the interpreter.

Some components will have the required parameters for an AI generated image. The DSL will talk to an AI, and use the generated image as the value for the component. Extensibility and modularity will be provided to the user, so that any AI API, with some constraints could be used, if implemented.

Grammar definition

The grammar for a Domain Specific Language includes rules for defining the structure of statements, expressions, and other constructs that are specific to the domain. To enhance comprehension of the proposed DSL, its structure is presented below. Initially, the terms which the grammar utilizes need to be defined. The grammar of the proposed language is described as $G = (V_N, V_T, S, P)$ where V_N is the finite set of non-terminal symbols, V_T is the finite set of terminal symbols, S is the start symbol, and P is the finite set of production rules.

Table 1

Meta notations

Notation	Meaning
<text>	a nonterminal parameter is written between <>
text	a terminal parameter is written in bold
text*	the symbol appears zero or more times
text$^+$	the symbol appears one or more times
|	an alternative follows

V_N = {<root_component>, <parent_component>, <terminal_symbol>, <component>, <left_square_bracket, <right_square_bracket>, <equal_sign>, <digits>, <component_name>, <double_quotes>, <component_param>, <text_char>, <root_component_name>, <parent_component_name>, <component_param_name>, <component_param_value>};

V_T = {**document**, **meta**, **data**, **author**, **name**, **format**, **path**, **size**, **title**, **text**, **image**, **barcode**, **batch_number**, **date**, **[**, **]**, /, **=**, **"**, -, **[a-z]**, **[A-Z]**, **[0-9]**, **value**, **type**, **width**, **height**, **font_size**, **font_weight**, **position_x**, **position_y**, **src**, **prompt**};

S = {<root_component>};

P = {

<root_component> → <left_square_bracket><root_component_name> <right_square_bracket><parent_component>*<left_square_bracket><terminal_symbol> <root_component_name><right_square_bracket>

<parent_component> → <left_square_bracket><parent_component_name> <right_square_bracket><component>*<left_square_bracket><terminal_symbol> <parent_component_name><right_square_bracket>

<left_square_bracket> → **[**

<right_square_bracket>→ **]**

<terminal_symbol> → /

<equal_sign> → **=**

<double_quotes> → **"**

<text_char> → **a** | **b** | ... | **z** | **A** | **B** | ... | **Z** | **ε** | **-**

<digits> → **0** | **1** | ... | **9**

<component_name> → **author** | **title** | **text** | **image** | **barcode** | **batch_number** | **date** | **format** | **name** | **path** | **size**

<parent_component_name> → **meta** | **data**

<root_component_name0> → **document**

<component_param_name> → **value** | **type** | **width** | **height** | **font_size** | **font_weight** | **position_x** | **position_y** | **src** | **prompt**

<component_param_value> → <text_char> | <digits> | <digits><text_char>*

<component_param> → <component_param_name><equal_sign><double_quotes><component_param_value><double_quotes>

<component> → <left_square_bracket><component_name><component_param>*<right_square_bracket>

};

Syntax example

For a better understanding of how the DSL will look like, an example of syntax is presented in the code snippet below.

```
[document]
    [meta]
        [name value="pinot_grigio_purcari.pdf"]
        [format value="pdf"]
        [path value="C:\Users\andre\OneDrive\Desktop"]
        [size value="320KB"]
        [date value="08-03-2023"]
        [author value="Bardier Andrei"]
    [/meta]
    [data]
        [title value="Pinot Grigio de Purcari" font_size="14"
         font_weight="2"]
        [image prompt="A label logo for a bottle of Moldavian wine Pinot Grigio
de Purcari. The logo has a white background with black text and a gold border.
The text says 'Purcari' in large cursive letters at the top and 'Pinot Grigio'
in smaller serif letters below it. There is also a small emblem of a grapevine
with green leaves and purple grapes on the left side of the logo. The image is
realistic, high-resolution (768x768), and sharp with no blurriness or noise."
src="C:\Users\andre\OneDrive\Desktop\logo.png"]
        [text type="origin" value="Vin cu indicatie geografica Purcari"]
        [text type="wineType" value="Alb sec"]
        [text type="bottled" value="2013"]
        [text type="alc&vol" value="Alc. 14,0% vol. 0,75l."]
        [text type="manufacturer" value="IM 'Vinaria Purcari'"]
        [barcode]
        [batch_number]
        [date]
    [/data]
[/document]
```

Parsing tree

A parse tree is a hierarchical representation of the syntactic structure of a string of symbols or code, typically in the form of a tree or a directed acyclic graph. It is created by a parser during the process of parsing, which is the process of analyzing the syntax of a program and determining its grammatical structure.

The parsing tree for a sequence of code, according to the grammar defined in the previous section, is shown in Fig. 1.

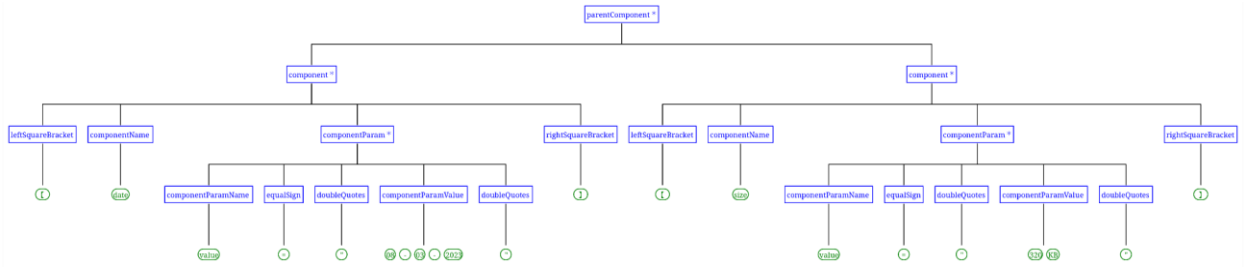

Figure 1. Parsing tree example

Conclusions

To summarize, by using a markup language described above, label designers can easily create and manage label designs with a high degree of precision and consistency. This is particularly important for businesses that need to produce large volumes of labels, as markup language makes it easy to generate labels in bulk and maintain consistency across different label designs. By using markup language, designers can also integrate dynamic data sources to populate label fields with up-to-date information. Furthermore, this DSL may be an essential tool for label design, offering numerous benefits in terms of efficiency, accuracy, and consistency. As more businesses recognize the advantages of using markup language for their label design needs, we can expect to see even greater adoption of this technology in the future.

References

1. MANAGOLI, G., *What developers need to know about domain-specific languages* [online]. 2020. [accessed 26.02.2023]. Available: https://opensource.com/article/20/2/domain-specific-languages
2. What is a label? Definition and examples. [online]. [accessed 26.02.2023]. Available: https://marketbusinessnews.com/financial-glossary/label/
3. Why Is Product Labeling So Important? [online]. [accessed 26.02.2023]. Available: https://www.luminer.com/articles/why-is-product-labeling-important/