DOMAIN-SPECIFIC LANGUAGE FOR ANALYZING MEDICAL RESULTS

Maia ZAICA[1*], Felicia LUPAȘCU[1], Cristian BRÂNZA[1], Nichifor POPESCU[1]

[1]*Department of Software Engineering and Automatics, Group FAF-212, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chișinău, Republic of Moldova*

*Corresponding author: Maia Zaica, maia.zaica@isa.utm.md

Abstract. *The following article aims to describe the way how the Medical Domain could benefit from a Domain Specific Language. The article explores the advantages of using Domain Specific Language in medical result analysis, such as improved accuracy, faster processing times, and reduced error rates. It also could benefit in collecting, updating, and keeping data. Finally, the article discusses potential future applications of Domain Specific Language in the medical field and how it can continue to contribute to the way medical results are analyzed and interpreted.*

Keywords: *analysis, data, domain-specific language, grammar, medical result, specific problem.*

Introduction

The analysis of medical results is an essential task in the healthcare industry, providing insights that aid in the diagnosis, treatment, and prevention of diseases. However, analyzing large volumes of medical data from different sources can be a challenging task, especially for healthcare professionals who may not have the necessary technical skills.

Domain-Specific Languages (DSLs) provide a promising solution to this problem by offering a specialized programming language that is tailored to the needs of the medical industry. In this article, a DSL designed specifically for the analysis of medical results is presented. The authors begin with an overview of domain analysis, discussing the key challenges associated with medical data analysis. An overview of the DSL is provided, highlighting its key features, including its ability to simplify complex data analysis tasks and improve the accuracy of medical diagnoses.

The grammar of the DSL is designed to be user-friendly and easy to understand. The potential impact of the DSL on the medical industry, including improved patient outcomes and reduced costs, is also discussed. Overall, the article provides a comprehensive overview of a domain-specific language for medical data analysis, offering insights into its design, implementation, and potential benefits.

Domain Analysis

General-purpose languages (GPLs) are robust and versatile, but they have weaknesses like everything. Comparing GPLs with general doctors:

"They are good at solving the first layer of diseases (problems), but they can't go deep, such as having a full diagnosis of heart disease (domain-related problems)" [1].

Thus, DSLs were created to provide a more effective way to address specific problems in a particular domain or industry. Starting with the definition of domain-specific language. "*Domain-specific languages are languages tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application"* [2]. Rather than relying on general-purpose programming languages, which may not be optimized for a specific task or domain, DSLs allow developers to create languages that are tailored to the specific needs of a particular domain. This can result in increased productivity, better code readability, and more efficient code execution [3].

Developing a Domain-Specific Language is a challenging task that requires a combination of expertise in both language development and the specific domain the language is intended for.

A healthy population is essential for sustainable development and a strong macroeconomy. In turn, a strong economy is necessary to generate sufficient resources for health systems. Yet, despite this mutually beneficial relationship, it is often overlooked and health systems expenditure isn't taken into consideration most of the time. They may support the goal of universal health coverage, but have concerns that spending on health may be a drain on the economy [4].

The Republic of Moldova has a universal health care system. The financing of healthcare services in the country is based on the principles of fairness and unity, ensuring that everyone has equal access to healthcare services regardless of their financial status. However, over 10% of the population lacks health insurance coverage [5]. The level of transparency of the domestic health system is 36%, which ranks us among the countries with an unsatisfactory level of transparency. It is relatively simple to increase transparency by publishing data that institutions have already created and utilized internally [6].

To identify potential issues within the healthcare system that could be addressed or improved, secondary research was conducted. Secondary research involves analyzing and utilizing pre-existing data sources rather than collecting new data through primary research methods. This approach allows for the examination of a large amount of information and can provide insights into current trends, patterns, and potential areas for improvement. Healthcare organizations should look for advanced solutions that support: Easy Information Gathering, Real-Time Data Updates, Clear Data Visualization, and Simple Reporting.

Domain-specific language can be an effective solution to address the lack of system interoperability in healthcare. Healthcare systems often use different data formats and structures, which can make it difficult to share data between systems. By creating a DSL specifically designed for healthcare data, healthcare professionals can standardize the way that data is stored, analyzed, and shared across different systems. By using a common language, DSL can help to reduce errors and inconsistencies that can arise when data is transferred between different systems, ultimately improving patient care and outcomes.

A markup language annotates text with tags or codes to provide meaning and context to content. Tags indicate how text should be displayed or structured, and are embedded in the text itself. Markup languages make it easy to display documents on various devices and process them with software applications.

Grammar

A grammar G is an ordered quadruple $G = (V_N, V_T, P, S)$, where:

$-V_N$ is a finite set of non-terminal symbols

$-V_T$ is a finite set of terminal symbols

$$V_N \cap V_T = \emptyset$$

$-P$ is a finite set of production rules

$$P \subseteq (V_N \cup V_T)^* V_N (V_N \cup V_T)^* \times (V_N \cup V_T)^*$$

$-S$ is the start symbol

In a markup language, the terminal symbols represent the actual content of the document or data being marked up. The non-terminal symbols represent the structure or syntax of the markup language itself.

The production rules describe how non-terminal symbols can be replaced or expanded into other non-terminal or terminal symbols. These rules define the syntax and structure of the markup language.

To specify the grammar representation for the markup language we need to use meta notations (table 1):

Table 1

Meta notations

Symbol	Definition
< >	Non-terminal symbol
*	Zero or more occurrences
+	One or more occurrences
|	Separates the alternative symbols
->	Derivation
~	Except: any character except
\\	Comment

V_N = {medical_results, test_result, imaging_result, lab_result, query, query_type, result_type, patient_data, patient_id, birthday, date, test_name, imaging_type, lab_name, result, unit, reference_range, normal_range, high_range, low_range, image_location, location}

V_T = {"test", "imaging", "lab", "show", "for", "all", "average", "tests", "imaging", "labs", "patient", "age", "-", ">", "<", "units", "normal range", "high range", "low range"}

P = {<medical_results> -> <test_result> | <imaging_result> | <lab_result> | <query>
<test_result> -> "test" <test_name> ":" <result> ["units" <unit>] [<reference_range>]
<imaging_result> -> "imaging" <imaging_type> ":" <result> ["units" <unit>] [<image_location>]
<lab_result> -> "lab" <lab_name> ":" <result> ["units" <unit>] [<reference_range>]
<query> -> "show" <query_type> "for" <patient_data>
<query_type> -> "all" <result_type> | "average" <result_type> "for" <time_frame>
<result_type> -> "tests" | "imaging" | "labs"
<patient_data> -> "patient" <patient_id> | "age" <birthday>
<patient_id> -> <word>+
<birthday> -> <date>
<date> -> <year> "-" <month> "-" <day>
<year> -> <number>
<month> -> <number>
<day> -> <number>
<test_name> -> <word>+
<imaging_type> -> <word>+
<lab_name> -> <word>+
<result> -> <number>
<unit> -> <word>+
<reference_range> -> "normal range" <normal_range> | "high range" <high_range> | "low range" <low_range>
<normal_range> -> <number> "-" <number> <unit>
<high_range> -> ">" <number> <unit>
<low_range> -> "<" <number> <unit>
<image_location> -> "location" <location>
<location> -> <word>}

An example of code using this grammar representation is:

```
Description {
	test glucose= 120
	units= mg/dL
	stage= normal
	range= 70-100
}
```

```
Setting {
        patient= John Smith,
        age= 1978-01-01
        gender= male,
        weight= 175 lbs,
        height= 5'11"
}
Response {
        show all tests for patient age 1978-01-01
         // Output: glucose: 120 mg/dL (normal)
}
Response {
        show average labs for last 6 months for patient John Smith
         // Output: No lab results found for patient John Smith in the last 6 months.
}
Response {
        show all imaging for patient John Smith
         // Output: No imaging results found for patient John Smith.
}
Response {
        length 300
        prompt "The patient's glucose test results were higher than normal and
        indicate the presence of diabetes. The patient is advised to follow up with
        a healthcare provider for further evaluation and treatment."
}
```

Parsing

Parsing algorithm:

A common approach for parsing context-free grammars like this one is to use a technique called recursive descent parsing. Here is how it could be done for this grammar:

1. Start with the <medical_results> rule.
2. Check if the next token matches <test_result>, <imaging_result>, <lab_result>, or <query>.
3. If it matches one of those rules, call the corresponding parsing function.
4. If it doesn't match any of those rules, return an error.
5. The parsing function for <test_result> would look for the "test" keyword, then call the <test_name>, <result>, and <unit> parsing functions in order. It would also check if there is a <reference_range> and parse it if it's there.
6. The parsing function for <imaging_result> would look for the "imaging" keyword, then call the <imaging_type>, <result>, and <unit> parsing functions in order. It would also check if there is an <image_location> and parse it if it's there.
7. The parsing function for <lab_result> would look for the "lab" keyword, then call the <lab_name>, <result>, and <unit> parsing functions in order. It would also check if there is a <reference_range> and parse it if it's there.
8. The parsing function for <query> would look for the "show" keyword, then call the <query_type> and <patient_data> parsing functions in order.
9. The parsing function for <query_type> would look for either "all tests", "imaging", "labs", "average tests", or "average imaging", and parse the associated data if necessary.
10. The parsing function for <patient_data> would look for either "patient" and a patient ID, or "age" and a date of birth.
11. The parsing function for <patient_id> would parse any string of alphanumeric characters.
12. The parsing function for <birthday> would look for a date in the format "YYYY-MM-DD".
13. The parsing function for <test_name>, <imaging_type>, <lab_name>, <number>, <unit>, <normal_range>, <high_range>, and <low_range> would all parse their respective data according to the grammar rules.

Recursive descent parsing works by recursively calling parsing functions for each rule in the grammar until the entire input is parsed. If the input is valid according to the grammar, a parse tree is constructed that represents the structure of the input. If the input is not valid, an error is returned.

Conclusions

In conclusion, the development of a Domain-Specific Language for analyzing medical results offers a promising solution to the challenges associated with medical data analysis. The DSL is designed to simplify the process of analyzing large volumes of medical data from different sources, improving the accuracy of medical diagnoses and reducing costs in the healthcare industry. Its user-friendly and intuitive nature, combined with a simple user interface, makes it accessible to healthcare professionals who may not have programming skills.

Overall, the DSL has the potential to transform the way medical data is analyzed, leading to improved patient outcomes and increased efficiency in the healthcare industry. As such, the development and adoption of this DSL should be a priority for healthcare professionals and organizations looking to improve their medical data analysis capabilities.

Acknowledgments

This article and the research behind it would not have been possible without the direction of our coordinator, Irina Cojuhari.

We'd like to express our gratitude and appreciation for the invaluable contribution to the completion of our article. The guidance and expertise provided were of utmost importance, and we appreciated the approachability and willingness to assist in every way possible, including providing mentorship and offering constructive feedback.

References

1. ALVES, I.R. Making your life easier with domain-specific languages (DSLs). In: *WAES*, January 2023. [online]. [accessed 06.03.2023]. Available: https://medium.com/wearewaes/making-your-life-easier-with-domain-specific-languages-dsl-1838d351d35
2. MERNIK, M., HEERING, J., SLOANE, A.M. When and how to develop domain-specific languages. In: ACM Computing Surveys, December 2005, p. 1 https://dl.acm.org/doi/10.1145/1118890.1118892
3. JetBrains. *Domain-Specific Languages*. [online]. [accessed 06.03.2023]. Available: https://www.jetbrains.com/mps/concepts/domain-specific-languages/
4. Euro Health Observatory (WHO). *Health and economy.* [online] [accessed 12.02.2023]. Available: https://eurohealthobservatory.who.int/themes/observatory-programmes/health-and-economy
5. Euro Health Observatory (WHO). *Country overview*. [online]. [accessed 12.02.2023]. Available: https://eurohealthobservatory.who.int/countries/republic-of-moldova
6. CARP, M.,GOMA, L. Efficiency of medical services in the Republic of Moldova. In: *Cercetarea în biomedicină și sănătate: calitate, excelență și performanță.* (R), october 20-22, 2021, Chișinău. Chișinău, Republica Moldova: 2021, p. 64. https://ibn.idsi.md/vizualizare_articol/144098