DOMAIN SPECIFIC LANGUAGE FOR CREATING API DOCUMENTATION

Artiom MARTÎNIUC, Vladislava MUSIN, Eugen OSTAFI,
Marius POPA*, Nichista POPOV

Department of Software Engineering, group FAF-222, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chișinău, Republic of Moldova

*Corresponding author: Marius Popa, marius.popa@isa.utm.md

Tutor/coordinator: **Cristofor FIȘTIC**, asistent universtar, UTM

Abstract. *This article introduces a Domain-Specific Language (DSL) designed for the management and display of API documentation, aiming to ease the documentation process and enhance the quality of API guides. The common approach to documenting APIs often involves manual processes that are time-consuming and vastly different across different sources, because of the variety in either company practice or industry standards. The proposed DSL addresses these challenges by offering a structured, simple format to understand the complexities involved in describing API endpoints, parameters, request and response models, and perhaps even examples of such use. The main goal of this work lies in understanding and getting rid of the struggles developers face when having to deal with new API documentation. The specific aim of the DSL is to allow for quick adaptation by developers and technical writers without requiring extensive, dreadful research and understanding of the different interfaces and classifications of various sources. By combining theoretical design principles and practical implementation strategies, this work aims to demonstrate the effectiveness of using a DSL for API documentation. It will provide a comprehensive solution that will reduce the effort needed for using and understanding API documentation, by making it more consistent and accurate, therefore enhancing developer experience when having to use APIs.*

Keywords: *complexity reduction, consistency, documentation quality, domain-specific language.*

Introduction

A Domain-Specific Language is a programming or specification language dedicated to a particular problem domain, designed to simplify tasks within that domain. DSLs are often created to enable more intuitive and efficient solutions compared to general-purpose languages when dealing with specific types of problems or processes [1].

An API, or Application Programming Interface, consists of a set of communication protocols and subroutines that allow different software programs to interact with each other. APIs can be developed for various systems such as operating systems, database systems, hardware, JavaScript files, or other object-oriented files to facilitate this interaction. API documentation refers to a set of technical instructions on how to use and integrate it properly [2].

In our article on creating a Domain-Specific Language for API documentation, we have divided the content into three primary sections: domain analysis, grammar definition, and a sample program.

Domain Analysis

The evolution of API documentation experienced many changes, similar to the shifts in software development practices, going from boring manuals to interactive online resources. With the introduction of web APIs, it led to more standardized documentation, made possible by tools like Swagger (now OpenAPI) and RAML. These tools improved the accessibility and usability of

API documentation, making it more understandable for humans and processable by machines, helping to create a more integrated digital ecosystem [3].

Nonetheless, API documentation faces several difficulties, including maintaining consistency and clarity across different APIs, ensuring accessibility and usability for both novice and expert developers, and keeping the documentation up to date. To address these problems, the concept of a Domain-Specific Language for API documentation has been proposed. A DSL could streamline and standardize documentation practices, ensure updates and even include accurate API guides, thus improving efficiency and reducing the work developers must put in.

Potential users of this DSL include API developers, technical writers, software development teams, and quality assurance engineers, who would all benefit from this product in their respective fields. Furthermore, people outside of this specialized domain could make use of it, like project managers and product owners that could use the DSL to improve project scoping and team performance. Educators and independent students could also find value in a DSL, using it as a learning platform and promoting collaboration between students.

The introduction of a DSL for API documentation would be a great improvement in creating and maintaining API guides, aiming to enhance the developer experience and support more efficient API integration within software ecosystems. This would create a more intuitive, accessible, and effective documentation ecosystem for everyone involved.

Language Overview

The Domain-Specific Language (DSL) for API documentation is made to simplify the creation and maintenance of API docs through a structured computational approach. The idea starts with parsing, where user-written DSL commands are converted into a more manageable internal format, known as an Abstract Syntax Tree (AST). This structure organizes the commands related to API elements and makes it easier to handle everything.

After the AST is fully fleshed out, the DSL taps into this structured data to craft API documentation in various formats, according to user preferences. The parsing process is supported by ANTLR, a powerful tool that helps in creating grammars and generating parsers in programming languages, in this case Python. By employing patterns like listeners or visitors, it becomes easy to traverse the AST and extract the necessary details needed to generate precise and comprehensive API documentation [4].

Input-wise, the DSL is quite accommodating, accepting commands directly in its own syntax or via structured data files, adding to its user-friendly nature. The language itself is declarative, emphasizing what the documentation should cover rather than how to assemble it. This approach makes the documentation process easier and minimizes errors by making sure data inputs stick to the expected formats and by conducting syntax checks right at the parsing stage.

In essence, the DSL makes it possible to generate efficient, structured, and error-free API documentation. This process not only ensures accuracy but also keeps the documentation aligned with the latest API specifications, greatly improving clarity and utility for developers.

Semantic Rules

For the user to be able to efficiently use the language without getting any errors, there is a set of rules put in place that he needs to follow:

- Keywords and identifiers are case-sensitive, distinguishing between similar terms.
- Supports only single-line comments, marked with a hashtag #, to simplify annotations.
- Spaces, tabs, and newline characters are used to separate tokens, making sure formatting errors don't affect interpretation.
- Enclosed in double quotes and can include escaped characters, used for specifying paths, descriptions, and text content.
- Must start with a letter or an underscore, can include letters, digits, and underscores, but cannot start with a digit.

- Keywords are reserved for defining and structuring documentation and cannot be used as identifiers.
- Uses symbols for specific syntactic roles such as defining routes, specifying methods, or delineating code blocks.
- Recognizes numeric literals for use in contexts like specifying versions or limits.
- Uses semicolons to end declarations, to make documentation clearer and separate statements.

Grammar

The grammar of a programming language is an important part that dictates how programs are written and understood by a compiler or an interpreter. It is made up of a set of production rules, which are guidelines for generating valid sequences of symbols and constructing a good program structure. These production rules are expressed using a combination of terminal and non-terminal symbols, special characters, and notations that determine how elements in the language are combined and interpreted.

Table 1

Grammar description

Symbol	Meaning
<notation>	Non-terminal
notation	Terminal
x*	x occurs 0 or multiple times
|	Separate alternatives
→	Derives
#	Comment
<notation>+	Must be one or more notations
<notation>?	Notation is optional

Sample Program

Here we can see, through the representation of a parse tree, a possible case generated with the help of the grammar rules stated previously:

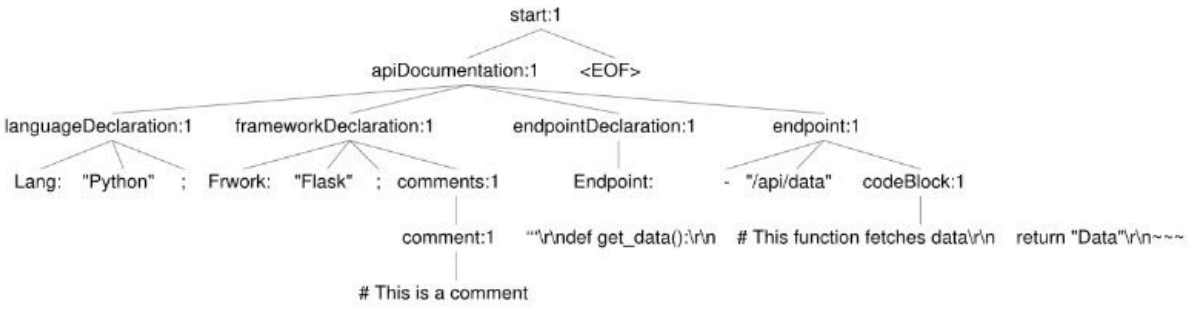

Figure 1. Parse tree

Conclusion

To conclude, creating a Domain-Specific Language (DSL) for API documentation marks an important step toward how we handle API interfaces. By implementing this specialized DSL, both developers and technical writers can overcome the usual problems related to traditional documentation methods. This approach brings a much-needed clarity, ensures a consistent style across various APIs, and cuts down on the time and effort needed to keep documentation fresh and accurate. With its well-defined grammar, semantic rules, and versatile output options—ranging from HTML and Markdown to PDF—the DSL allows for systematic documentation production that adapts easily to different needs. The introduction of this DSL could change the way developers

and companies approach API documentation, making it a more intuitive and productive part of the software development lifecycle, and allowing their resources to be diverged towards other problems. This change does not only benefit technical users but also educators, project managers, and others involved in the broader scope of the field of documentation and API use.

References

[1] "Domain-Specific Languages [Online]. Available: https://www.jetbrains.com/mps/concepts/domain-specific-languages/

[2] "What is an API?" [Online]. Available: https://www.geeksforgeeks.org/what-is-an-api/

[3] "6 Best API Documentation Tools" [Online]. Available: https://blog.dreamfactory.com/5-best-api-documentation-tools/

[4] "What is ANTLR?" [Online]. Available: https://www.antlr.org/