# DEVELOPMENT OF DOMAIN SPECIFIC LANGUAGE: WEB API QUERY LANGUAGE

**Artiom BALAN[1], Dan-Octavian CARP[1], Andreia-Cristina SIREȚEANU[1], Iuliana STEȚENCO[1]**

*[1]Department of Software Engineering and Automation, group FAF-211, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chișinău, Republic of Moldova*

*Corresponding Author: Artiom Balan, artiom.balan@isa.utm.md

**Coordinator: Vasile DRUMEA**, univ. assist., Departament of Foreign Languages, TUM

***Abstract.*** *This paper was created in the context of a Problem Based Learning (PBL) project, whose main objectives were the analysis of compilers, interpreters, and programming languages and the further development of a Domain Specific Language called RCCN: REST Complex Consolidation Notation, which aims to simplify the use of Web APIs. This article examines and explains the Domain Specific Language's implementation, including its advantages, grammar, and parse tree.*

***Keywords****: API, grammar, domain specific language, parse tree*

## Introduction

A domain-specific language (DSL) is an executable specification language that provides expressive capacity targeted at and typically limited to a single issue domain through proper notations and abstractions [1]. The targeted expressive capability of DSLs is its defining feature and that's also the reason why they are often compact and simpler, providing only a few number of notations and abstractions.

The term "API," or its expanded form "Application Programming Interface," is usually always used in the context of the modern methodology, in which HTTP is used to provide access to machine-readable data in a JSON or XML format, sometimes referred to as "web APIs." Although APIs have existed for almost as long as computing, contemporary web APIs only started to emerge in the early 2000s [2].

RCCN (REST Complex Consolidation Notation) would represent a query language, designed to simplify and offer an easier way of working with APIs by structuring the needed information in a JSON format for further usage.

## Benefits

The existence of RCCN has multiple benefits such as:

- Simplifies the work with APIs;
- The programs in this DSL are way more concise;
- Enhance productivity for developers;
- Embody domain knowledge.

Of course, there are some pitfalls as well like: the costs of designing and implementing such a DSL and the need for users (e.g. programmers) to learn how to use it. In order to diminish the impact of these disadvantages, the language will be as simple as possible for the developers to understand, learn and use.

## Grammar

A formal grammar is defined as G = ($V_N$ ,$V_T$ , $P$, $S$), where:

- $V_N$ is a finite set of nonterminal symbols,
- $V_T$ is a finite set of terminal symbols,
- $P$ is a finite set of production rules,
- $S$ is the start symbol.

Following is the grammar definition of the DSL, written in a variant of EBNF notation, which is explained briefly in the Table 1.

*Table 1*

**Metanotations**

| Notation | Description |
|---|---|
| foo | Nonterminal |
| ‘…’ | Terminal string |
| foo+ | 1 ≥ occurrences of |
| foo* | 0 ≥ occurrences of |
| \| | Alternative elements |
| (…) | Grouping elements |
| […] | Optional element |

$S$ = { document }

$V_T$ = { ’type’, ’{’, ’}’, ’:’, ’(’, ’)’, NL, INT, WS, Name}

$V_N$ = {document, definition, typeDefinition, objectTypeDefinition,
fieldsDefinition, fieldDefinition, params, param,
queryDefinition, selectionSet, fields, field}

$P$ = {
document ::= definition* EOF

definition
::= ( typeDefinition | queryDefinition ) NL*

typeDefinition
::= objectTypeDefinition

objectTypeDefinition
::= ’type’ Name ’:’ NL* ’{’ fieldsDefinition ’}’

fieldsDefinition
::= fieldDefinition*

fieldDefinition
::= Name params? ’:’ Name NL*

params ::= ’(’ NL* param ( ’,’ NL* param )* NL* ’)’

param ::= Name ’:’ Name

queryDefinition
::= ’query’ Name ’:’ NL* selectionSet

selectionSet
::= ’{’ fields NL* ’}’

fields ::= field*

field ::= Name ( NL* | ’:’ NL* selectionSet )

INT ::= [0-9]+

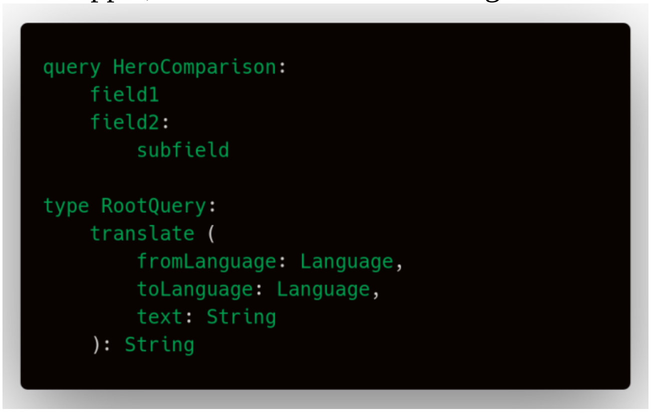
WS ::= [ \t\r\f]+

NL ::= ’\r’? ’\n’

Name ::= [a-zA-Z_] [a-zA-Z_0-9]*
}

**Parse Tree**

A representation of code snippet, that was constructed using RCCN is shown in Figure 1:



```
query HeroComparison:
    field1
    field2:
        subfield

type RootQuery:
    translate (
        fromLanguage: Language,
        toLanguage: Language,
        text: String
    ): String
```

**Figure 1. RCCN code example**

A derivation tree or parse tree is a graphical illustration that shows how strings in a language are generated while taking the grammar rules into account [3]. Given the grammar provided earlier in this paper and the code snippet from Figure 1, the Parse Tree for the domain specific language is shown in Figure 2.

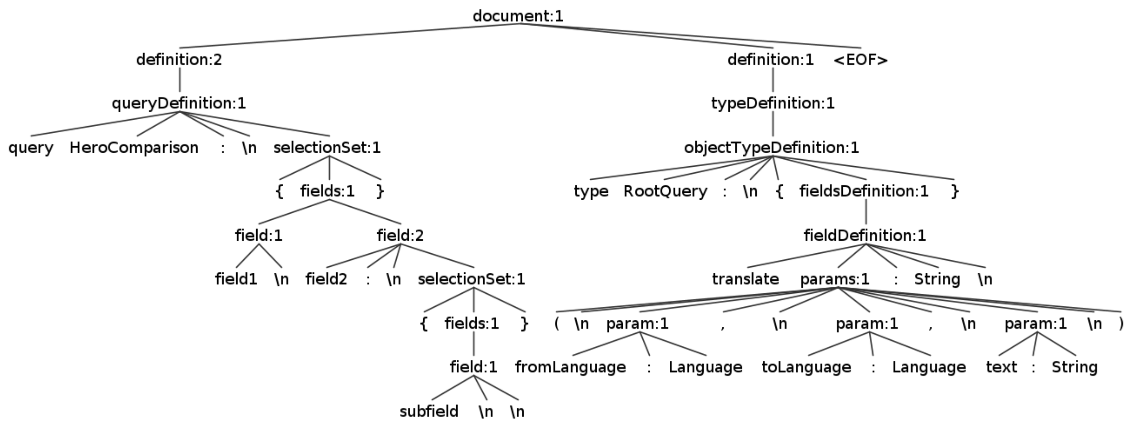


**Figure 2. RCCN Parse Tree**

**Conclusions**

This study introduced the Domain Specific Language for Web APIs, which is designed to address the issue of difficult API work and provide a solution by making the language more approachable and attractive. Developers can benefit from improved workflow, faster processes, and time savings by implementing the RCCN language. The grammar of the language is recognizable by other high-level programming languages, making it a simple and approachable tool for newcomers to the industry. Because of RCCN's simple design, skilled developers may quickly integrate it into their current processes without making major adjustments or undergoing additional training. Overall, RCCN will be a development, giving programmers a strong, effective tool that can help them easily and swiftly accomplish their objectives.

**References:**

1. DEURSEN van A., KLINT P., VISSER, J. Domain-Specific Languages: An Annotated Bibliography [online]. [Accessed: 20.02.2023]. Available: https://web.archive.org/web/20160316125655/http://www.st.ewi.tudelft.nl/~arie/papers/dslbib.pdf
2. LANE, K. Intro to APIs: History of APIs [online]. [Accessed: 21.02.2023]. Available: https://blog.postman.com/intro-to-apis-history-of-apis/
3. SCIENCEDIRECT, Derivation Tree [online] [accessed 25.02.2022] Available: https://www.sciencedirect.com/topics/computer-science/derivation-tree