A DSL SOLUTION FOR MOLDOVA'S TAX SYSTEM

Vlad UNGUREANU*, Ludmila FRIPTU,
Ecaterina MUNTEANU, Dmitrii CRAVCENCO, Vasile CEBAN

Department of Software Engineering and Automation, gr. FAF-223, Faculty of Computers, Informatics, and Microelectronics, Technical University of Moldova, Chisinau, Republic of Moldova.

*Corresponding author: Vlad Ungureanu, vlad.ungureanu@isa.utm.md

Tutor/coordinator: **Gabriel ZAHARIA**, universitary assistant

Abstract. *Moldova's tax system is incredibly complex, with its many rules, changing rates, and special deductions. This complexity can lead to costly mistakes for individuals and businesses when calculating taxes. To address this challenge, a Domain-Specific Language specifically designed for Moldovan taxes is being developed. This DSL aims to use the same language and structure as the tax laws, making it easier to understand and reducing the likelihood of errors during tax calculations. Imagine a system where individuals, small businesses, and even large companies can save time and feel more confident about fulfilling their tax obligations. The DSL has the potential to make this a reality by simplifying calculations and minimizing the risk of unexpected penalties. This, in turn, could lead to better financial planning and reduced stress for business owners.*

Keywords: *Moldova, computation, DSL, simplification, tax system, automation*

Introduction

For individuals and businesses in Moldova, the annual task of calculating taxes often means grappling with complex regulations, a multitude of variables, and the ever-present risk of costly errors or missed opportunities for deductions. This paper proposes a practical solution in the form of a DSL created expressly for Moldovan tax computations. This DSL has the potential to improve accuracy, save time, and reduce the stress associated with tax obligations by offering a streamlined and intuitive approach.

Designed to reflect the terminology and structure of Moldovan tax laws, the DSL looks to eliminate the need to translate complex regulations into generic spreadsheet formulas. This tailored approach aims to minimize the potential for miscalculations and misunderstandings. Moreover, the DSL could be designed to incorporate automated updates, ensuring it remains synchronized with the latest tax code changes. This would alleviate the burden of manually tracking amendments, further simplifying the process for taxpayers. By handling calculations with greater precision and efficiency, the DSL could ultimately help individuals and businesses feel more confident about their tax compliance, providing peace of mind within a typically complicated process [1].

Understanding Moldovan Taxes: A Clearer Path with a DSL

Understanding taxes in Moldova can be daunting due to the complex regulations and jargon like "bracket" and "deduction" [2]. To address this, a DSL is being developed to simplify the process of calculating Moldovan taxes. This DSL will employ familiar terms from the tax code, such as "income" and "dependents," making it easier to use. This approach not only reduces stress but also enhances understanding, enabling users to see how different tax law components affect their dues. The DSL is designed to be intuitive, replacing complex formulas with straightforward language and step-by-step calculations. This transparency helps users stay informed about the latest tax changes, ensuring they use current rates and avoid mistakes. It's more than just a tool for calculation, it's a learning aid that empowers users to manage their taxes more effectively,

potentially assisting others too [3]. Ultimately, the DSL aims to provide peace of mind by offering a clear and reliable method for tax calculations, with the prospect of future integration with official government tax platforms for even greater ease and accuracy.

Grammar

Table 1

Representation of the Meta notation

< >	These are metacharacters used to denote optional elements within a rule. Anything enclosed within angle brackets might appear zero or one time in the corresponding construct.
[x]	Means zero or one occurrence of x, i.e., x is optional; note that brackets in quotes ′ [′ ′]′ are terminals.
$x*$	This signifies zero or more occurrences of the element x. In simpler terms, the element x can appear zero times (absent) or any number of times when constructing the grammar rule.
$x,+$	A comma-separated list of one or more x's.
|	This symbol represents choice or alternatives. It separates two or more possibilities within a rule.

```
<program> -> prog: (decl | expr)+ EOF # Program
 ;
<declaration> -> decl: ID ':' (INT_TYPE | DOUBLE_TYPE) '=' expr
#Declaration
 ;
<method_call> -> methodCall: (PRINT | TVA) LPAREN expr RPAREN
 ;
<IF_expr> -> ifExpr:
 IF expr THEN expr (ELSE expr)? # IfExpression
 ;
<statement> -> expr:
 methodCall # MethodExprCall
 | ifExpr # IfExprStatement
 | expr EQULITYOP expr # EqualityComparison
 | expr RELATIONALOP expr # RelationalComparison
 | expr '*' expr # Multiplication
 | expr '/' expr # Division
 | expr '+' expr # Addition
 | expr '-' expr # Subtraction
 | BOOL # Boolean
 | STRING # String
 | ID # Variable
 | NUM # Number
 ;
<type> -> INT_TYPE : 'INTEGER';
<type> -> DOUBLE_TYPE : 'DOUBLE';
<bool_literal> -> BOOL : 'TRUE' | 'FALSE';
<relation_comp> -> RELATIONALOP : (GT | LT | GTE | LTE);
<ecuality_comp> -> EQULITYOP : (EQ | NEQ);
<string_literal> -> STRING : '"' (~["])* '"';
COMMA : ',' ;
LPAREN : '(' ;
RPAREN : ')' ;
SEMI : ';' ;
<eq_op> -> EQ : '==' ;
<eq_op> -> NEQ : '!=' ;
<rel_op> -> GT : '>' ;
```

```
<rel_op> -> LT : '<' ;
<rel_op> -> GTE : '>=' ;
<rel_op> -> LTE : '<=' ;
IF : 'if';
ELSE : 'else';
THEN : 'then';
PRINT : 'print';
TVA : 'tva';
FUNC : 'function' ;
<alpha> -> ID : [a-z][a-zA-Z0-9_]*;
<digit> -> NUM : [0-9]+ ('.' [0-9]+)? ;
<comment> -> COMMENT : '//' ~[\r\n]* -> skip; // ~ - negation, skip
everything except \r or \n
<whitespaces> -> WS : [ \r\t\n]+ -> skip; // skip whitespaces
```

This is a program created according to grammar rules:

```
i : INTEGER = 5;
print(i);
tva(i);
j : INTEGER = tva(i);
print(j);
```

The program adheres to your defined grammar, with the following structure:

- Declaration (variable i)
- Expression (function call print(i))
- Expression (function call tva(i))
- Declaration (variable j)
- Assignment (variable j)
- Expression (function call print(j))

All program elements (variables, functions, operators) align with the grammar rules.
This is the result:

```
5.0
Value 5.0 has TVA: 1.0
1.0
```

The parse tree analysis (Fig. 1) outlines the structure of the DSL program. The root node 'prog' branches into sequences reflecting the program's flow. It begins with a declaration of variable 'i' as an integer initialized to 5, followed by a print statement outputting 'i'. Next, an expression involving a function call 'tva' takes 'i' as its argument, though the function's behavior remains undefined. Another declaration sets variable 'j' as an integer, initialized via a similar function call. The program concludes with a print statement for 'j', replicating the earlier output structure.

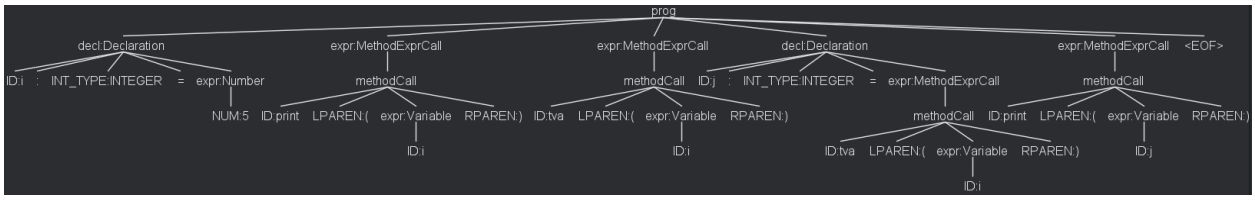

Figure 1. The parse tree.

Based on the tree structure, the code follows the expected grammatical constructs for variable declarations, expressions, function calls, and print statements.

Conclusions

Moldova faces a choice: stick with the outdated, stressful tax system or embrace a new approach through a Domain-Specific Language. This DSL simplifies taxes, using plain language to make them accessible to all, not just experts. By securely integrating with government systems, it would reduce paperwork and ensure fair tax payments, aiding in responsible financial planning for individuals and businesses alike. The shift to a DSL transforms the government's role to a partner, fostering trust through tax transparency. Having defined the DSL's grammar, analyzed the domain, and constructed the parsing tree, this move towards efficient, reliable tax systems demonstrates Moldova's commitment to innovation and business-friendly policies, attracting investment and job creation. Collaboration among technologists, tax professionals, government officials, and the public is crucial to tailor the DSL to Moldova's evolving needs. Embracing this change represents a step towards a future where smart financial decisions are within everyone's reach, marking a new chapter in Moldova's tax narrative.

References

[1] Serviciul Fiscal de Stat. (2023).Tax Code. [Online]. Available: https://sfs.md/en/page/tax-code

[2] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. ACM Comput. Surv., 37(4):316-344, 2005.

[3] Eelco Visser. WebDSL: A case study in domain-specific language engineering. In R. Lammel, J. Saraiva, and J. Visser, editors, Generative and Transformational Techniques in Software Engineering (GTTSE 2007), Lecture Notes in Computer Science. Springer, 2008.