DOMAIN SPECIFIC LANGUAGE FOR PLC

Inga PALADI[1*], Vladislav FRUNZE[1], Vlad URSU[1], Roman TRANDAFIR[1], Irina RACOVCENA[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

*The corresponding authort: Inga Paladi, inga.paladi@isa.utm.md

Scientific coordonator/ mentor : Irna COJUHARI, conf. univ., dr., Technical University of Moldova

Abstract. *This article discusses the potential of the domain specific language (DSLs) for programmable logic controller (PLCs), it presents the grammar and it is provided an example of code that could be used to create programs for the PLC. The grammar includes essential elements such as input, memory, and output variables, logical operations, and program structure. DSLs for PLCs offer several benefits, including efficiency, maintainability, and flexibility, making them valuable tools in the development of industrial automation.*

Keywords*: domain specific language, programmable logic controller, grammar, syntax, parser.*

Introduction

Programmable Logic Controllers (PLCs) are used as a vital component in modern industry which helps to control the manufacturing processes. In order to use the PLCs, it is needed to program them and the most used programming languages for it are Ladder Logic, Function Block Diagram, Sequential Function Charts, Structured Text, Instruction List [1].

Nevertheless, as the industry evolves, more complex programs are needed. And to do them in simpler ways it is necessary to develop a domain-specific language (DSL). A DSL is a programming language that specializes in a particular domain of control systems.

DSLs for PLCs give the possibility to create more maintainable, readable, and, in general, simpler programs that will govern industrial processes. DSLs provide us high-level abstractions which could help to program complex control logic in a more intuitive manner. As a feature, DSLs for PLCs could have code generation, which would significantly reduce the time necessary for PLC programming.

Generally, DSLs for PLCs have great potential for the development of industrial automation. They can provide efficiency, maintainability, and flexibility over ”ladder logic”. Therefore, while PLCs will have a critical role in the present manufacturing processes, the DSLs will become more helpful and in demand.

Introduction in PLCs

PLCs, or Programmable Logic Controllers, industrial-grade computers that can be programmed to operate specific control tasks. Some characteristics of the PLC include significantly reduced amounts of hardwiring used for traditional relay control circuits, easy programming, and installation, fast control, network interoperability, easy troubleshooting and testing, etc.

A programmable logic controller is supposed to work with a variety of input and provide output configurations, operate over a wide array of temperatures, be immune to electrical noise, and be impact and vibration resistant. Typically, battery-backed or non-volatile memory is where software for controlling and operating industrial process equipment and machinery is kept. A PLC is an illustration of a real-time system since the output of the system it controls is dependent upon the input circumstances.

The PLC was first used to replace relay logic, but due to its wide range of capabilities, it is now utilized in a variety of applications. A PLC can execute relay switching operations as well as other applications like timing, counting, calculating, comparing, and the processing of analog signals because of the fact that its structure is built on the same concepts as those used in computer architecture[1].

When discussing PLCs in relation to a traditional relay kind of control, programmable controllers have a number of benefits. To carry out a given task, relays must be hardwired. The relay wiring has to be altered or modified as soon as the system requirements change. In severe circumstances, such as the automotive industry, whole control panels had to be rebuilt because it was not practical to rewire the old panels for each model change. The programmable controller has significantly reduced the amount of hardwiring required for traditional relay control circuits. Compared to other relay-based process control systems, it is compact and reasonably priced. Relays are still a part of contemporary control systems, but logic is rarely applied to them.

Grammar

An example of a grammar that could help create the Domain Specific Language (DSL) for programming Programmable Logic Controllers (PLCs) is given below [2].

```
<program> ::= <start> <programName> <inputDeclarations> <outputDeclarations>
<memoryDeclarations> <logicStatements> <endProgram>
<start> ::= "BEGIN"
<endProgram> ::= "END"
<programName> ::= <identifier>
<identifier> ::= <alpha> <alphaNum>*
<alpha> ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z"
<alphaNum> ::= <alpha> <digit>
<digit> ::= "0" | "1" | "2" | "3" | ... | "9"
<inputDeclarations> ::= "INPUT" <var> {"." <var>} ";"
<outputDeclarations> ::= "OUTPUT" <var> {"." <var>} ";"
<memoryDeclarations> ::= "RAM" <ram> {"." <ram>} ";"
<var> ::= "I" <x1> <x2> | "Q" <x1> <x2> | "M" <x1> <x2>
<x1> ::= "0"
<x2> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<ram> ::= <CN01> "." "AND" <coils> | <CN02> "." "OR" <coils> | <CN03> "." "XOR"
<coils> | <CN04> "," "NOT" <coils>
<CN01> ::= "CN01"
<CN02> ::= "CN02"
<CN03> ::= "CN03"
<CN04> ::= "CN04"
<coils> ::= <coil> {"." <coil>}
<coil> ::= <var>
<logicStatements> ::= <statement> {"." <statement>}
<statement> ::= <var> ":=" <logicOp>
<logicOp> ::= <var> "AND" <var>
          | <var> "OR" <var>
          | <var> "XOR" <var>
          | "NOT" <var>
```

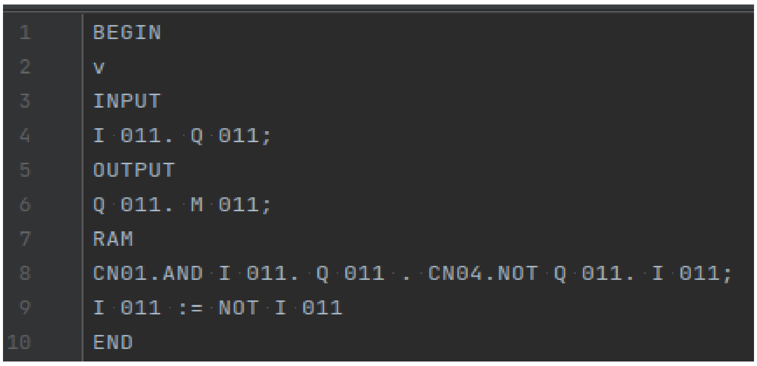
The most important parts of this grammar are the following:

1. <program> production rule: This rule suggests the starting point of this grammar and defines the whole structure of a PLC program. It is made of several sub-rules for a program name, logic statements, input, and output declarations, and so on.
2. <var> production rule: This rule suggests the syntax and variable declaration. The DSL in question has three types of variables: input (I), output (Q), and memory (M). The variables are defined by first writing a single letter and then two digits. In this way, it will identify accurately where the inputs come from and where the outputs should go.
3. <ram> production rule: This rule denotes the syntax for memory declaration. The memory can be identified by a four-letter code in combination with logical operations (AND, OR, XOR, NOT) and a list of coils. Coils imply the variables which can be connected together to create complex operations.
4. <logicStatements> production rule: This rule attributes the result of a logical operation to a variable. The logical operation can be a simple AND, OR, XOR, or a NOT operation applied to a variable.

All in all, this DSL with the above grammar is created in order to help engineers in programming PLCs with a programming language that is similar to the ladder logic, but which is easier to learn and to use.

Code example

A presentation of a code snippet, that was constructed using the DSL described in this paper is shown in Fig. 2:



```
BEGIN
v
INPUT
I 011. Q 011;
OUTPUT
Q 011. M 011;
RAM
CN01.AND I 011. Q 011 . CN04.NOT Q 011. I 011;
I 011 := NOT I 011
END
```

Figure 1. DSL code example

Derivation Tree

A derivation tree or parse tree is a graphical representation that illustrates the way in which strings in a language are being derived, taking in consideration the rules of the grammar. Fig. 3 illustrates the derivation tree obtained from an example of code, by using ANTLR as a tool for parsing [3].

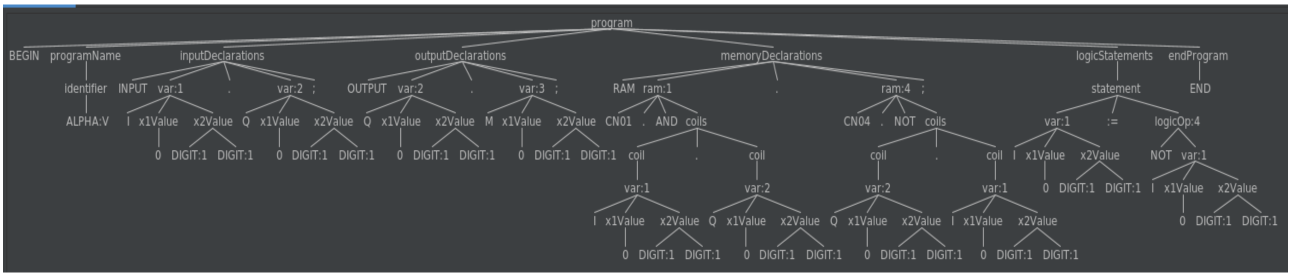

program
BEGIN programName inputDeclarations outputDeclarations memoryDeclarations logicStatements endProgram
identifier INPUT var:1 . var:2 ; OUTPUT var:2 . var:3 ; RAM ram:1 . ram:4 ; statement END
ALPHA:V I x1Value x2Value Q x1Value x2Value Q x1Value x2Value M x1Value x2Value CN01 . AND coils CN04 . NOT coils var:1 := logicOp:4
0 DIGIT:1 DIGIT:1 0 DIGIT:1 DIGIT:1 0 DIGIT:1 DIGIT:1 0 DIGIT:1 DIGIT:1 coil . coil coil . coil I x1Value x2Value NOT var:1
var:1 var:2 var:2 var:1 0 DIGIT:1 DIGIT:1 I x1Value x2Value
I x1Value x2Value Q x1Value x2Value Q x1Value x2Value I x1Value x2Value 0 DIGIT:1 DIGIT:1
0 DIGIT:1 DIGIT:1 0 DIGIT:1 DIGIT:1 0 DIGIT:1 DIGIT:1 0 DIGIT:1 DIGIT:1

Figure 2. Parsing tree for program example

Conclusion

In conclusion, Programmable Logic Controllers (PLCs) are essential components in modern industry for controlling manufacturing processes. They are easy to program, fast, interoperable, and can operate in a wide array of conditions.

Furthermore, the most commonly used programming language for PLCs is "ladder logic", but as industry evolves, more complex and maintainable programs are needed. Domain-specific languages (DSLs) provide high-level abstractions that can simplify the programming process, reduce programming time, and increase maintainability.

DSLs for PLCs can be designed with a grammar that includes essential elements such as variables, logical operations, and program structures. By utilizing a DSL for PLCs, the programs created can be more maintainable, efficient, and flexible compared to those created with ladder logic.

Although relays are still used in contemporary control systems, PLCs have significantly reduced the amount of hardwiring required for traditional relay control circuits. PLCs have become essential for replacing relay logic in many applications.

As a result, with the potential benefits of DSLs for PLCs, it is likely that their demand will increase as industry continues to evolve, and they will become an increasingly vital component in industrial automation.

References:

1. PETRUZELLA, D.F. Programmable Logic Controllers 4th Edition, McGraw Hill, 2010, p.2.
2. AHO A.V., LAM M.S., SETHI R., ULLMAN J. D., *Compilers, Principles, Techniques, & Tools, second Edition*, p. 42-45.
3. PARR T., *The Definitive ANTLR Reference, Building Domain-Specific Languages*, 2007.