DOMAIN SPECIFIC LANGUAGE FOR MICROCONTROLLERS

Dana SPEIANU, Eugen-Cătălin CHISELIOV, Vera ȘVEȚ, Maria-Mirabela CHIRICĂ

Technical University of Moldova

Abstract: *In this paperwork is presented the first phase of development of the Domain Specific Language for microcontrollers in the domains where microcontrollers are used. In order to start to implement this Domain Specific Language, it was analyzed and presented the grammar of the language, its syntax, and semantic rules. In this article is also presented an example of an algorithm which follows the grammar, the syntax and the semantic rules of this language. As a tool which analyses the given language was used ANTLR, this tool makes it easier to parse the algorithm syntactically and generate the parse tree.*

Keywords: *Domain Specific Language, microcontroller, grammar, irrigation modes, lighting modes, Arduino.*

Introduction

In our days, microcontrollers tend to spread their domain of application being used in different activities. Embedded systems perform control of physical components in various fields. Microcontrollers are used in a lot of areas of embedded systems, as robotic control systems or smart devices. The quantity of various hardware platforms is expanding with a spread of embedded systems. From another point of view, in software development is increasing the number of not qualified programmers. In this way, a person who is involved in a specific domain will easily learn a Domain Specific Language (DSL) than to learn a programming language with a big complexity as C or Java.

This is because the DSL gives the possibility to the user to program a microcontroller with more familiar terms. The DSL gives the possibility that more people with different level of programming skills could interact with them.

This article presents the development of a specific language for microcontrollers. The process includes the creation of DSL for one of the most popular platforms for non-industrial microcontrollers - Arduino, ensuring the execution of program schemes on the hardware platform.

In this paper are identified some areas where DSL for microcontrollers will be applied: traffic lights and irrigation. It's very easy to make a program for these as can be made a lot of algorithms that represent different modes of lighting or irrigation. In the next pages, you can find the description of our DSL for microcontrollers, grammar, and semantic rules.

The traffic light control is a domain where can be made some modes, for example during day time a traffic light can have one mode, but during night time another one (because there are few cars). Also, there are rush hours when in the city are many traffic jams then the traffic light is more active in order that traffic would be well organized in order to avoid traffic jams.

Another interesting area where this DSL can be used is irrigation. Irrigation is the application of controlled amounts of water to plant at needed intervals. The flow of irrigation water in the canals must always be under control. For this purpose, we will use the DSL to program the microcontroller to control how long, at which hour and in which mode the plants must be irrigated.

1. Grammar

In this paper are presented the analysis of the domains where microcontrollers are used, especially where different modes are applied. It is identified a lot of domains but decided to choose those two described above: traffic lights and irrigation. In order to create an effective DSL, it is proposed to make it such that it will be easy to use by a user with weak programming skills. The terminal symbols from grammar can be found in Fig.1.

V_T = {
:, (,), ., , , algorithm, step, program, set, run, schedule,
end, from, to, on, off, intermittent, 0..9, A..Z, a..z
}

Fig.1. Terminal symbols

Then non-terminal symbols are shown in figure 2.

```
V_N = {
      <source code>, <pin_definition>, <algorithms_definition>,
      <schedules_definition>, <program>, <set_pin>, <num>, <alfa_num>,
      <pin_identifier>, <algorithm>, <step_list>, <step>,
      <instructions_list>, <instruction>, <on>, <off>, <intermittent>,
      <schedule>, <time_range_list>, <time_range>, <time>, <digit>,
      <program>, <run_list>, <run>, <alfa>
      }
```

Fig.2. Non-terminal symbols

The set of production rules can define how a program can derive as in the following figure (Fig.3).

```
P={
      <source_code> →  <pins_definition><algorithms_definition>
      <schedules_definition><program>

      <pin_definition> →  < set_pin >+
      <set_pin> →  set <alfa_num> <pin_identifier>
      <pin_identifier> →  <num>

      <algorithms_definition> →  < algorithm >+
      <algorithm> →  algorithm (<alfa_num>): <step_list> end
      <step_list> →  < step >+
      <step> →  step(<num>): <instructions_list> end
      <instructions_list> →  <instruction>*
      <instruction> →  <on> | <off> | <intermittent>
      <on> →  on <alfa_num>
      <off> →  off <alfa_num>
      <intermittent> →  intermittent(<num>, <num>) <alfa_num>

      <schedules_definition> →  < schedule >+
      <schedule> →  schedule (<alfa_num>): <time_range_list> end
      <time_range_list> →  < time_range >+
      <time_range> →  from <time> to <time>
      <time> →  <num>:<num>:<num>.<num>

      <program> →  program <run_list> end
      <run_list> →  run*
      <run> →  run <alfa_num> on <alfa_num>

      <alfa_num> →  <alfa> | <alfa_num><alfa> | <alfa_num><num>
      <num> →  < digit >+
      <digit> →  [0 ,..,9]
      <alfa> →  [a ,.., z]
}
```

Fig.3. Productions

2. Semantic rules

Semantic rules add additional constraints on the valid program besides the constraints implied by the grammar. Semantics can be partitioned into static semantics and execution semantics. Static semantics are implemented by the constraints and type system rules. Execution semantics denote the observable behavior of a program as it is executed [1]. Constraints are simply boolean expressions that check some property of a model. The following list show the semantic rules of this DSL:

- The names of a set of attributes of some entity are unique(these can be: pin name, algorithm name, schedule name).
- Every non-start state of a state machine has at least one incoming transition.
- The value which is attributed as pin number after pin_name must be of type **int**.
- In the **step** body, all pins name should be a valid pin identifier which was declared before.

- In the **step** body all pins must be declared with a single state(**on, off, intermittent**).
- In the program section, after keyword **run** should be a valid algorithm name and after keyword **on** should be a valid scheduler name.
- As argument at step declaration in round brackets must be the time in milliseconds and of type **int**.

In order to understand how does the DSL works, a sample program will be explained (Fig.4). Firstly, the user defines the pins for every plant, then he defines an algorithm and a schedule. The algorithm is named algorithm1 and has 2 steps:

1. The first step is running for 30 sec with on mode for apple1 and off mode for apple2 and strawberry. This means that during this time apple1 will be wet, but apple2 and strawberry will not.
2. The second step is running for 2 sec with the off mode for apple1 and strawberry and intermittent mode for apple2. The intermittent mode is described as 0.5 sec on mode and 1 sec off mode.

Then the schedule called schedule1 where the user has defined two intervals. These intervals determine intervals during which a specific algorithm will run. After schedule definition, the user must execute the source code and this starts with the program keyword. In the program field, it's running the algorithm1 on intervals specified in schedule1. The figure (Fig. 4) below represents an irrigation mode of how plants can be wet. It is easy to create new irrigation modes in dependence on the user's needs.

```
set apple1 0
set apple2 1
set strawberry 2

algorithm (algorithm1):
    step(30000):
        on  apple1
        off apple2
        off strawberry
    end

    step(2000):
        off apple1
        intermittent(500, 1000) apple2
        off strawberry
    end
end

schedule (schedule1):
   from 00:00:00.000 to 05:00:00.000
   from 21:00:00.000 to 23:00:00.000
end

program
   run algorithm1 on schedule1
end
```

Fig.4. A sample program using the DSL for microcontroller

3. Solution

The solution for irrigation is based on the following scheme (fig 5). The user writes his algorithms in DSL, the code is analyzed syntactically and semantically in ANTLR and using Java and jArduino the generated code is uploaded to an Arduino device, then the watering device or another device works by the algorithm and steps which are defined in DSL.

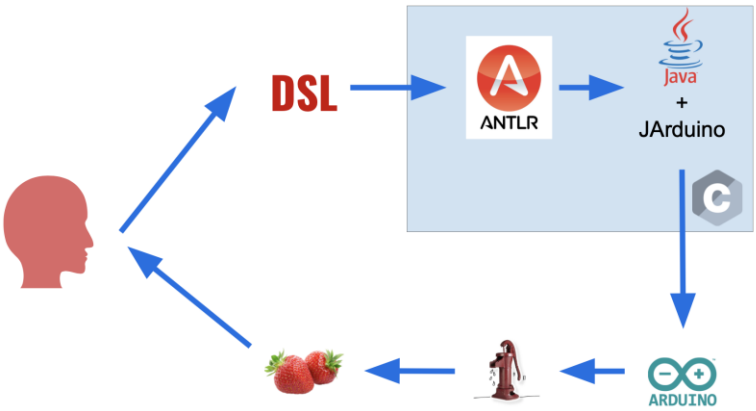

Fig.5. Solution scheme.

Conclusions

Nowadays controllers and especially microcontrollers are widely used which means that demand for IT specialists is constantly increasing. This is why in some specific areas, domain-specific areas, this DSL for microcontrollers will become a very powerful tool and will provide a very good solution for this specific area. So, in this way, having very basic instructions, almost everyone will be able to create an irrigation system from scratch and easily manage it.

References

1. Markus Voelter with Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart Kats, Eelco Visser, Guido Wachsmuth, DSL Engineering, Designing, Implementing and Using Domain-Specific Languages, 556 pages, Source: Dslbook.org, 2013.