Easy Implementation of Domain Specific Language using XML

Marek NOVÁK

Dept. of Computers and Informatics, FEI TU of Košice, Slovak Republic
marek.novak@tuke.sk

I. INTRODUCTION

There are some situations when software users, usually people without programming skills, have to inject some rules into the system or define conditions to achieve required functionality. To teach a user at least the basics of programming language in which the system is developed and allow to define rules directly in source code can be very inefficient. A configuration using domain specific language (DSL) seems to be a better solution. DSL is a computer language that is targeted to a particular kind of problem, rather than a general-purpose language which is aimed at any kind of software problem [1]. In comparison to general-purpose programming languages DSLs provide expressibility targeted directly to users domain and therefore they are easier to learn and use. Apparently graphical user interface (GUI) with drag-and-drop components provides the most intuitive way to define the rules for user who is non-programmer.

Although GUI makes an average user work productively, an expert or a user with advanced skills in application domain can be slowed down [2]. Moreover, to develop robust GUI may be more complex task than building the system itself if the system is not expected to be very complex [3]. In general DSL development may be considered a difficult task because there are both domain knowledge and language development expertise necessary. Existing technologies as XML and parsers like Java Architecture for XML binding (JAXB) [4] with the assumption that rule set is not large and rules by themselves are not very complicated makes DSL development makes much easier.

The purpose of this paper is to show an example how to simply build DSL for a specific purpose. At Technical University of Košice (TUKE) we have developed our own simple DSL when working on project MonAMI to support research at Department of Biomedical Engineering, Automation and Measurements, Faculty of Mechanical Engineering.

II. DEVELOPING DSL WITH XML

As mentioned above, DSLs are designed to be useful for a specific task in a fixed problem domain. An advanced computer user uses daily DSL: configuration file, makefile, CSS etc. Expression capabilities of DSL are fairly limited. They are focused on a certain type of problem or domain, as its name implies, and on expressing narrow set of solutions within the context of that limited scope [2]. Simplicity is very important feature of DSL. A person familiar with domain must easily understand the domain language. Keywords must be very close to user’s vocabulary. Syntax should also be simple and clear to facilitate user’s work and focus on domain problems that user tries to solve. Depending on how DSL is implemented, we classify it on external or internal DSL [1].

External DSL – is designed to be independent on any particular language. An author of such language must decide about syntax, grammar and the way to parse the syntax. Any language and tools can be used to implement this DSL. For instance it can be Java and Groovy. When using external DSL, author is free to define syntax as he likes – to use symbols, operators, constructs and structures, which fit the best to the domain. On the other hand, it is necessary to define grammar for the language, to create a compiler to parse and process the syntax and map it to the semantics that is expected. Flexibility provided is an advantage, but it may be a really complex task to implement DSL well.

Internal DSL – is designed and implemented using a host language. The advantage is that author does not have to worry about grammar, parsers and tools. However, it brings disadvantages in form of constrains and limitations of the host language. Internal DSL provides easier implementation at the expense of flexibility.

The example of external DSL is also ANT build file [5], which uses XML representation. XML file is processed by the ant utility using the XML parser. Ant's vocabulary contains various terms, such as target and properties, which are valid in the domain and context of compiling and bundling code.

XML brings many advantages to DSL development, though it is not appropriate everywhere. XML-based DSL, grammar is described using DTD or XML schema [6] where nonterminals are analogous to elements and terminals to data.
content. Element definitions determine grammar rules when the element name is the left-hand side and the content model is the right-hand side. XML documents form a tree structure that starts at the root, which corresponds to start symbol in grammar. DSL defined by Backus notation (BNF, EBNF) may be transformed easily to XML tree structure. There is no need to make a big effort to create a parser, since DOM parser or SAX (Simple API for XML) tool already provides this functionality. Since the parse tree can be encoded in XML as well, XSLT transformations can be used for code generation. Therefore, XML and XML tools can be used to implement a programming language compiler [7], [8].

III. MONAMI SERVICES

An intention to create our own DSL arose from a need of service configuration when cooperation on MonAMI project. The configuration should have been made primarily by postgraduate students from Department of Biomedical Engineering, Automation and Measurements without programming knowledge.

MonAMI - Mainstreaming on Ambient Intelligence project [9] (funded by the EU is 6th framework program. It was built on an assumption there will be a high percentage of population over the age over 65 who are still active, computer literate with the ambition to maintain their quality of life. Mission of the project is to improve daily activities and the quality of life of elderly and disabled people at home. It is based on mainstream systems and platforms which create one complex system comprising of different technologies. Services and applications developed with a “Design for All approach” are from following areas [10]:

- Home control, personalized communication interface, activity planning.
- Health control, medication.
- Safety and security at home, visitor validation, activity detection.
- Communication and information.

Department of Biomedical Engineering, Automation and Measurement took a part as centre for testing and validating technologies and services. Selected elderly people and people with disabilities will test services in a laboratory – Feasibility and Usability (FU) centre where the whole system is installed. There are different types of sensors for temperature, humidity, light level, motion and gas detection and different types of actuators as light, shutter and alarm actuator installed. People involved into project research are responsible for testing, satisfaction evaluation, measurement, services adjustment and configuration. Testing is divided into two phases. First phase comprises of evaluation, bug resolving, user insights incorporation and system adjustments to user’s needs in laboratory. After this phase is finished, the system will be installed in real households. The role of people working on project is to prepare questionnaires for interviews with users, evaluate the answers, present elderly and disabled people offered MonAMI services in order to understand their purpose and functionally; and finally control, configure and create new services depending on user’s ideas in the testing phase [11].

MonAMI service is an activity with strictly defined behavior. Implementation of services diverse:

- Collect data from sensors.
- Actuate devices when pre-defined conditions are fulfilled.
- Provide information to users in friendly way.
- Actuate devices by human input.

The services could be divided into two groups depending on an action taken when measured values exceed defined thresholds or when some values are detected e.g. smoke, gas. The first group represents actions taken by humans – carer responsible for disabled person is at the moment in a shop and is noticed by SMS about fall or heart attack of treated person. He can immediately call an ambulance or just check up the state of treated person depending on character of information. The second group represents actions executed automatically. In case smoke and high temperature is detected in a kitchen, fire department is automatically informed about this situation.

The whole system is implemented in Java programming language based on component oriented architecture OSGi. More detailed description of system architecture is not in the scope of this article but it is important to sketch service implementation. Particular sensors can be understood as services as well, which purpose is nothing more than reading values from sensors and providing them to other services. Each service is one component in the system that could be added or removed.

Behavior of majority of services can be simply defined by following formula: if values from sensors exceed thresholds, take an appropriate action. This condition has to be defined in particular service source code.

Here the problem has arisen, because thresholds could be made configurable by some simple interface, however, to add only one extra AND/OR condition could be a serious problem. To simplify the testing we have created a component, which parses XML configurable file where all services are defined by DSL. Structure of XML is defined by XSD and JAXB has been used as a parser.

IV. FLEXIAMI DSL

FlexiAMI is the name of component, which provides configuration capabilities for MonAMI services (“flexi” as flexibility in definition and AMI is taken from MonAMI) [12]. Keywords and service definition principle are close to user domain however users are tightened up by a relatively larger set of rules. They are straightforward, therefore easy to learn. Rules format is based on XML, which has been used because of many advantages, as mentioned also in previous part, the most important advantages are:

- hierarchical structure of final configuration file;
- self-describing and simple syntax;
- existence of validators;
- existence of APIs parsing the XML file;
- easy implementation in Java.

To provide the whole formal grammar is not necessary, however simple showcase can be very helpful. The configuration itself comprises of two parts – definition of
sensors and actuators represents first part and rules definition is the second.

First part could be following:

```xml
<sensor>
  <name>temperature</name>
  <type>TemperatureSensor</type>
  <location>Kitchen</location>
</sensor>

<actuator>
  <name>alarm</name>
  <type>AlarmActuator</type>
  <location>Kitchen</location>
</actuator>
```

Each sensor and actuator has to be declared by its unique identificator `<name>`, which is used consequently in rules, `<type>` specifies sensors and actuators according to MonAMI naming and `<location>` determines room where devices are installed.

Rules follow this diagram:

![Rule behavior diagram](image)

Below is an example of rule, which uses temperature sensor, smoke sensor and alarm actuator. When smoke and high temperature is detected at once, alarm actuator is turned ON. Alarm actuator can represent beeper in house, alarm connected directly to firehouse station or some other notification method as SMS.

Definition is following:

```xml
<service>
  <name>FireService</name>
  <type>AmbientMonitoringAlarm</type>
  <condition>
    <and>
      <cond oper="eq" val="TRUE">smoke</cond>
      <cond oper="ht" val="40.0">temperature</cond>
    </and>
  </condition>
  <action>
    <actIf val="ON">alarm</actIf>
    <actElse val="OFF">alarm</actElse>
  </action>
</service>
```

Each service has its name and type according to MonAMI specification. Other bundles in OSGi system can find and use this service by defined type, which tells something about its behavior.

Condition comprises of three values: sensor identifier, operation and value to compare. The meaning of `<cond oper="eq" val="TRUE">smoke</cond>` is: *if smoke is detected in kitchen then TRUE*. The result of conditional expression can be *TRUE* or *FALSE*. The content of `<cond>` element refers to identifier in first part service definition, where is specified which sensor from which room should be used. Possible operations `oper="eq"` are: `eq`, `nq`, `ht`, `lt`, `he`, `le` denoted to: equal, not equal, higher than, less than, higher equal, less equal respectively. Compare value `val="TRUE"` depends on used sensor. Some sensors provide only two values – detected/not detected and some numeral values. Element `<and>` represents Boolean logical operator. Conditional expressions surrounded by this operator are translated as: `cond1 AND cond2 AND cond3`.

```xml
<and>
  <cond ... >...</cond>
  <cond ... >...</cond>
  <cond ... >...</cond>
</and>
```

Logical operators can be embedded to define condition evaluation priority. For example (1, 2, 3 here means conditional expression number, not sensor identifier):

```xml
<and>
  <cond ... >1</cond>
  <cond ... >2</cond>
  <or>
    <cond ... >3</cond>
    <cond ... >4</cond>
  </or>
</and>
```

It is evaluated as: `cond1 AND cond2 AND (cond3 OR cond4)`.

Condition embedding allows defining any type of logical expression. Elements `<and>` and `<or>` can surround arbitrary number of conditional expressions.

Execution of some services may depend on more aspects than only on sensor values. An example is time when sensor value is measured.

Elderly person is used to wake up at 8:00 am. When there is no motion detected in an hour after 8:00 am, carer is informed about this situation by SMS. This service is defined accordingly:

```xml
<service>
  <name>WakeUPService</name>
  <type>PersonPresenceDetector</type>
  <condition>
    <and>
      <cond oper="FALSE" val="TRUE" duration="3600">motion</cond>
      <cond oper="ht" val="08:00:00">time</cond>
    </and>
  </condition>
  <action>
    <actIf val="ON">sms</actIf>
  </action>
</service>
```

In the first conditional expression there is one additional attribute `duration="3600"`. This attribute determines how long measured value has to be unchanged. In this case, no motion is detected during one hour. Duration value is set in seconds, because there could be some services when only
some seconds are needed. For example when person leaves the toilet and no movement is detected for 20 seconds, the light is turned off:

```xml
<cond oper="ht" val="08:00:00" >time</cond>
```

is translated as: if actual time is higher than 8:00 am then TRUE.

Service action comprises of two action expressions. One is executed when condition result is TRUE (actIf) and the other is executed when condition result is FALSE (actElse). There can be some additional properties to service added by:

```xml
<properties>
  <prop name="name">value</prop>
</properties>
```

**CONCLUSION**

FlexiAMI component becomes very helpful in the phase of MonAMI services testing. DSL has enabled programmers to delegate configuration responsibility to people working on MonAMI project who are non-programmers. For that reason it was not necessary to change the application code after each new requirement of user. XML format and used elements have been descriptive enough and FlexiAMI users get used to them very quickly. JAXB parser significantly simplifies the DSL development process, and therefore XML technologies seem to be very effective for DSL creation for this purpose. There are also some other innovative approaches of DSL development as Annotation Based Parser Generator [13].

**ACKNOWLEDGMENT**

The authors would like to thank the European Commission for the support within the 6th Framework Program by the grant of the integrated project within the priority 2.3.2.10 e-Inclusion "Mainstreaming on Ambient Intelligence" - MonAMI IST-5-0535147, and Coordination action "Design for All for e-Inclusion" DfA@eInclusion IST 033838.

**REFERENCES**


