Abstract
After 7 years in operation the Software Interlock System (SIS) has become an indispensable and mission-critical control tool covering many operational areas from general machine protection to diagnostics. The growing number of running instances as much as the size of existing configurations has increased both the complexity and maintenance cost of running the SIS infrastructure. In response to those issues, a new way of controlling the system has been implemented that simplifies the configuration process by making it faster, more user friendly and understandable for wider audiences and domain experts alike. As one of the possible choices the Groovy scripting language has been considered as being particularly well suited for writing a custom Domain-Specific Language (DSL) due to its built-in language features like native syntax constructs, command chain expressions, hierarchical structures with Built-in Syntax Trees (AST) transformations. This document explains best practices and lessons learned while introducing an accelerator physics domain oriented DSL language for the configuration of the Software Interlock System developed by the Data & Application Section at CERN.

Configuration Process Comparison

<table>
<thead>
<tr>
<th>XML</th>
<th>DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Processing</td>
<td>DSL Interpretation</td>
</tr>
<tr>
<td>Source extraction from XML documents</td>
<td>Language Specification</td>
</tr>
<tr>
<td>Source code instrumentation</td>
<td>Abstract Syntax Tree Transformations</td>
</tr>
<tr>
<td>Compilation on demand using Java API</td>
<td>Transparent compilation using Groovy Shell</td>
</tr>
<tr>
<td>Bootstrap</td>
<td></td>
</tr>
</tbody>
</table>

Overview of the workflow in both configurations

Domain Model

The SIS system helps protecting the machine by surveying the state of the devices. It continuously evaluates user-defined conditions and dumps or inhibits the beam production if an abnormal situation is detected. The SIS is designed to protect the machines against repetitive faulty conditions limiting radiation, extending the equipment lifetime and making the machine diagnostics much easier.

DSL Language Features

```
  def virtualDev = "[on|off]",
  def logicalCondition = "[NOT|AND|OR]",
  def hardwareDev = "[on|off]",
  def isii = "[string name, string virtualParam]",
  def timeOut = "[timeOut]
```

Example configuration in XML and DSL

```xml
 xsi:schemaLocation="sisi/channels.xsd">
  <config>
    <configId>ISIC_D_4811149</configId>
    <device>
      <virtualDev name="virtualDev1" virtualParam="virtualParam1"/>
      <hardwareDev name="hardwareDev2"/>
      <isii name="isii1" virtualParam="virtualParam2"/>
      <timeOut name="timeOut1"/>
    </device>
  </config>
</xml>
```

CONCLUSIONS

Taking the DSL approach for the SIS configuration proved itself to be the right choice in practice. Its interoperability with Java on the binary level is a great advantage opening ways for the implementation of the DSL in a mixed Java & Groovy mode. Also its build-in features targeting directly the DSL construction make the design of such language much easier. The corresponding files are much smaller and more readable comparing to their XML counterparts. At the same time the configuration is more concise with all its entities represented as Groovy code constructs. Overall it improves significantly the level of user satisfaction and maintainability of the system as a whole.