ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA163

A MODEL-DRIVEN GENERATOR TO AUTOMATE THE CREATION OF HMIS FOR THE CERN GAS CONTROL SYSTEMS

T.Bato, G.Thomas, F.Varela CERN, Geneva, Switzerland

Abstract

A total of 33 gas control applications are currently in production in the LHC Experiments and the CERN accelerator complex. Each application contains around fifty synoptic views and hundreds of plots. In this paper, the entirely model-driven approach followed to generate all these HMIs is presented. The procedure implemented simplifies the creation of these graphical interfaces; allowing the propagation of changes to all visualizations at once in a coherent manner, thus reducing the long-term maintenance effort. The generation tool enables the creation of files of similar content based on templates, specific logic (rules) and variables written in simple user-defined XML files. This paper also presents the software design and the major evolution challenges currently faced, how the functions performed by the tool, as well as the technologies used in its implementation, have evolved while ensuring compatibility with the existing models.

INTRODUCTION

The gas systems are essential for the LHC Experiments and accelerator complex, as they have to provide their corresponding chambers with the appropriate proportion and correct gas mixture, hence a software control layer is mandatory for their monitoring and control. Since 2005 the requests for new gas systems and upgrades of existing ones are increasing.

Thanks to the model driven approach [1] and common standards adopted in 2005, the development, support and maintenance of the software control layers can be achieved with minimal manpower and costs.

These gas control systems are independent application instances, which consist of a supervision layer based on a SCADA System (SIEMENS WinCC Open Architecture (OA) [2]), a process control layer (based on Schneider and SIEMENS PLCs) and standard middleware protocols. Although both layers are built following the same approach, only the supervision layer is addressed in this paper. The supervision layer provides the gas expert central team and Experiment end-users with a homogeneous look & feel and standard control for the monitoring and operation of their gas systems. Each of these supervision instances are based on the UNICOS [3] and JCOP [4] frameworks and are composed of several user interfaces, means for navigation between views and trending plots, which are all generated automatically.

The next sections describe the methodology behind the adopted model driven approach, the limitations that the current generation tool reached with time and the challenges of the design of a new tool to achieve the same functionalities and beyond.

HOW THE MODEL DRIVEN APPROACH WORKS

All layers of the gas control systems were designed with a modelling approach and use standard and homogenous building blocks. The architecture of the LHC's gas systems is modular. Every element that can be inserted into the gas control system is previously modelled. Models, templates and generation tools are created to build the systems.

A gas control system is hierarchically organized, as shown in Figure 1. A plant is always made of gas systems (i.e. sub-detector of an Experiment) which are in turn made of gas modules like Mixers, Pumps, Purifiers, etc. .

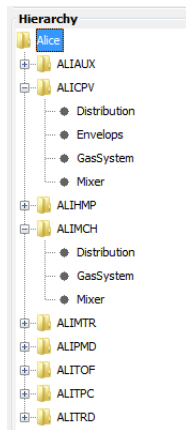

Figure 1 ALICE gas systems hierarchy.

A gas module is composed, in turn, of graphical objects or devices, which may or may not be present in the gas system. All gas control applications are organized following this model. The specificity (such as the required gas modules, and the optional elements) of each gas control system is then captured in so-called variable files, as shown in Figures 2 and 3, and described in the next section.

The gas systems are described with system templates. These templates specify the architecture of the plants, the modules used, the building blocks used and their configuration. The templates can refer to variables and use their values to configure the resulting generated gas system, as will be described in the next sections.

Specifying the Diversity, the Variables

This paragraph will introduce the concept of variables, and how these variables can define the architecture of the systems and the modules used.

All gas systems (sub-detectors) have a variable file, which specifies all the modules a plant is composed of. The

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA163

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

next figures show a part of the variables of the ATLAS experiment in a graphical representation.

Variable	Description	ATLAUX	ATLCSC	ATLMDT	ATLRPC	ATLTFC	ATLTGC	ATLTRT
subdetector	System Name	AUX	CSC	MDT	RPC	TFC	TGC	TRT
has_Mixer	standard module optional (option valid? 1=yes, 0=no)	0	1	1	1	0	1	1
has_Pump	standard module optional (option valid? 1=yes, 0=no)	0	0	1	1	0	1	1
has_Distribution	standard module optional (option valid? 1=yes, 0=no)	0	1	1	1	0	1	1
	standard module							

Figure 2 Variables of the ATLAS experiment.

The ATLAS experiment has seven sub-detectors. In Figure 2, they are displayed as seven columns in the table. Each row represents the variables specified within the files. The "has_" lines specify if a module is part of the system or not. All installed modules within a sub-detector, have in turn, another variable file. In this example, the gas systems of five sub-detectors contain a mixer module. Each of these five sub-detectors has a variable file for their mixer module with the values of the variables specifying the composition of the mixer itself, as shown in Figure 3.

Variable	Description	ATLRPC	ATLTRT	ATLCSC	ATLTGC	ATLMDT
wago_node_nb	Number of wago nodes	1	1	1	1	1
double_mfc	is double MFC present? (1=Yes 0=No)	1	1	0	0	1
gas_lines	Number of Mixer gas lines	3	3	3	2	3
Liquid_line	Number of liquid line	0	0	0	2	3

Figure 3 Variable values for a Mixer module.

The XML Format

The tools, like the HMI generator, used to generate and maintain the plants use an XML file format to describe the variables.

The root of the variable file is a *ns2:variable* tag. It can contain multiple description tags with the name, type, value and comment information. One description tag describes one variable. An example of such a file is shown below:

```
<ns2:variable>
   <description>
       <name>has_Mixer</name>
        <type>Integer</type>
        <value>1</value>
        <comment>standard module optional
(option valid? 1=yes, 0=no)</comment>
    </description>
</ns2:variable>
```

THE HMI GENERATOR TOOL

The HMI of each gas control system instance consists of panels (views), trending pages and explorer trees for navigation between views and trending pages.

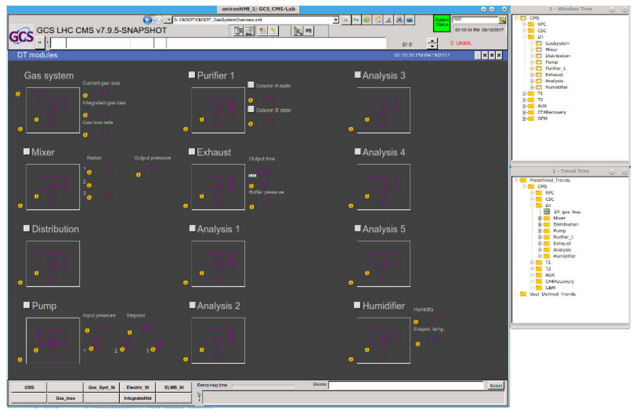

Figure 4: Generated Panel, Window and Trend trees.

The HMI of the 30 LHC gas control instances are generated by an application that was developed at CERN in early 2005. This generator processes the variables, system and panel templates to generate the output files. The generated application is defined by the system template files, which contain the generation rules. The generator uses these rules and previously created XML panel templates to generate the desired HMIs. In the **Functionality of the tool** section these rules will be described.

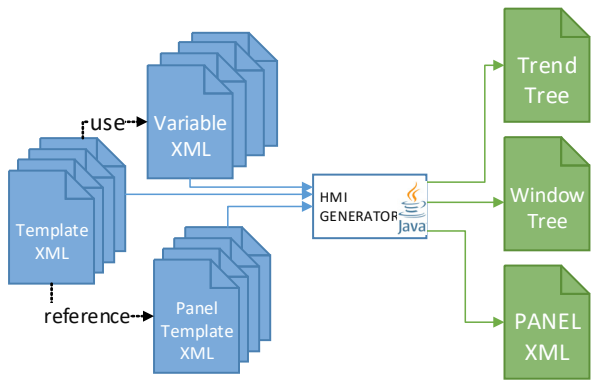

Figure 5: Inputs and outputs of the generator.

Today, almost 400 input files (208 variable files, 33 system templates and 146 WinCC OA template) are used to develop and maintain the HMIs of the 30 LHC gas control systems. These files include system templates with generation rules, variables to define the different system specificities and pre-created panel templates with control scripts.

For instance, the HMI of a medium-size gas control system of an experiment is composed of a total of ~500 files, whereas large systems, like ATLAS, are composed of more than 700 files.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA163

PROJECT GOALS

During more than ten years of usage, the HMI generator became more and more complex. Although the extensions done to the generators serve the current needs well, its current implementation became an obstacle for the integration of some very specific projects into the model driven approach and could not be covered by the tool. Moreover, some generator implementations were too function and view specific with hardcoded elements and variables.

For these reasons, it was necessary to re-engineer the HMI generator tool. The focus was to eliminate these barriers in order to have a more generic tool that could be used for other WinCC OA control based applications with a similar device hierarchy.

The project aimed at re-engineering the generator tool while keeping compatibility with the generation templates and avoiding application specific developments. Given the large number of existing templates, compatibility was of prior importance. Even small modifications to the templates would require many working hours.

TECHNICAL SPECIFICATION

The new tool is written in Java 1.8 whereas the previous application was implemented in C#. The reason for the change was to achieve platform independency and to rationalize the number of programming languages used by the Industrial Controls and Safety (ICS) group.

To make the generation process traceable, the new tool uses SLF4J (Simple Logging Facade for Java) [5] with the Log4J2 [6] logging framework with different logger managers for the generators. In this way, the logging levels are easily adjustable depending on the needs while the solution is independent of the logging framework, such that in the future, it would be easily replaceable. The output logs can be written to a file or to the console, depending on the log messages and the configuration of the framework. The detailed log messages can help to debug errors during the generation process and assist developers to troubleshoot the templates or variable files.

Another angle of the re-engineering was the possibility to test new releases of the tool easily. The tool is tested using JUnit [7] tests. In case of code changes, the tests can highlight the impact on the generation process. If the output of the modified generator is different from the expected one, the tests will fail.

FUNCTIONALITY OF THE TOOL

Besides the operational panels of the gas control system, the HMI generator is also used to create the corresponding trends and arrange them in a navigation tree, as well as the so-called window tree that allows navigation among the different operational views.

To ensure the independency of the tool on the different control domains, all application specific parts of the code were moved to the template files; both the panel and system templates. This required to introduce new rules and to modify the panel templates. The new generator implementation processes the information received from the template and injects it into the prepared user interface templates. The tool itself has no knowledge about the resulting plant and it makes no assumptions.

For example, Figure 6 shows a panel generation template where the coordinates of the elements are calculated by the rules and not hardcoded in the tool like in the precedent version. Wherever this solution was not possible, the panel templates were modified to add control scripts that dynamically attach the graphical elements to the view.

WinCC OA user interfaces are defined in XML format. This format is readable and modifiable using the generation tool. The trend tree and window tree outputs are exported to text files in the custom format of the UNICOS WinCC OA Tree component. These tree files can be imported into the gas control application without any further transformation or modification.

For large gas systems where the number of plots to be created is large, the import of the trend tree into the gas control application can take a long time; in some cases, more than an hour. Hence for upgrade purposes it was decided to implement a new functionality to handle partial trend tree generation. The partially generated trend tree can then be imported into a given position on the application's trend tree.

Concept of the Tool

This paragraph introduces the basic concept of the generation process, i.e. how the processing of the input elements happens to create the output files.

The generation process contains three major steps. The first one is a preprocessing step where all input files are interpreted. In the second stage, the HMI and files used by other tools are created. In the final step, the trends, and trend and window tree files are generated in the format of the UNICOS framework.

The generation process requires two input files. One is a system template file, which describes the generation rules, the other is an instance file, which contains the main system variables.

The template file can contain "*generate*" tags. In this case, a new generation process is started. The template structure can have infinite generation levels and a tree-like structure. Each branch of the tree is processed in turn. At each level of the branch the processing can also read one new variable file. The newly read variables are available to all sub-tree below but its scope is limited to this sub-tree, i.e. does not cause variables with the same name used in upper levels to be overwritten.

To create a complex system without unnecessary duplication of tags, the template can contain "*for*" loops, "*if*" expressions, and allows defining local "*variable*"-s. The generator processes these kinds of instructions first. The generation itself happens on the resulting template, which only contains the element generation rules (user interfaces, windows tree, trend tree, trends) and IO operations (merge, delete file).

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA163

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Example of a Panel Template

Figure 6 shows an example of a panel generation template. It shows a complex rule structure used to generate the user interface illustrated in Figure 4. The following subsections describe the generation rules with the help of this example.

```
<!-- generate subdetector panel overview-->
<panel>
  <template>SubdetectorOverviewTemplate.xml</template>
  <output>{panels_dir}/{prefix}_GasSystemOverview.xml</output>
  <variable type="integer" name="xPos">20</variable>
  <variable type="integer" name="yPos">40</variable>
  <rules>
    <set_property item="PanelHeader"
     prop="FileName">gcsSynopticHeader.pnl</set_property>
    <set_dollar ref="PanelHeader"
     name="$sTitle">"{subdetector} modules"</set_dollar>
    <rename_dollar_param param_name=".*"
     match="GCSPrefix" replace="{prefix}"/>

    <for variable="c_module" in="modules">
      <if variable="has_{c_module}" value="true">
        <if variable="c_module" value="Purifier">
          <for variable="c_purifier" in="1:{Purifier_count}">
            <add_modul position="{xPos},{yPos}"
             panel="{prefix}_{c_module}{c_purifier}Summary.xml"/>
            <variable name="yPos">{yPos}+195</variable>
            <if expr="yPos > 810">
              <variable name="xPos">{xPos}+420</variable>
              <variable name="yPos">40</variable>
            </if>
          </for>
        </if>
        <if expr="c_module != Analysis and c_module != Purifier">
          <add_modul position="{xPos},{yPos}"
           panel="{prefix}\\{prefix}_{c_module}Summary.xml"/>
          <variable name="yPos">{yPos}+195</variable>
          <if expr="yPos > 810">
            <variable name="xPos">{xPos}+420</variable>
            <variable name="yPos">40</variable>
          </if>
        </if>
      </if>
    </for>
  </rules>
</panel>
```

Figure 6: Panel generation in rules.

Rule Abilities

The strength of the generator tool lays in the flexibility of the template rules. The new tool can process complex logical expressions. Figure 6 shows how the generation rules look like in the template. This example demonstrates that two kinds of *if* expression can be used. A basic implementation, which is based on a value check, and a complex expression interpreter, like

```
<if expr="(!(!test3 == !Y ) == Y ) or
(test2 == N) and test1 == ICALEPCS">.
```

The tool can process complex boolean expressions. The "*variable operator value*" type expressions is processed according to the variable. For example, on strings only the equality (==) and inequality (!=) operators can be used, while on integers a wide range of operators (<, >, <=, >=, ==, !=) are possible.

Operators, such as 'and', 'or' and '!' can be used in expressions. During the evaluation of the expression, the operator precedence is mathematically handled.

It is possible to create subexpressions with brackets, which are evaluated and their result is substituted into the parent expression. The nesting level of the expressions is not limited, i.e. a subexpression can be inserted into any other expression.

Variable Substitution

To generate the HMIs it is necessary to inject variables into the generation templates. The HMI generator has two sources of variables. The variable files described above and variable tags within the templates. The variables are accessed by their names. The variable names inserted into curly brackets, shown in Figure 6, are replaced in the template with the value of the variable. It is also possible to use multiple embedded variable substitutions in the templates.

Variable Expressions

It is possible to use expressions as value of a variable. The next example shows the assignment of a new value to a variable.

```
<variable name="xPos"> {xPos}+420 </variable>
```

In the example above, the value of the variable is not a number, but an expression (the current value of the variable itself increased by 420). The tool interprets all the basic numeric operations and keeps the operator precedence in variable expressions too.

Implementation

After the preprocessing step, the *for* and *if* expressions are executed. The remaining elements are then processed. The generator reads through the resulting XML DOM. Every child of the *generate* tag is processed by a new generator instance. The tool reads the tag, gets the registered generator to handle the generation and hands the element over to the generator.

The tool is designed to be easily extendible. All generators implement the same *EntityGenerator* abstract class and overload the generate function. This abstract class implements every step to process a tag except the generation process itself, which must be overwritten in the specific implementation, i.e. in the derived class.

The multiple instances of the generator used to process a file, are loaded into a map. This map stores the constructor of the generators as a functional interface to the name of the element to be generated.

PERFORMANCE IN APPLICATION UPGRADE AND CREATION

The model driven approach is currently used to produce the supervision layer of the four main Experiments of the LHC. Over the years, many gas systems have been upgraded (i.e. new gas modules have been added to existing gas systems), new gas systems have been integrated into Experiment plants and new versions of WinCC OA have been introduced.

These change requests are easily managed with the model driven solution presented here. A modification of a

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA163

template or a variable file is a onetime process, and the time required to re-generate all HMIs of a large experiment is half a minute.

CONCLUSION

The experience confirmed that the effort required for the maintenance of this model driven design and generation system is much lower than a manual approach. The modification and update processes are easier and faster, and less error prone than systems maintained by human intervention. The possibility of human error is low, due to the minimal amount of interaction required and to the easily reproducible output results.

The maintenance and support of the new tool take less effort, thanks to the commonly used technologies and high level of Java knowledge of the ICS group.

Beyond that, the new tool is Experiment independent, as it does not contain experiment or any user interface specific code. Its flexibility allows to integrate and create any WinCC OA based application following a model based approach.

In the context of the gas control system; other plants are planned to be integrated into the model-generated approach, and the generator tool is now ready to support this process.

REFERENCES

[1] G. Thomas et al., "LHC GCS: A Model-driven approach for automatic plc and scada code generation," *ICALEPCS,* 2005.

[2] "WinCC OA," [Online]. Available: http://www.etm.at/index_e.asp [Accessed 19 09 2017].

[3] "UNICOS FW," CERN, [Online]. Available: http://unicos.web.cern.ch [Accessed 19 09 2017].

[4] "The Joint Control Project," [Online]. Available: http://itco.web.cern.ch/itco/Projects-Services/JCOP/. [Accessed 19 09 2017].

[5] "SLF4J," [Online]. Available: https://www.slf4j.org [Accessed 19 09 2017].

[6] "Log4j2," Appache, [Online]. Available: https://logging.apache.org/log4j/2.x/. [Accessed 19 09 2017].

[7] "Junit," [Online]. Available: http://junit.org [Accessed 19 09 2017].

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.