LHC GCS: A MODEL-DRIVEN APPROACH FOR AUTOMATIC PLC AND SCADA CODE GENERATION

G. Thomas[1], K. Azarov[1], R. Barillère[1], S. Cabaret[2], N. Kulman[1], X.Pons[1], J. Rochez[1],
[1]*CERN, Geneva, Switzerland,* [2]*UPJV / LMBE-ESIEE, Amiens, France – CERN, Geneva, Switzerland.*

ABSTRACT

The LHC experiments' Gas Control System (LHC GCS) project [1] aims to provide the four LHC experiments (ALICE, ATLAS, CMS and LHCb) with control for their 23 gas systems. To ease the production and maintenance of 23 control systems, a model-driven approach has been adopted to generate automatically the code for the Programmable Logic Controllers (PLCs) and for the Supervision Control And Data Acquisition (SCADA) systems.

The first milestones of the project have been achieved. The LHC GCS framework [4] and the generation tools have been produced. A first control application has actually been generated and is in production, and a second is in preparation.

This paper describes the principle and the architecture of the model-driven solution. It will in particular detail how the model-driven solution fits with the LHC GCS framework and with the UNICOS [5] data-driven tools.

INTRODUCTION

The development and maintenance of the LHC experiments' control systems require non-negligible resources and effort. The Joint COntrol Project (JCOP) [2] has been set-up to find common solutions for the four LHC experiments. In the context of JCOP, LHC GCS has been initiated to provide control applications for all the LHC experiments' gas systems.

The LHC experiments need gas mixtures for 23 of their sub-detectors. A single CERN group, PH-DT1-Gas, has been mandated to provide the experiments with gas systems. This group has designed a modular architecture in order to build and to maintain homogeneous gas systems. Although these gas systems are very similar, they are not identical. They are all built from the same set of functional modules but the number and type of the actually fitted functional modules differ from one system to another. In addition, some devices of the functional modules are optional.

The LHC GCS instances are turn-key control applications covering the supervision and the process control layers. A model-driven solution has been developed to be able to produce and maintain these 23 control applications with limited manpower resources.

STRATEGY

As described in [3] the nature of the process led to the selection of industrial tools and principles for the implementation of all layers of the LHC GCS instances: A SCADA system – ETM's PVSS - for the supervision layer, Schneider PLCs for the process control, standard protocols for the middleware and field buses for the access to the devices.

In order to ease the production of homogeneous control systems it was decided to produce first a set of components, the LHC GCS framework [4], which would be the common basis for the development of all LHC GCS instances. UNICOS [5] was identified as a solution which could cover most of the needs of the LHC GCS framework. UNICOS offers libraries and data-driven tools to build consistent PLC and SCADA layers. A collaboration has been set-up with the UNICOS developers to share the development and maintenance of some of the UNICOS components.

The UNICOS tools provide developers with skeletons of the SCADA and PLC applications which have then to be completed manually. Completing 23 skeletons would have been error prone and painful. In addition, the first steps of the project demonstrated that the validity of the options as well as generic user requirements could be modified in a later phase of the project. It has therefore been decided to go a step further and to add a model-driven suite of tools above the LHC GCS framework to automate completely the code production.

PRINCIPLES AND ARCHITECTURE

The heart of the approach is a gas system model which has been produced thanks to the collaboration with their designers. The model-driven approach consists of tools (Meta-Model generator, SCADA generators) which produce from the meta-model and objects databases the inputs required by the LHC GCS SCADA and PLC components.

As displayed on Figure 1, tools have been developed to edit the meta-model database. The meta-model generator extracts from this model all the pieces of information required by the LHC GCS framework and generators. The LHC GCS framework [4] consists of libraries and highly data-driven components (SCADA and PLC components) which allow the production of the different layers of the LHC GCS instances. As inputs, they require either populated databases or files.

The LHC GCS generators and databases have been designed in such a way that they remain compatible with the UNICOS ones.

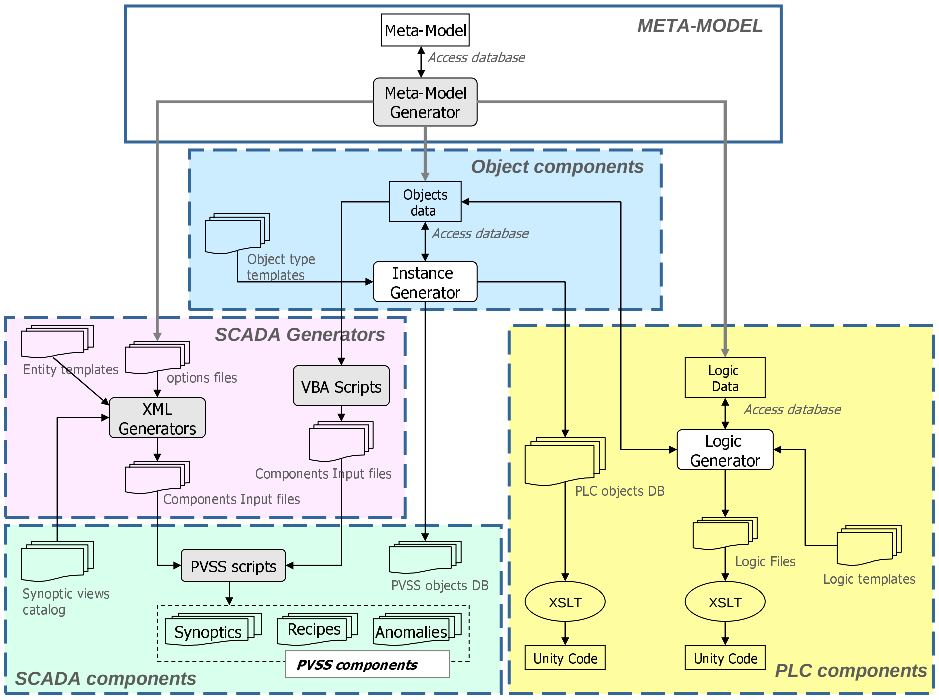

Figure 1: Meta-model data-flow

META MODEL

The meta-model is the core component of the model-driven approach. The schema of the meta-model database has been developed by interacting with the gas system designers. The goal was to capture all hardware architecture details and variations of the 23 gas systems as well as differences in the control requirements. The challenge was then to guarantee that any further modifications of the gas system architectures would remain compatible with the schema.

The meta-model identifies all the entities that exist within a gas plant and their associations. They are hierarchically organized (see Figure 2): A plant is made of gas systems; a gas system is made of gas modules (mixer, pump, purifier, etc.). The module's attributes define which optional devices are fitted or which optional control loop are required.

The meta-model is implemented with a Microsoft Access database. The referential integrity of the database schema is implemented in such a way that any updates done at the level of the plant are propagated to all its systems (if one removes a plant, its associated systems are also deleted).

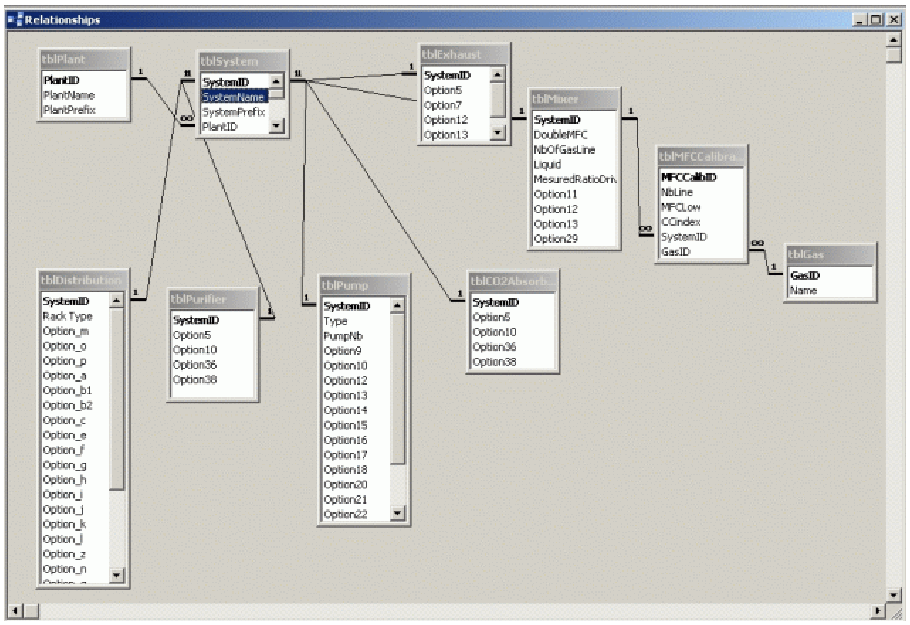

Figure 2: LHC GCS Meta-Model schema.

Application developers are provided with scripts and forms to edit the database. The following functionality is supported:

- Add/Remove: adding (removing) a plant/system/module to (from) the meta-model.
- Edit: updating the properties of a plant/system/module and options.
- Publish reports: producing reports of the meta-data content in two different layouts: one lists the systems and their characteristics for each plant; the other one prints the list of systems for each module type.

These features have been developed using the Microsoft Access scripting facilities to ease their integration with the UNICOS tools.

MODEL-DRIVEN OBJECTS GENERATION

The instance generator is a UNICOS data-driven tool [5] which is the key element in the code generation of UNICOS-based applications. It actually produces the interface between the PVSS and PLC layers. From the exhaustive objects list of the application and template files (one defined per object type), the tool generates: the object's code (which is the code that implements the behavior of the object) in PLC and PVSS and the middleware timestamp files (to allow communication between the two layers).

In the UNICOS approach, the list of required objects is produced manually by process experts and stored in dedicated Excel spreadsheets. They also provide the values for these objects' properties in the Excel columns.

The added value of the model-driven approach is the automatic creation of these Excel files from the meta-model. The process control designers have defined the required object's inventory for each gas module type taking into account all optional devices.

VBA scripts query the options in the meta-model and populate the objects database according to the options and rules defined by the process control designers. The scripts also include, for each object record, default values for its fields. Some additional facilities to interact with the object database have been added to the generator. For any gas systems (gas modules) the application developers can delete the object's database and/or export the object's database into Excel spreadsheets.

SCADA APPLICATION GENERATORS

The purpose of the SCADA generators is to prepare the necessary inputs for the SCADA components of a given LHC GCS instance. SCADA components include: synoptic and trends views, window and trend trees, anomaly views, alert summary, recipes and access control. Two different approaches have been used for the generation of these input files (see Figure 3). A simple one generates the required files directly from the object's database. A more complex method combines the information of the meta-model with generation rules to produce the input files.

The simple method is used to generate the inputs for the recipes, alert summary, anomaly views and access control components. For instance, to produce the recipe component input files, VBA scripts query parameters and values directly from the objects database and build files which contain for each required recipe the recipe type name and the list of elements of the recipe type. The input files are written in a predefined CSV format. As described in the LHC GCS framework [4] these files are then processed by the PVSS scripts of the GCS recipe component to create the required data points in the PVSS database.

The more complex method is used to build the inputs for the remaining SCADA components (synoptic and trend views, window and trend trees). Instead of directly producing the component input files from the objects database, this method combines information extracted from the meta-model with production rules to produce the required component input files (see Figure 3). The information extracted from the metal-model is basically the list of options which are applied to a given LHC GCS instance. The rules describe for each type of gas modules, and for each component, the entities to be created according to the valid option.

The complex method is for instance used for the generation of the PVSS synoptic views. In addition to the options and rules files, the generator needs, for this component only, another input: the 'synoptic views catalog'. The catalog is composed of synoptic view templates which have been developed using the PVSS interactive graphical editor and exported in XML format. The catalog contains for each module at least one synoptic template representing all its possible hardware architectures and showing all its optional devices.

The list of options ('option instance file' in Figure 3) is built by VBA scripts which analyze the meta-model and return the valid options (fitted devices) of a given gas system. The scripts produce one file per gas system and one file per module fitted in that gas system. They are coded with a very simple XML format which consists of an <instance> root element, which in turn contains an arbitrary number of variable definitions and option assignment elements. The rules are described in XML file format ('entity template' in Figure 3) and the processing of these files is done by a generator (XML generator in Figure 3) written in C# language.

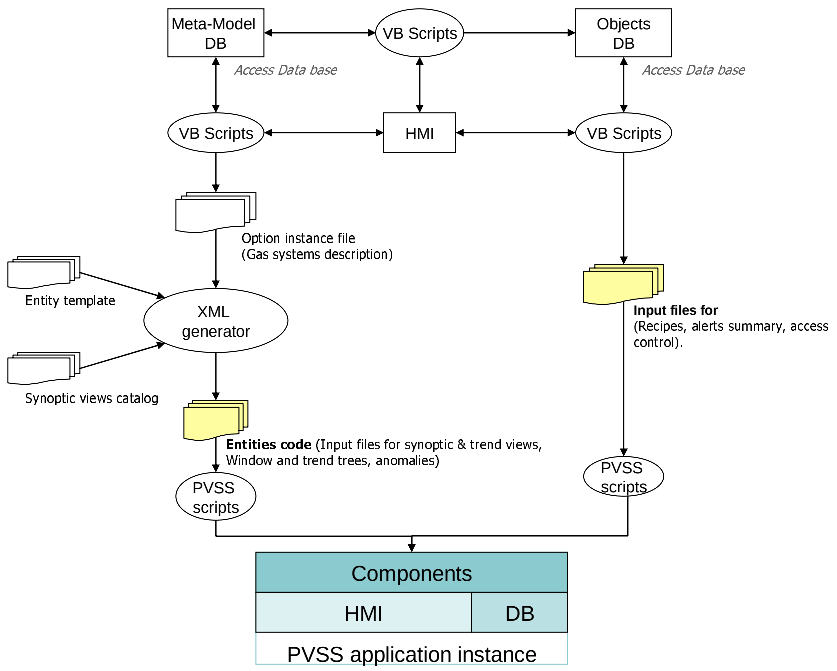
Meta-Model DB
VB Scripts
Objects DB
Access Data base
Access Data base
VB Scripts
HMI
VB Scripts
Option instance file (Gas systems description)
Entity template
XML generator
Synoptic views catalog
Input files for (Recipes, alerts summary, access control).
Entities code (Input files for synoptic & trend views, Window and trend trees, anomalies)
PVSS scripts
PVSS scripts
Components
HMI
DB
PVSS application instance

Figure 3: SCADA application generators data flow.

The file is divided in sections marked by XML tags. The root element of this file is <template>, it has two children namely <options> and <generation> (where the order is significant). The children of <options> contain the declaration of the possible gas system module options defined in the model. Each child of the <generation> section is dedicated to a given component (XML tag: <component_1>). It contains general information for the generation (e.g. name of output file) of this component and the component specific generation rules (XML tag: <rules>). These rules are actually production rules for the component.

```
<generation>
        <component_1>
        …
        <rules>
        …
        </rules>
        </component_1>
        ….
        <component_n>
        …
        </component_n>
</generation>
```

The XML generator reads the entity template file and creates all variables declared in the <options> section. Then it reads the option file and assigns all values to the variables. Following this, it parses the <generation> section using the depth-first method and pre-processes any directive rules such as ‘if’ and ‘for’ expressions. Finally it processes the component rules and produces the corresponding entities code.

For instance to generate the synoptic view files, there is a generation rule section defined by an XML tag called <panel>. It has 3 children: <template>, <output> and <rules> which respectively specify the synoptic template file name of the synoptic catalog, the output file name to be created and a section describing the processing rules.

The <rules> tag can contain several production rules (such as “remove_options”, “rename_dollar_param”, “set_property”, etc.). For example the “remove_options” removes graphical items (e.g. an optional device in a template synoptic) to produce the final synoptic view.

```
<panel>
        <rules>
         <remove_options options=”Option5, Option7, Option12, Option13” pattern=”*{{}}.*”/>
        </rules>
</panel>
```

For each significant option listed in the ‘options’ attribute, which has a ‘false’ value, the generator substitutes ‘{}’ in the ‘pattern’ attribute by the option name. The generator then removes all items of the synoptic template whose name matches the produced pattern.

The production rules are evaluated by a tool written in C# language using .Net framework 1.1. Therefore additional entity generators may be written in any language which can be compiled to a .Net framework assembly.

The XML generator functionality can easily be extended to deal with new entities. For instance if a new component is introduced, one will have to define according to the XML generator guidelines, new XML tags for the component, new production rules and code to handle these new tags and rules in the C# tool.

MODEL-DRIVEN LOGIC CODE GENERATION

The logic generator is an additional UNICOS tool which generates the skeleton of the application PLC code. It is tightly linked to the UNICOS instance generator as the code it generates is organized according to the objects declared in the objects database. As described in [4], developers have to associate each field object (e.g. valve, compressor, heater, etc.) with a piece of code driving this object and each process object (PCO) with 7 pieces of code. The LHC GCS framework provides developers with routines which implement the final application code instead of the skeleton.

The tool takes as inputs the objects database and logic code template files (described in LHC GCS framework [4]). It produces as outputs a list of pre-compiled PLC code files. In the UNICOS approach, the developers associate manually each object with the appropriate include files and specify a list of parameters as properties of this association. These relationships are kept in an additional database, the logic database.

The model-driven approach consists of filling and configuring automatically the logic database fields. The process control designers have specified for each object the required include file (if the object is instantiated) and the required parameter values according to the options of a given LHC GCS instance. The logic database is populated by VBA scripts which query the options in the meta-model and fill all the fields according to the process control designers' specifications. As for the objects generation, the application developer can delete the database and export the objects to Excel spreadsheets.

CONCLUSION

A fruitful collaboration with the UNICOS team allowed the development of a complete suite of tools to generate the full code for the PLC and the SCADA layers of the LHC GCS instances. These tools have been used to produce the first of the 23 LHC GCS instances. The second is in production while the gas system hardware is being installed.

A model-driven approach has been possible thanks to the modular architecture of the gas systems and to the efforts of its designers to keep their detailed design generic. This approach reduces the development effort and enables the generation and commission of all 23 gas systems in a short time. Thus it allowed the LHC GCS team to delay the production of the LHC GCS instance code and to put the initial efforts on the development of the generation tools. This flexibility was particularly appropriate for the gas systems' scheduling. Indeed the hardware architecture of some of the gas systems is to be finalized in a later phase of the project. These changes can easily be handled without any additional development provided the hardware architecture remains compatible with the model.

REFERENCES

[1] The LHC GCS project, URL: http://itcofe.web.cern.ch/itcofe/Projects/LHC-GCS/

[2] The Joint Control Project, URL: http://itco.web.cern.ch/itco/Projects-Services/JCOP/

[3] R. Barillère et al., "LHC GCS: A homogeneous approach for the control of the LHC experiment gas systems", ICALEPCS'2003, Gyeongiu, Korea, October 2003.

[4] G. Thomas et al., "LHC GCS: A framework for the production of 23 homogeneous control systems", ICALEPCS'2005, Geneva, Switzerland, October 2005. [PO1.042-6]

[5] P. Gayet et al., "UNICOS a framework to built industry-like control systems, Principles and methodology", ICALEPCS'2005, Geneva, Switzerland, October 2005. [WE2.2-6I]