SOFTWARE FACTORY TECHNIQUES APPLIED TO PROCESS CONTROL AT CERN

Mathias Dutour, CERN, Geneva, Switzerland

Abstract

The CERN Large Hadron Collider (LHC) requires constant monitoring and control of quantities of parameters to guarantee operational conditions. For this purpose, a methodology called UNICOS (UNIfied Industrial COntrols Systems) has been implemented to standardize the design of process control applications. To further accelerate the development of these applications, we migrated our existing UNICOS tooling suite toward a software factory in charge of assembling project, domain and technical information seamlessly into deployable PLC (Programmable logic Controller) – SCADA (Supervisory Control And Data Acquisition) systems.

This software factory delivers consistently high quality by reducing human error and repetitive tasks, and adapts to user specifications in a cost-efficient way. Hence, this production tool is designed to encapsulate and hide the PLC and SCADA target platforms, enabling the experts to focus on the business model rather than specific syntaxes and grammars. Based on industry standard software, this production tool together with the UNICOS methodology [1] provides a modular environment meant to support each process control expert to develop his solutions quickly.

This article presents the user requirements of the software factory and the chosen approach. Then the focus moves to the benefits of the selected architecture, and ends up with the results and a vision for further improvements.

INTRODUCTION

The large scale process control applications developed at CERN cannot be presented as a production line, but rather as custom-designed applications in constant evolution during their lifecycles. This evolution during the development phase is dictated either by extensions to the system or by applying corrections and implies the need for often regenerating the process control applications.

Therefore the maintenance on the long term of the tools to produce such process control applications is critical.

With these ideas in mind, a software factory [2], also called the UAB (UNICOS Application Builder) tool, was designed to enable faster and cheaper code generation in a context of often changing requirements.

DRIVING REQUIREMENTS

Besides the limitations of the existing code generation tools [3], the users expect more support and efficiency from the UAB, to focus only on their field of expertise and to be able to reuse the assets they produced across teams and projects.

Extensibility

Process control applications are subject to many extensions. Typical examples: a new type of physical device has to be integrated or additional parameters are required for process control logic.

The UAB tool reflects this versatility and provides the means to integrate seamlessly these new elements. Additionally, the assets produced by the team for this integration work, are valuable and directly reusable in a different context for another team, thus increasing productivity.

Since the domain knowledge involved in the development of a process control application is not platform specific (e.g.: PLC vendor independent), the UAB tool provides the required support to guarantee the reusability of the business assets produced as well. On the same level, it also means the UAB tool can be easily extended to address new platforms without starting from scratch again and again.

Separation of Concerns

The inputs and outputs of the UAB tool are handled by several people with different expertise and responsibilities.

Typically the description of the project data, i.e. the field-level description of the process control application, is realized by a System developer, while its business logic is the responsibility of a Domain expert. Finally the PLC or the SCADA developer is responsible for the integration and the deployment of the generated application.

For the UAB tool it is therefore important to decouple these different aspects and keep them separate from the UAB tool internals.

Consistency Checking Support

The consistency checking support offered by the UAB tool is implemented at different levels:

First an unambiguous means to feed the information into the code generation process is provided, enforced through the use of predefined models.

Second, the UAB tool provides the users with powerful means to validate the semantic consistency of the code generation process inputs. The objective here is to detect and fix issues as early as possible during the development phase.

Finally, the UAB tool reports to the users any problems identified during the code generation and provides automatically hints for resolution.

ARCHITECTURE

More than a simple tool, the UAB tool is rather an approach to deal with automatic code generation.

The central idea of the UAB architecture is to decouple the low-level information of the project (rather descriptive), from the usage of this information (the domain knowledge), from the project instantiation itself (the platform-specific generated code).

The various packets implementing this approach together with the stakeholders are presented below:

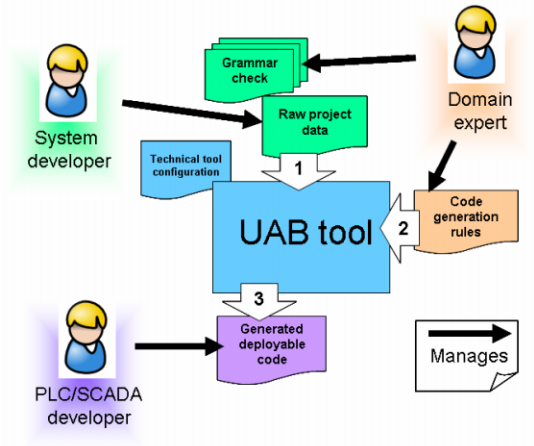

Figure 1: UAB tool context.

Raw Project Data and Grammar Check Packets

The Raw project data typically contains the information describing the process control system itself, and is therefore target-platform dependent (i.e. it describes platform-specific PLC and SCADA information). However, it doesn't describe how this information shall be used for the code generation process.

This project data is likely to be updated on a regular basis during the project development phase, as the user requirements are received and integrated.

The chosen format to gather this information is XML (eXtended Markup Language), the 'de facto' industry standard and vendor independent data–encapsulation language. XML allows this data to be constrained by an XML schema, presented here as the 'Grammar check" packet in the Figure 1. Unlike the Raw Project data files, the Grammar check packet contains only structural definition and is designed as an extensible asset to be shared across process control projects.

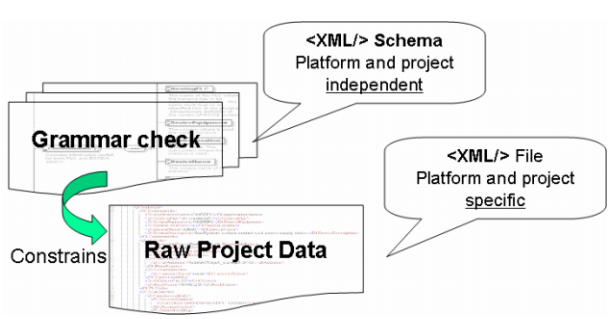

Figure 2: Project information.

The Grammar check packet structure is known by the Code generation rules packet and used by the UAB Tool packet to generate an internal representation of the Raw project data. This internal representation is managed by the JAXB [4][5](Java Architecture for XML Binding) library. Thanks to JAXB, the extension or structural modification of the Raw project data can be realized on the fly with no need to modify the UAB Tool and immediate availability to the Code generation rules.

Code Generation Rules Packet

Just like the orchestra conductor, the Code generation rules don't contain any data, but simply encapsulate the business knowledge of the output expected. Their primary goal is to drive the code generation through a set of abstract services. (E.g.: same rules apply whatever the PLC target platform is)

By focusing on the "What" rather than on the "How", the Domain expert in charge of the Code generation rules can focus only on the system behaviour he expects. The Code generation rules have been designed to enable platform syntax abstraction, a step away from error prone syntaxes.

To achieve their goal, the Code generation rules have at their disposal two handles. A first handle on the Raw project data to extract any relevant information, and a second one on the code generation services of the UAB Tool to dictate what to do with this information:

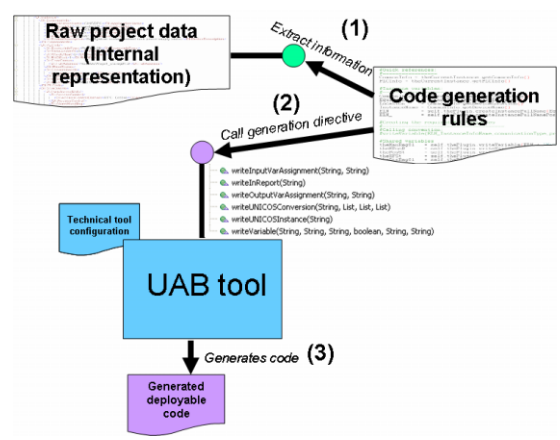

Figure 3: Code generation principle.

The flow of operations is the following: The Code generation rules accesses the UNICOS Project data (Step 1), possibly verifying or pre-processing this data, then calls abstracts services of the UAB tool (Step 2), which in turn generates the related pieces of code with the proper syntax (Step 3).

Concretely, the Code generation rules consist of a set of files written in a scripting language. The Jython [6] (Python for Java) language was chosen for this purpose, as it integrates perfectly with the UAB Tool and provides extensive functionality. Using such a scripting language rather than a flat properties file allows very powerful constructions. It allows the Domain expert, first to perform consistency and semantic checks on the Raw

project data, and second to request code generation services possibly adapting dynamically to the current context.

UAB Tool Packet

The UAB tool main packet is actually a container for platform–specific code generation plug–ins. To minimize maintenance, the UAB Core itself follows the broker design pattern and provides the plug–ins with an extensive set of high level interfaces (see Figure 4). The UAB Core is also in charge of other traditional aspects as well, such as graphical user interface, command line handling, file management, online error logging, etc...:

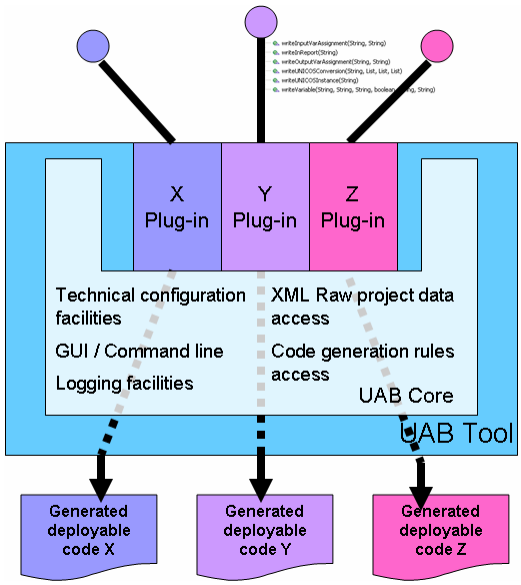

Figure 4: UAB tool packet internals.

To achieve the objective of extensibility, the UAB Core is highly configurable and has no static knowledge of its plug-ins, the content of the Code generation rules, or even any UNICOS concepts.

The chosen language for the development of the UAB Core and its plug-ins is Java. Java permits high coding productivity and abstraction mechanisms such as introspection and runtime class loading, which are used efficiently by the UAB Core to dynamically adapt to its environment.

UAB Tool Plug-ins

The plug-ins managed by the UAB Core are independent from each other and have different responsibilities. They are only focussed on pure code generation aspects; and know how to transform the abstract requests of the Code generation rules into proper vendor-specific source code. For example, one of our plug-in is responsible for the Schneider UNICOS PLC code generation and simply knows how to instantiate PLC objects and map them in PLC memory. For the rest the plug-in relies on the UAB Core mechanisms.

All plug-ins are built onto the same model. Each one can interact with the Code generation rules, access the Raw project data, and use UAB Core interfaces. Having a reusable model makes it easy to develop and integrate new plug-ins, even with little programming experience.

BENEFITS

The software factory approach, implemented here in the context of process control, allows to focus on the expected result rather than on the means to produce this result. Mixing static configuration, auto-adaptive software and abstract user directives, the UAB tool is a powerful and yet simple rule–driven code generation environment.

The project technical data, business logic and tooling configuration are clearly separated preventing the spaghetti plate effect: The long term maintenance of the process control applications is made safer and cheaper.

The multi-level error checking mechanisms addressing grammar, syntax and semantic aspects filter-out many mistakes which could be difficult to detect before deployment and therefore very costly to track down and fix.

Nonetheless, this approach is not self sufficient and does enforce on the onward a rigorous design of the project constructions to be used, such as the Grammar check and Code generation rules packets. This is also to the direct benefit of the quality of the process control application produced.

CONCLUSION

At this date the UAB Core is being finalized while various UNICOS plug-ins are well advanced, namely for the PLC objects code generation on the Schneider Unity© platform and their supervision counterpart on the PVSS SCADA. Siemens Step7© plug-ins are under development as well for objects and control logic code generation.

However, the UAB is not limited to UNICOS or even code generation, and its architecture can adapt to many domains with a need for a flexible offline data processing solution.

REFERENCES

[1] Philippe Gayet and Renaud Barillere, “UNICOS A framework to build Industry like control systems: Principles and methodology”, CERN, Geneva, Switzerland.

[2] Jack Greenfield and Keith Short, “Moving to Software Factories”, Microsoft© Corporation.

[3] G. Thomas, “LHC GCS: A model-driven approach for automatic PLC and SCADA code generation”, CERN, Geneva, Switzerland.

[4] The GlassFish community, http://java.sun.com/javaee/community/glassfish/

[5] Joseph Fialli and Sekhar Vajjhala, Sun Microsystems© Inc. “The Java™ Architecture for XML Binding (JAXB)”, January 8th 2003.

[6] The Jython Project, http://www.jython.org.