# SOFTWARE FACTORY TECHNIQUES APPLIED TO PROCESS CONTROL AT CERN

**Mathias Dutour** (Mathias.Philippe.Dutour@cern.ch)

**CERN Information Technology / Controls, Geneva, Switzerland**

**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 errors 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 provide a modular environment meant to support process control experts to develop their solutions quickly.**

## Software for automatic code generation ?

**Automatic? Sure, but…**

The automatic code generation process aims indeed at more productivity, improved time to market and increased final product quality. However, the maintenance of the code generation process shall not become a nightmare due to changing user requirements and extensions to the process control application at end. Based on this observation, we identified the following needs to be addressed:

- Assets reusability across teams and projects.
- Enhanced control over code generation.
- Robustness toward input structural changes.
- More user support for troubleshooting.
- Distinguish domain from technical knowledge.
- Support of multiple platforms at once.

## The proposed solution: The UAB Tool.

**…ok, no problem.**

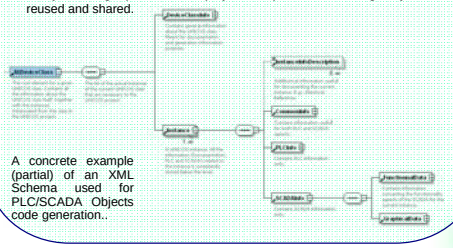
To meet the user needs, the UAB (UNICOS Application Builder) is implementing the following mechanisms:

- Platform-independent models and data structures
- More power to the users to drive the code generation process through domain knowledge scripting.
- Automatic adaptation to user inputs structural modifications.
- Automatic grammar and syntax checking + support for semantic consistency verification.
- Technical, domain and project specific knowledge are handled and defined separately.
- Target platform abstraction through high level PLC/SCADA code generation services.

## Grammar check

The Grammar check packet consists of one or several XML Schema files designed to be easy to extend and to maintain. These XML Schemas have the following responsibilities:

- They define the XML properties which describe every piece of information of the Raw project data XML file(s).
- They used to validate the XML input, to prevent errors in the typing of the Raw project data.
- They are target-platform and project independent, meaning they can be reused and shared.



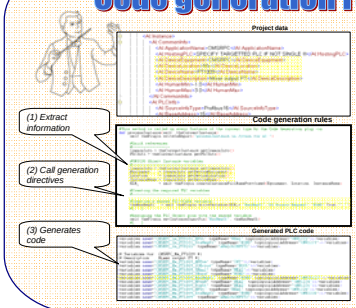
A concrete example (partial) of an XML Schema used for PLC/SCADA Objects code generation..

## Raw project data

The Raw project data consists of a simple monolithic XML file validated by the Grammar check XML schemas. The information gathered here is to be used directly or interpreted by the Code generation rules and the UAB Plug-ins. One could identify 3 distinct categories of information described here:

- Documentation: This information is simply present to document other contained information, to help understanding the purpose of the information provided.
- Meta data: This information characterizes other pieces of data contained in the XML file for proper processing during the code generation process.
- Technical: Finally the low level technical information such as process control object characteristics.

## Code generation rules




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 offered by the code generation plug-ins.

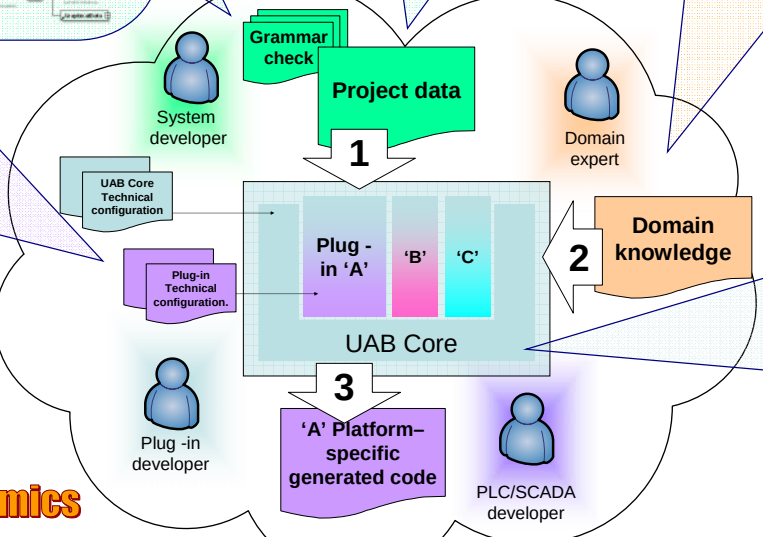
## Plug-ins


"Plug-ins act just as code weavers."

Each Plug-in is only in charge of the formatting of the generated output code and repetitive tasks.

According their purpose and usage, they offer several platform-independent code generation services as illustrated below. A plug-in neither has the direct knowledge of Raw project data nor contains hard-coded information.

Finally the information to tune their behavior is described in their respective Plug-in technical configuration.




## UAB Core

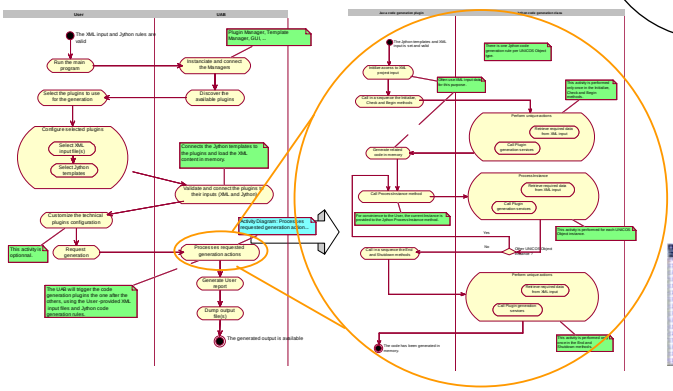
The UAB Core packet contains the central tool management mechanics.

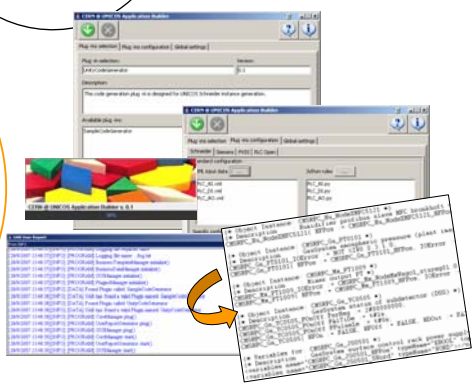
It has a pure technical role and has no interest or knowledge in Raw project data, Code generation rules and targeted platforms, it acts simply as a transparent information broker.

One could summarize its responsibilities as follows:

- It discovers dynamically the various Plug-ins and assert their validity;
- It loads the Raw project data information as an internal representation (Multiple XML inputs is allowed per Plug-in);
- It provides the Plug-ins with their respective UNICOS Project data information and Code generation rules.
- It provides common services required by most of the plug-ins (E.g.: User report, file I/O,…).

## UAB Tool dynamics





## Technologies


The chosen technology for every input data to the UAB Core and recommended for the code generation plug-ins is XML, validated by XML Schemas.


Java is programming language for the UAB Core and the plug-ins development to rely on free, industry-standard and vendor – independent technologies to develop the UAB Tool.

JAXB (the Java Architecture for XML Binding) is the chosen technology to access the XML information. JAXB is a SUN©-supported open initiative to provide both automatic XML-to-Java binding and XML mangling/de-mangling. It was selected by the UAB Tool mainly because it allows runtime XML Schema modification and adaptation.


Finally the Jython scripting language (Python for Java) provides power and simplicity to the Code generation rules.

**SUMMARY**

**The software factory approach, implemented here in the context of process control, allows focusing 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 de-correlated 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 forehand 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.**

**Finally the UAB tool 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.**