MODEL ORIENTED APPLICATION GENERATION FOR INDUSTRIAL CONTROL SYSTEMS

B. Copy, R. Barillere, E. Blanco, B. Fernandez Adiego, R. Nogueira Fernandes, I. Prieto Barreiro
CERN, Geneva, Switzerland

Abstract

The CERN Unified Industrial Control Systems framework (UNICOS) is a software generation methodology and a collection of development tools that standardizes the design of industrial control applications [1]. A Software Factory, named the UNICOS Application Builder (UAB) [2], was introduced to ease extensibility and maintenance of the framework, introducing a stable meta-model, a set of platform-independent models and platform-specific configurations against which code generation plugins and configuration generation plugins can be written. Such plugins currently target PLC programming environments (Schneider and SIEMENS PLCs) as well as SIEMENS WinCC Open Architecture SCADA (previously known as ETM PVSS) but are being expanded to cover more and more aspects of process control systems. We present what constitutes the UNICOS meta-model and the models in use, how these models can be used to capture knowledge about industrial control systems and how this knowledge can be leveraged to generate both code and configuration for a variety of target usages.

INTRODUCTION

Models are an abstraction of real-world phenomena that represent them for the purpose of problem solving. Model usage is gaining wide acceptance in the domain of industrial control software engineering as it allows to produce implementations that are closer to human understanding of a system than low-level coding.

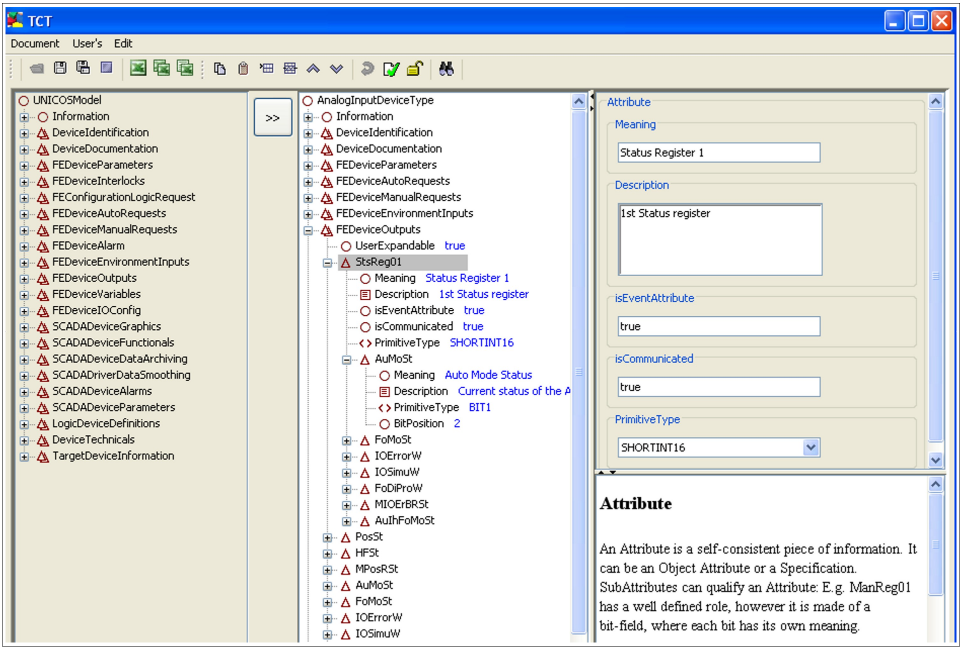
A meta-model is the “model of a model”, describing the properties of the model in a way that makes it applicable to wider range of related problem domains. For instance, a programming language grammar can be considered a meta-model of the programming language it describes. Such a grammar can be used to automatically determine whether programming statements are indeed valid and can therefore result into functional problem resolutions.

Another example of meta-models can be an XML Schema Description (XSD) – which is a meta-model for XML documents, themselves acting as models for the data they contain.

THE UNICOS META-MODEL

The UNICOS framework [1], designed to deal with heterogeneous COTS equipment, provides equipment abstractions of one of the following categories :

- I/O Devices – interfaces, which act as data transfer objects.
- Field Devices (pumps, valves etc..) – which act as domain objects.
- Process Control Object (PCO) Devices – which act as controllers, coordinating I/O and field devices.



```
UNICOSModel
  Information
  DeviceIdentification
  DeviceDocumentation
  FEDeviceParameters
  FEDeviceInterlocks
  FEConfigurationLogicRequest
  FEDeviceAutoRequests
  FEDeviceManualRequests
  FEDeviceAlarm
  FEDeviceEnvironmentInputs
  FEDeviceOutputs
  FEDeviceVariables
  FEDeviceIOConfig
  SCADADeviceGraphics
  SCADADeviceFunctionals
  SCADADeviceDataArchiving
  SCADADriverDataSmoothing
  SCADADeviceAlarms
  SCADADeviceParameters
  LogicDeviceDefinitions
  DeviceTechnicals
  TargetDeviceInformation
```

```
AnalogInputDeviceType
  Information
  DeviceIdentification
  DeviceDocumentation
  FEDeviceParameters
  FEDeviceAutoRequests
  FEDeviceManualRequests
  FEDeviceEnvironmentInputs
  FEDeviceOutputs
    UserExpandable   true
    StsReg01
      Meaning   Status Register 1
      Description   1st Status register
      isEventAttribute   true
      isCommunicated   true
      PrimitiveType   SHORTINT16
      AuMoSt
        Meaning   Auto Mode Status
        Description   Current status of the A
        PrimitiveType   BIT1
        BitPosition   2
      FoMoSt
      IOErrorW
      IOSimuW
      FoDiProW
      MIOErBRSt
      AuIhFoMoSt
    PosSt
    HFSt
    MPosRSt
    AuMoSt
    FoMoSt
    IOErrorW
    IOSimuW
```

Attribute
Meaning: Status Register 1
Description: 1st Status register
isEventAttribute: true
isCommunicated: true
PrimitiveType: SHORTINT16

Attribute

An Attribute is a self-consistent piece of information. It can be an Object Attribute or a Specification. SubAttributes can qualify an Attribute: E.g. ManReg01 has a well defined role, however it is made of a bit-field, where each bit has its own meaning.

Figure 1 : Screenshot of the FESA General Editor for UNICOS Device Type creation

Devices (in effect, object instances) that compose a UNICOS control system all therefore correspond to a Device Type (equivalent to an object class).

The UNICOS meta-model describes the properties and constraints that could be used to specify device types, thereby allowing to provide assistance and formal validation during the specification and implementation of device types.

An XSD based meta-model

The UNICOS meta-model, being expressed in standard XSD, leverages off-the-shelf validation and binding technologies. Being a meta-model for XML documents, it also guarantees that UNICOS device types and device instances (both expressed as XML data) be exploited by any XML aware technology.

In particular, the reliance on XSD to express its meta-model allows UAB to reuse the FESA General Editor [4], a lightweight and general purpose graphical XML editor that provides input completion and validation using an XSD document. The FESA General Editor is used in particular to produce device types that comply with the UNICOS meta-model.

Fig.1 presents a screen capture of the FESA General Editor (also referred to as the "UAB Type Creation Tool" in the UAB context). The UNICOS meta-model is featured on the left hand-side column, where type creators can pick and mix Attributes belonging to different Attribute Families in order to compose their type. The middle column is a user-friendly XML document editor that allows to prepare a UNICOS device type. The right hand-side column provides a meta-data-based property sheet, complete with data validation, inline documentation hints and data entry assistance (such as support for drop-down lists of supported values).

Characteristics of the UNICOS meta-model

The UNICOS meta-model supports the definition of Device Types, *i.e.* a generalization of devices such as the ones found at CERN in industrial continuous processes.

Device types exhibit Attributes, which are the definition of device type properties. *For example*, attributes typically have properties such as :

- **Description**: used to document an attribute. , its purpose.
- **PrimitiveType** : specifies the property's data type (FLOAT, TIME, STRING, etc.).
- **isCommunicated**: defines whether the property is remotely accessible (from the SCADA layer for instance).
- **isSpecificationAttribute**: this property is used when the attribute value can be modified by the end-user.

To take an analogy with traditional object-oriented programming, Device Types can be thought of as an Object Class, while Attributes are the class' field members.

For better readability, Attributes of a Device Type are grouped in attribute families, such as *for example* :

- **FEDeviceOutputs**, which identifies a front-end device's outgoing signals.
- **SCADADeviceDataArchiving**, which specifies that SCADA-relevant configuration parameters to log data held in device properties to persistent storage.
- **TargetDeviceInformation**, which holds target platform-specific information (*e.g.* information that varies whether we are targetting a SIEMENS or a Schneider PLC device).

Once a Device Type is defined through the UNICOS meta-model (with the usage of the FESA General Editor [4]), and its peer SCADA and device level implementations are made available, the device type is ready to be handed over to UAB application experts, who will establish lists of devices present in their applications.

If we take back our OO programming analogy, application experts simply provide object instances composing their application, and benefit immediately from all the validation and tooling that the UNICOS meta-model grants them usage of.

MODELLING USAGES IN UAB

First and foremost, the UAB model's most prominent application is the UNICOS-CPC framework [5]. UNICOS-CPC provides a library of device types and associated code generation templates suitable for the implementation of continuous control processes, relying on PLC automata and the SIEMENS WinCC Open Architecture SCADA software package.

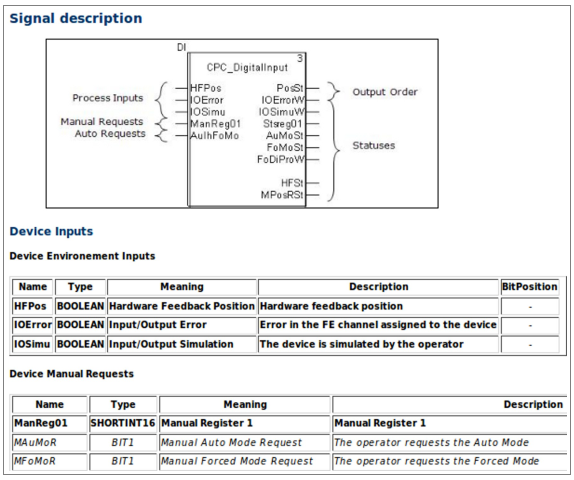
Building upon the UNICOS meta-model, a significant number of tools have been added to the UAB tool suite, in order to improve the quality of generated applications or better integrate UAB generated applications with the rest of the CERN infrastructure. These tools take full advantage of the standards compliance of the UNICOS meta-model but provide a stark example of how models can be applied to a variety of contexts and be leveraged with diverse levels of software expertise.

Automated device type documentation

UNICOS Device Types, especially those defined in the UNICOS-CPC device types library typically exhibit a large number of properties that grant the UNICOS framework its support for fine-grained operation (any single property of a device instance can for instance be overridden, insuring that process experts can take over in order to deal correctly with unexpected or abnormal situations without experiencing hindrance).

It is therefore paramount to be able to automatically document device types in a human readable format, in order to assist control system developers in preparing device instances. Since the UAB model relies on XML, it can be transformed by means of XML Stylesheets (XSL), an XML dialect specialized in performing structured template based transformations. Usage of XSL against UAB XML device type definitions makes generating web-based documentation a natural fit.

Fig. 2 below presents an example of a manual Digital Input description diagram, mixed with auto-generated documentation of the device environment inputs


Signal description

Device Inputs

Device Environement Inputs

Name	Type	Meaning	Description	BitPosition
HFPos	BOOLEAN	Hardware Feedback Position	Hardware feedback position	-
IOError	BOOLEAN	Input/Output Error	Error in the FE channel assigned to the device	-
IOSimu	BOOLEAN	Input/Output Simulation	The device is simulated by the operator	-

Device Manual Requests

Name	Type	Meaning	Description
ManReg01	SHORTINT16	Manual Register 1	Manual Register 1
MAuMoR	*BIT1*	*Manual Auto Mode Request*	*The operator requests the Auto Mode*
MFoMoR	*BIT1*	*Manual Forced Mode Request*	*The operator requests the Forced Mode*

Figure 2 : Screenshot of UNICOS-CPC manual

Integrating the UNICOS-CPC device type definitions with a content management system such as Atlassian Confluence and its support for inline XSL transformations provides a very simple way to assemble documentation mixing formal technical definitions (such as a list of device pins and their usage) with informal textual descriptions or comments. Other target documentation formats could be DocBook or OASIS DITA, both XML dialects suitable for authoring (and therefore potential targets for XSL transformations), providing results suitable for professional publishing.

Semantic verification for system specifications

While the choice of XML schema as a meta-model provides UAB with a large choice of supporting technologies, XSD (intentionally and inherently) lacks the flexibility and richness of other meta-models, such as the OMG Meta Object Facility (MOF) [6]. As a result, certain constraints cannot be easily expressed in XML-friendly form.

Furthermore, UNICOS device types are meant to be preparable by individuals unfamiliar with programming languages or constraint languages such as OCL [6], but still require the possibility to check in details that control system device instance lists comply with formal rules.

To avoid introducing too much complexity in the UNICOS meta-model, the Jython scripting language was chosen to perform these kind of validations, increasing the flexibility of the semantic rules mechanism and allowing the creation of platform specific rules. Most of these rules are applied over the specifications file (this file contains all the devices of a UNICOS application and its parameters).

Examples of existing rules are :

- Check the maximum length of a device's platform dependent name alias.
- Check the relevance of all inter-device relationships referenced in the specifications file.
- Verify that platform dependent input data does not contain illegal characters.

Such rules cannot be easily specified in XML schema.

Extended configuration generation

Another example of the application of the UNICOS meta-model lies with the support for extended configurations. UAB generated applications typically need to be integrated with the rest of CERN's infrastructure, such as :

- the LHC Alarm Service (LASER), to inform CCC operators of abnormal conditions across the entire accelerator and technical infrastructure
- other LHC devices communicating through protocols such as CMW (CERN Common Middleware) or DIP (CERN distributed information protocol)

Such integration relies on simple, well-known integration bridges that do not need any advanced code generation, but rather well-known and structure configuration parameters. Using code generation templates for such use cases would therefore introduce unwanted complexity and unnecessarily increase the UAB codebase.

For the purpose of the generation of such configurations, UAB employs a Python-based transformation tool called FlexExtractor [8].

FlexExtractor employs a pluggable strategy to obtain input data (for instance, from XML or a binary Excel file) and produce output data (for instance CSV, or binary Excel file), providing a language independent transformation processor in the middle.

Such a tool is ideal for non-programmers, who benefit from complex transformations without the burden of a programming environment and limitations of XSLT, and allows a decoupling that considerably eases maintenance of configuration generators.

EVOLUTION PERSPECTIVES

While we have seen that an XML Schema-based meta-model and XML data formats open the door to a variety of supporting technologies and target environments, its lack of expressiveness and flexibility also imposes unwanted limitations. Such limitations are the main reason for the creation of tools such as the Semantic Verifier mentioned ealier in this document, delivering the type of validation that XML based ones cannot support at all.

Another important point is that disconnected XML files typically lead to fragmentation of the knowledge and encourage the usage of these XML files in isolated processes.

Since the advent of the UAB project, new technologies have emerged, delivering the same affinities with XML while offering better data management capabilities and superior tooling.

Content Repositories

Content Repositories (also referred to as Java Content Repositories) are an emerging type of object-oriented databases that provide services such as :

- meta-data support – which, as we have seen, is the keystone of any generation or validation process.
- Navigation and query – either based on relational syntax (*i.e.* SQL), XML oriented (*i.e.* Xpath), Object oriented or even free text indexing (*e.g.* fuzzy search engines).
- Serialization support – allowing to import or export tree fragments so that they can be exploited or transformed offline – for instance to an XML representation, or to an in-memory object graph.
- Transaction and versioning support.
- Eventing support – allowing a decoupling between data providers and data consumers.

Some of these services, such as meta-data support, navigation and serialization are already present in the UAB architecture – others such as query, transaction and eventing support are essential in order to let the UNICOS meta-model scale up (for instance, to deal with redundant or hierarchical SCADA architectures, or to handle inter-PLC communication).

Xtext and the Meta Object Facility (MOF)

MOF [6] is a standard for model-driven engineering introduced by the Object Management Group (OMG). It was primarily used to act as a much-needed meta-model for the Unified Modeling Language (UML). MOF can be compared in some respect to XML Schema, but offers much higher flexibility, in its status of a closed meta-model, in effect, a model that can represent itself.

The Eclipse Foundation provides a generation tooling called “Xtext” which accepts models, defined using a closed-meta-model subset of MOF called ECore, but also more traditional inputs such as grammar normal forms or XML schema definitions, and can produce automatically tooling supporting :

- First off, the essential facilities enumerated earlier - meta-data support, navigation, queries, serialization, events.
- Advanced model data entry assistance such as autocompletion, on-the-fly validation and property sheets support, thereby replacing and enhancing the FESA General Editor (written manually) with auto-generated editors.
- Meta-programming APIs that can be used to manipulate data and call methods against the model programmatically at the meta-model level.

CONCLUSION

The UAB technology stack has already demonstrated the importance of a meta-model foundation in its support of the development of code generation tools.

Our experience with tooling development also showed that, even with a meta-model, flexibility and scalability are essential to ensure that the models can evolve and avoid knowledge fragmentation (for instance, rules defined by the Semantic Rule Verifier UAB tool have no place in the UNICOS meta-model today, and the effort of adding them there today would outweigh the benefits).

REFERENCES

[1] R. Barillère, Ph. Gayet, *"UNICOS A Framework to build industry-like control systems, principles and methodology "*, ICALEPCS 2005, Geneva, Switzerland, WE2.2-6I

[2] M. Dutour, "*Software factory techniques applied to Process Control at CERN*", ICALEPCS 2007, Knoxville Tennessee, USA, TPPA03

[3] W3C XML Schema Working Group, “*XML Schema*”, 2000-2007, http://www.w3.org/XML/Schema

[4] M. Arruat et al., "*Front-End Software Architecture*", ICALEPCS 2007, Knoxville, USA

[5] B. Fernandez Adiego, E. Blanco, I. Prieto Barreiro, “*UNICOS CPC6: Automated Code Generation for Process Control Applications*”, ICALEPCS 2011, Grenoble, France, WEPKS033

[6] Object Management Group, “*Meta Object Facility*”, 1997-2011, http://www.omg.org/mof/

[7] Eclipse Foundation, “*Xtext*”, http://www.eclipse.org/Xtext/

[8] R. Nogueira Fernandes, “*FlexExtractor*”, 2010-2011, http://cern.ch/flexextractor