Model-Based Validation of Industrial Control Systems

E. Estévez and M. Marcos, *Member, IEEE*

***Abstract*—Current industrial applications demand the design of more and more complex, safe and trustworthy control systems exhibiting a high degree of flexibility and reutilization. To achieve this, the engineering process should be improved by making the engineering tools involved in the development process to collaborate during the design. This paper presents a model-based approach for designing complex automation applications. The core of the approach is constituted by a set of domain specific models that depend on the application field and whose elements, syntax and semantics are defined from the point of view of the experts that participate in the design of the system. The domain models are defined using engineering tools as the design progresses and they can be used to achieve tool integration through model collaboration. This can be achieved following the Model Driven Engineering approach by means of model transformations. This paper specifically focuses on the first step of this paradigm: the definition of domain languages, in this case for industrial control systems, as well as validation mechanisms of application designs coming from different domain tools. Three well known and widely used industrial standards have been used: Computer Aided Engineering eXchange (CAEX), PLCopen (a representation format for the IEC 61131-3 standard) and MathML (a language for defining mathematical constraints). Using model checking it is possible to assure the correctness of the control system specification and using model transformation it is possible to detect design errors in early stages of the design.**

***Index Terms*—AutomationML, CAEX, IEC 61131-3, Industrial control systems, MathML, model driven engineering, PLCopen.**

I. Introduction

INDUSTRIAL control systems are specific computer systems not only because of the special characteristics of the environment they deal with, industrial processes, but also because they use specific equipment, such as Programmable Logic Controllers (PLCs) as the primary control equipment, industrial communication systems, fieldbuses, if they are distributed, as well as specific purpose I/O devices. On the other hand, as a computer system, its design deals with the definition of the software architecture. Traditionally, control equipment vendors usually offered a development system based on proprietary software architectures and programming languages. In the last years, a great effort is being done by international organizations for promoting the use of standards in this application field. Two of the most remarkable and active ones are PLCOpen [1] and AutomationML [2]. PLCopen was founded in 1992 and it is a vendor- and product-independent worldwide association whose mission is to be the leading association resolving topics related to control programming to support the use of international standards in the field. The International Electrotechnical Commission (IEC) published the IEC 61131-3 standard [3] that proposes a software model and programming languages for Industrial Process Measurement and Control Systems (IPMCS). Most of the PLC manufacturers are aiming at becoming IEC 61131-3 standard compliant, offering powerful development tools to deal with the design of complex industrial control applications.

On the other hand, AutomationML allows engineers to describe the components of a real plant as objects in which different aspects are encapsulated. An object can consist of other sub-objects, and it can itself be part of a bigger composition.

Notwithstanding this, as fast as industry reaches a greater maturity level and applications become more complex, a consolidation of methodologies, which allow system description and definition before its construction, becomes more necessary. Different fields of expertise are involved in the design of distributed IPMCS; the design must deal with the strategies to meet the functional requirements as well as the nonfunctional requirements (security, timing, energy, etc.). It also deals with its implementation in the form of a software system as well as with the equipment needed to start up the final system. Every field of expertise defines the same control system from a different point of view using its own lexicon, syntax and semantics. The domain experts use domain specific tools to express the design of the system, e.g., Gant/PERT charts during analysis phase, MatlabTM for designing control strategies, programming tools, hardware configuration tools, testing tools... Usually, the main drawback is that the tools used in the different phases do not communicate among them. Thus, the user must perform manual transformations from one tool to another following an error prone process. But, given that at some extent each tool internally uses a domain model, the collaboration among the different tools involved in the development cycle could be achieved by means of the models they manage. To achieve this, a common representation format for the different models of the system is needed.

Therefore, modeling languages, that allow system description and definition before construction, could be used to automate the design. Model-Driven Development (MDD) [4] can be considered as an emerging paradigm which solves a number

Manuscript received March 30, 2011; revised June 15, 2011, September 18, 2011; accepted October 11, 2011. Date of publication November 01, 2011; date of current version April 11, 2012. This work was supported in part by the MCYT&FEDER under DPI- 2009-08102 and in part by UPV/EHU GIU 10/20. Paper no. TII-11-170.

E. Estévez is with the Department of Electronic and Automatic Engineering, University of Jaen, Jaen 23071, Spain (e-mail: eestevez@ujaen.es).

M. Marcos is with the Department of Automatic System Engineering, UPV/EHU, Bilbao 48013, Spain (e-mail: marga.marcos@ehu.es).

Digital Object Identifier 10.1109/TII.2011.2174248

of problems associated with the composition and integration of large-scale systems while, at the same time, allows incorporating the latest advances in software development.

Two model-based standards must be referenced as a success of the use of models for system design: Model Driven Architecture (MDA) proposed by the Object Management Group [5], and Model Integrated Computing (MIC) from ISIS (University of Vanderbilt) [6]. MDA distinguishes the Platform Independent Model (PIM) that describes the system from a high abstraction level, i.e., it defines what the system has to do in terms of functionality. Then, using model transformations the PIM is refined into Platform Specific Models which are the starting points for code generation. On the other hand, MIC approach uses Domain Specific Modeling Languages (DSML), as well as the application of model transformations for integrating analysis and other tools into a MDD process. The generic components of the MIC tool architecture are detailed in [7] and [8]. This work follows both approaches proposing domain models for distributed IPMCS using the MDA concepts.

On the other hand, standards of common representation formats have appeared within the manufacturing/automation field that allow having internal textual representations of models defined in engineering tools. These textual representation formats can be found in practically all disciplines from scientific (Matlab, simulink, MathML ...), manufacturing, automation or software engineering. In a sense, they constitute domain languages in which the models that domain tools manage internally are defined. The metalanguage in which most of them are expressed is eXtensible Markup Language (XML) metaschema [9]. In fact, AutomationML standard combines existing data formats that are already available and designed to store and exchange data of different aspects of engineering information. This standard uses Computer Aided Engineering eXchange (CAEX) [10] as a technical basement for the top level format (topology), COLLADA [11] as a file format to store geometry and kinematics information and PLCopen XML [1] to store logic information. Hence, AutomationML does not offer a new information store format; instead it manages three kinds of files that are compliant to the aforementioned standards. To do this, the CAEX compliant file is used as backbone and other files could be linked using resources offered by CAEX standard.

This work uses well spread standard formats to go a step further, using them as linguistic metalanguages in which ontological metamodels are defined. A linguistic metamodel defines the language in which models are expressed (structural aspects), while an ontological metamodel deals with the domain semantics, i.e., the internal meaning of models. Examples of linguistic metamodels are Meta Object Facility (MOF), the metamodel to which UML conforms to, or W3C metaschema, the metamodel to which XML conforms to. This work relies on the linguistic metamodel provided by existing standard Markup Languages (ML) and gives guidelines for defining domain modeling languages, i.e., the syntax and composition rules for a particular ontology. Thus, a methodology for defining domain specific modeling languages is presented and guidelines to implement model validation and transformation are given. In this sense, domain languages and model transformation rules can be used as the core of a framework integrating the set of Commercial Off the Shelf Tools (COTS) tools used through the development cycle of automation applications. Hence, the purpose of this work is not to substitute engineering tools. On the contrary, the ultimate goal is to make them collaborate by defining a common language, as well as transformation rules. The latter allow assuring that design specifications in different domain engineering tools are coherent. Once the design specification has been validated, and using again model transformations, the code skeleton for the target controllers could be automatically generated. The proposed approach is illustrated as applied to the design specification of a distributed industrial control system using the AutomationML editor.

The layout of the paper is as follows: Section II presents the related work that uses the concept of model in industrial manufacturing / automation. Section III describes the steps to define a domain language based on standard representation formats, while Section IV is dedicated to the automatic generation of a domain markup language from the domain language. In Section V, the methodology is applied to the definition of domain languages for distributed industrial control systems. Section VI illustrates the automatic generation of the corresponding ML and the methodology is applied to model a heat treatment line in Section VII. Finally, Section VII is dedicated to the concluding remarks and future work.

II. Related Work

Some attempts have been done by different authors towards the use of modeling concepts in the Industrial Process and Measurement Control Systems (IPMCS) field in order to support the design and code generation of the application, e.g., [12] present a model-based approach for mechatronic applications and [13] for PLC code generation. The Special Section on Information Technology in Automation [9] focuses on the development, adoption, and application of information technology in automation systems.

Other works extend the use of models to distributed IPMCS. References [14] and [15] describe a framework called CORFU (Common Object-oriented Real-time Framework for the Unified development of distributed IPMCS application) that defines an Industrial Process-Control Protocol for the development, distribution, and operation of IPMCS applications based on IEC 61499 Function Blocks [16]. The design is object oriented and uses UML to define a four-layered architecture. In [17], authors combine UML and SysML in order to model mechatronic systems using the four-layered architecture.

The Open, Object-Oriented kNowledge Economy for Intelligent inDustrial Automation (O3NEIDA) project [18] introduces the concept of "Automation Object" that encapsulates the intelligence of the components for every participant in the creation chain in industrial automation (vendors of industrial electronic devices, machine vendors, system integrators, industrial enterprises and vendors of software tools and services). The IEC 61499 FB is again the key encapsulation element for achieving the reuse of design solutions. Regarding to the IEC 61499 standard, a mathematical framework that captures the application and runtime execution semantics is presented in [19]. These semantic models are used in [20] to analyze and compare how an application would behave when executed in different platforms.

Other examples can be found in the embedded systems field, such as the MEDEIA project (Model Driven Embedded Systems Design Environment for Industrial Automation Sector). Its objective is to achieve an improvement in productivity for the development of embedded control systems within the European industrial automation sector. To do this, the Automation Component concept is defined. This latter is defined as a means of bridging the gap between the different users and their special ways of specifying and implementing any solution [21]. This common model stands between the specification and implementation elements of a plant automation system.

All of them are interesting approaches. Nevertheless, as far as authors know, none of them focuses on performing model validation or on how to achieve the integration of COTS tools commonly used in the field, such as simulation, temporal analysis or programming tools. Besides, they do not model specifically the hardware architecture, although this could be very useful to generate information for the hardware configuration phase. Finally, its use commonly implies a deep knowledge in object orientation technologies as well as keeping track of a continuous evolving discipline, as it is the software engineering field. Nevertheless, all of them are valuable works in the field that define the main concepts for describing the type of systems they deal with.

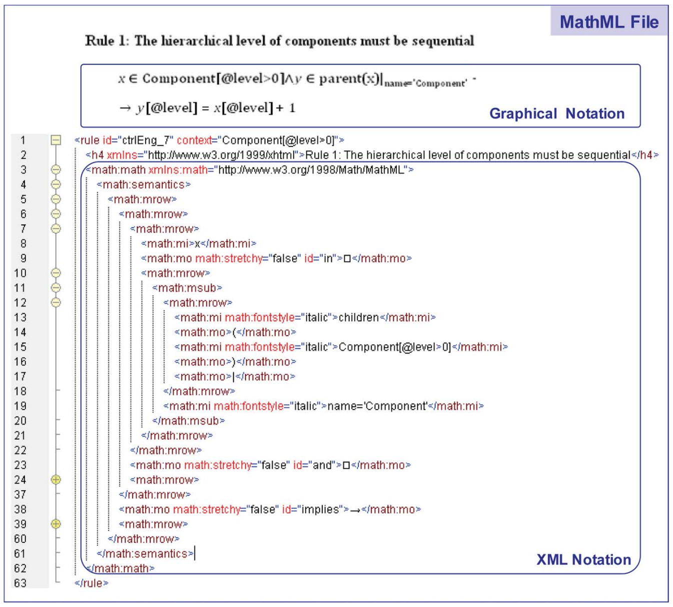
The work presented here is intended to cover the commented issues going a step further by defining a common format, based on standard representation formats from different fields. Jointly, they can be used not only for defining domain languages but also for performing model validation and transformation. In this sense, the same ideas could be applied to extend the aforementioned approaches. Guidelines for both, defining modeling languages and transforming models between different domains are provided.

III. Definition of Domain Languages Using Common Representation Formats

This section is centered on the definition of a common representation format for defining domain languages. The metamodel of a domain specifies the concepts the domain manages as well as the rules for combining the concepts to form models. There can also be a set of constraints or composition rules. In this work, two widespread standards have been selected (CAEX [10] and MathML [22]) that combined with standards of the application field can be used to define domain languages, i.e., ontological metamodels.

Domain concepts and the rules for combining them to form models (domain ontology) are specified using the CAEX standard using its flexible syntactic mechanisms for defining specific semantics and structure about objects. On the other hand, composition rules are implemented in MathML.

The metadata offered by CAEX follows the object oriented paradigm; it can be classified in two main groups: classes and instances. Three different library classes, SystemUnitClassLibrary, RoleClassLib and InterfaceClassLib, are used to represent definitions (i.e., Classes) and the element InstanceHierarchy represents the structure of a system being described in a CAEX file, i.e., a set of objects of the previous classes. Hence,



Rule 1: The hierarchical level of components must be sequential

$$x \in \mathrm{Component[@level>0]} \wedge y \in \mathrm{parent}(x)|_{\mathrm{name='Component'}} \rightarrow y\mathrm{[@level]} = x\mathrm{[@level]} + 1$$

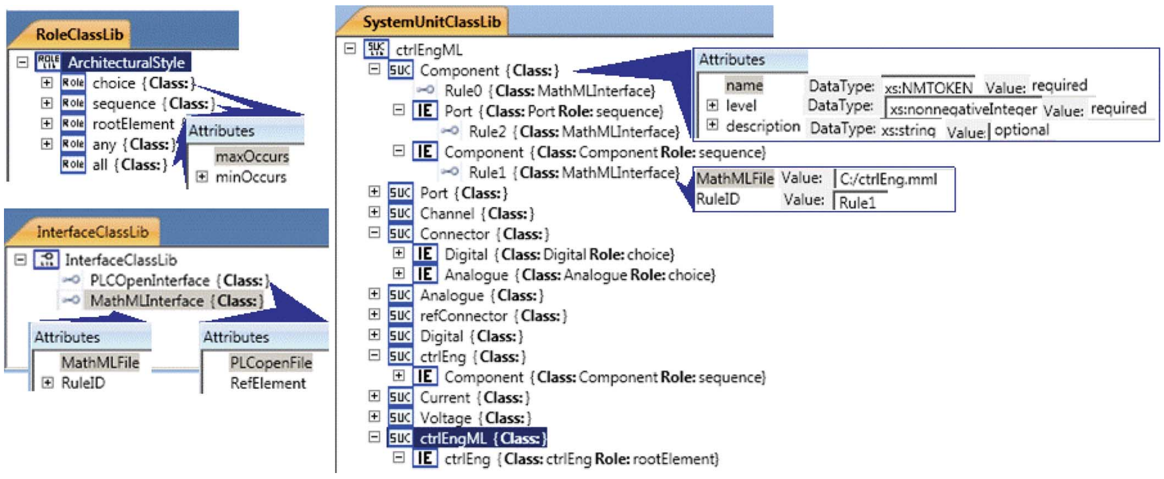
```
<rule id="ctrlEng_7" context="Component[@level>0]">
  <h4 xmlns="http://www.w3.org/1999/xhtml">Rule 1: The hierarchical level of components must be sequential</h4>
  <math:math xmlns:math="http://www.w3.org/1998/Math/MathML">
    <math:semantics>
      <math:mrow>
        <math:mrow>
          <math:mrow>
            <math:mi>x</math:mi>
            <math:mo math:stretchy="false" id="in">□</math:mo>
            <math:mrow>
              <math:msub>
                <math:mrow>
                  <math:mi math:fontstyle="italic">children</math:mi>
                  <math:mo>(</math:mo>
                  <math:mi math:fontstyle="italic">Component[@level>0]</math:mi>
                  <math:mo>)</math:mo>
                  <math:mo>|</math:mo>
                </math:mrow>
                <math:mi math:fontstyle="italic">name='Component'</math:mi>
              </math:msub>
            </math:mrow>
          </math:mrow>
          <math:mo math:stretchy="false" id="and">□</math:mo>
          <math:mrow>
        </math:mrow>
        <math:mo math:stretchy="false" id="implies">→</math:mo>
        <math:mrow>
      </math:mrow>
    </math:semantics>
  </math:math>
</rule>
```

Fig. 1. Example of a mathML rule.

the definition of the concepts and the relationship among concepts of a domain language are expressed as Library Classes and models in such domain as the InstanceHierarchy element of the corresponding CAEX file.

Using CAEX as a linguistic metamodel, the lexicon of a domain language is defined in a SystemUnitClassLibrary, where every model concept is defined as a System Unit Class (SUC) characterized by means of attributes. For instance, configuration, resource, task or Program Organization Unit are concepts of the domain language for defining the IEC 61131-3 software architecture. If complex elements are needed, for instance, if a model element can be composed by other model elements, CAEX offers the Internal Element (IE) concept that can be used to define complex SUCs. IEs are instances of other SUCs previously defined. The architecture of a model instance is defined by means of a RoleClassLib. This library contains as many roles as needed to define model compositions. Typical architecture roles are all, sequence, any and choice, and they can be used to combine concepts. All of them are characterized by the minOccurs and maxOccurs attributes for fixing the multiplicity of the elements. The architectural style of the language is defined in complex SUCs. Specifically, through enriching InternalElements with role classes.

In summary, the syntax of the language is implemented through the definition of a SUC library and a RoleClassLibrary. To complete the domain ontology, it is necessary to define the static semantics of the domain, i.e., the constraints that must be met in order to define a correct model. This is achieved by means of an InterfaceClassLibrary, used to link a SUC (or an InternalElement) to an external file. In this case, the external file contains the composition rule that the element must meet. The composition rule is expressed in MathML. This markup language allows expressing mathematical constraints in XML and it is supported by most of the mathematics editors. Fig. 1 illustrates a composition rule example in MathML. The top part of the figure illustrates the view provided by an equation editor



```
RoleClassLib
⊟ ArchitecturalStyle
   ⊞ choice {Class:}
   ⊞ sequence {Class:}
   ⊞ rootElement {Class:}
   ⊞ any {Class:}
     all {Class:}
Attributes
   maxOccurs
   ⊞ minOccurs

InterfaceClassLib
⊟ InterfaceClassLib
   PLCOpenInterface {Class:}
   MathMLInterface {Class:}
Attributes
   MathMLFile
   ⊞ RuleID
Attributes
   PLCopenFile
   RefElement

SystemUnitClassLib
⊟ ctrlEngML
   ⊟ Component {Class:}
        Rule0 {Class: MathMLInterface}
      ⊟ Port {Class: Port Role: sequence}
           Rule2 {Class: MathMLInterface}
      ⊟ Component {Class: Component Role: sequence}
           Rule1 {Class: MathMLInterface}
   ⊞ Port {Class:}
   ⊞ Channel {Class:}
   ⊟ Connector {Class:}
      ⊞ Digital {Class: Digital Role: choice}
      ⊞ Analogue {Class: Analogue Role: choice}
   ⊞ Analogue {Class:}
   ⊞ refConnector {Class:}
   ⊞ Digital {Class:}
   ⊟ ctrlEng {Class:}
      ⊞ Component {Class: Component Role: sequence}
   ⊞ Current {Class:}
   ⊞ Voltage {Class:}
   ⊟ ctrlEngML {Class:}
      ⊟ ctrlEng {Class: ctrlEng Role: rootElement}

Attributes
   name          DataType: xs:NMTOKEN           Value: required
 ⊞ level         DataType: xs:nonnegativeInteger Value: required
 ⊞ description   DataType: xs:string            Value: optional

MathMLFile  Value: C:/ctrlEng.mml
RuleID      Value: Rule1
```

Fig. 2. Domain languages for distributed IPMCS in CAEX.

(what the user sees) and the bottom part represents its XML expression (not visible to the user).

The InterfaceClassLibrary can also be used to associate a SUC defining a language concept to a file that contains a detailed specification of such element in a textual domain specification language. This second use of an InterfaceClass is illustrated in Section V, where the minimal programming unit within the IEC 61131-3 standard, the POU type, is linked to a file containing the corresponding code following the PLCopen XML schema. This XML schema has been defined by the technical committee 6 of PLCopen organization in order to achieve interoperability among all different kinds of programming environments [25]. The current version 2.0 covers most of the IEC 61131-3 2nd edition.

Fig. 2 illustrates the definition of a domain language using CAEX. As commented above, the SUC library defines the concepts of the language, the RoleClassLibrary defines the architectural style, and the InterfaceClassLib is used to associate composition rules (static semantics) in MathML to the elements of the language (SUCs and IEs).

IV. Extraction of a Domain Language From the Domain Modeling Language

In order to perform model validation, a domain specific markup language must be extracted from the CAEX and MathML files that represent the domain language.

In this work, a DSML is defined by a XML schema containing the abstract syntax of the language, the set of rules that restrict the abstract syntax (if these constraints cannot be performed by means of XML schema mechanisms) and the set of rules that express the static semantic of the domain. The latter can be expressed as a set of schematron rules.

In order to automatically obtain the DSML, it is necessary to establish the mapping rules from CAEX elements and MathML rules to XML Schema and Schematron rules. Three separate groups can be distinguished: 1) mapping rules related to the lexicon, 2) mapping rules related to the architectural style and 3) mapping rules related to the static semantics (composition rules). The first two correspond to transformations from CAEX to XML Schema. The latter involves transformations from MathML to schematron rules that will be finally embedded into the markup language.

As commented above, the lexicon is defined through a CAEX SUC library composed by a set of simple and/or complex SUCs containing at least one Internal Element. The characterization of these elements is done by means of attributes as Fig. 2 illustrates. These elements are mapped to W3C mechanisms; SUC libraries are mapped to XML element and to complexType and the characterization of a SUC is mapped to XML Attributes.

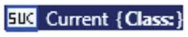
A RoleClassLibrary represents the architectural style of the language. It contains the choice, sequence, all and any roles characterized by two attributes for fixing the multiplicity: minOccurs and maxOccurs. They are used to enrich IEs belonging to a SUC definition and they are mapped directly to the XML Schema elements that define architectural styles (sequence, choice, all and any mechanisms and their multiplicity, minOccurs, and maxOccurs XML attributes). Finally, there is also a rootElement role.

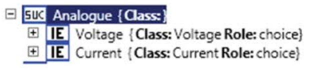
Table I summarizes the transformations to be applied to the language definition in CAEX in order to automatically obtain the domain ML. These transformations can be performed by means of a XML stylesheet (illustrated in Fig. 3 as CAEX2ML.xsl) [26]. A XML stylesheet template has been developed for every mapping rule.

V. 3 + 1 Domain Languages for Distributed Industrial Control Systems

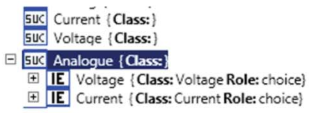
The design of distributed industrial control applications involves different domain disciplines, defining each of them a different view of the same system: the control engineering domain defines the functional design of the control system. It is commonly defined as a modular and hierarchical design where the modules (components) represent the control strategies connected through the actual data that the control system, the process and its environment exchange (field signals and internal connectors). These connectors may be grouped in channels in order to cope with designs involving a big number of process signals. Channels are linked to components via input and output ports. During the analysis phase of the development cycle, these modules can be related to both, the dynamic model

TABLE I
CAEX-DSML MAPPING RULES

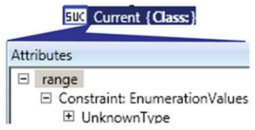
CAEX	*DSML using W3C XML schema resources*
LEXICON	
Simple SUC	Element definition
 SUC Current {Class:}	`<xs:element name="Current"/>`
Complex SUC	complexType definition

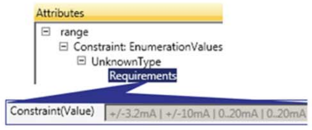
 SUC Analogue {Class:}
IE Voltage {Class: Voltage Role: choice}
IE Current {Class: Current Role: choice}
	`<xs:complexType name="Analogue">`
	  `<xs:choice>`
	    `<xs:element ref="Current"/>`
	    `<xs:element ref="Voltage"/>`
	  `</xs:choice>`
	`</xs:complexType>`

Internal Element (IE)	Reference to a previously defined element or Definition of an element using a complex type .

 SUC Current {Class:}
SUC Voltage {Class:}
SUC Analogue {Class:}
IE Voltage {Class: Voltage Role: choice}
IE Current {Class: Current Role: choice}
	`<xs:element name="Current">`
	`</xs:element>`
	`<xs:complexType name="Voltage">`
	`</xs:complexType>`
	
	`<xs:complexType name="Analogue">`
	  `<xs:choice>`
	    `<xs:element ref="Current"/>`
	    `<xs:element name="Voltage"`
	`type="Voltage"/>`
	  `</xs:choice>`
	`</xs:complexType>`

Attributes of SUC	Attributes of elements or complexTypes

 SUC Current {Class:}
Attributes
range
Constraint: EnumerationValues
UnknownType
	`</xs:simpleType>`
	`<xs:element name="Current">`
	  `<xs:complexType>`
	    `<xs:attribute name="range"`
	`type="ctrl:range"/>`
	  `</xs:complexType>`
	`</xs:element>`

Constraints
 Attributes
range
Constraint: EnumerationValues
UnknownType
Requirements
Constraint(Value) +/-3.2mA | +/-10mA | 0..20mA | 0..20mA
	Simple types
	
	`<xs:simpleType name="range">`
	`<xs:restriction base="string">`
	`<xs:enumeration value="0..20mA"/>`
	  ...
	`</xs:restriction>`

ARCHITECTURAL STYLE	

of the process (using a simulation tool such as Matlab™) they control and the corresponding control strategy (using for instance a PLC programming tool). In a sense, this view defines "what" the control system has to do in order to meet the functional requirements.

The second view of the application, the electronic-electrical engineering domain, deals with the selection of the equipment needed to startup the control system. The hardware architecture of industrial control systems uses specific hardware equipment, commonly consisting of a set of network nodes that correspond to industrial controllers (containing a set of resources: processing, memory, IO,... characterized by the manufacturer, serial number, firmware version, description,...), and to data acquisition devices (IO nodes, such as PROFIBUS_DP slaves characterized by its IO cards, node id, ...) connected through a set of industrial communication systems (buses). A node element is connected to a network segment through a communication board. Commonly this type of information is managed through a hardware configuration tool.

Finally, the software engineering domain defines the software architecture that implements the functional specification. As commented above, in this work it corresponds to the IEC 61131 standard software model. This standard proposes two types of components: those that do not encapsulate code such as configurations, resources and tasks and those that encapsulate code, the Program Organization Units (POUs). Variables represent the data exchanged among software components. Models in this domain are managed by PLC programming tools.

While the former domain view defines the functional design, the two latter domain views define the implementation issues, i.e., "how" to develop the functional specification. In this sense, the domain views follow the MDA approach; the same control engineering model, representing the system functionality, could have different implementations in different platforms.

A fourth view relates the elements of the three domain views, assuring the definition of a complete and consistent model of the real system [23]. For instance, field connectors in the functional model are mapped univocally to I/O data connectors in I/O nodes of the hardware model and to global variables at configuration level in the software model. More detailed information on the lexicon and syntax and semantics of domain views for industrial control system can be found in [24].

This section illustrates the approach for defining domain languages proposed in Section III to the $3+1$ views of distributed industrial control systems, making use of the AutomationML editor [2].

A System Unit Class library for every domain view has been defined. The static semantics of a SUC, when needed, is defined as a rule in MathML linked to the SUC through an InterfaceClass. Fig. 1 illustrates a composition rule related to the control engineering view. In particular, it imposes a sequential level attribute for the components of the hierarchy.

A RoleClassLibrary is used to define the architectural style of the language, using sequence, choice, any and all roles. They are used within the SUC libraries for defining the architectural style associated to SUCs and Internal Elements.

Fig. 2 illustrates the definition of a domain language for the Control Engineering View using the CAEX standard as a backbone and MathML for defining the semantics and the composition rules.

A SUC library containing the concepts of the control engineering view enriched with the corresponding attributes has been defined. For instance, the SUC Component is characterized by its name, level and description attributes. The value of the first one is required and it must be unique. The level attribute is also required and it must have a non-negative value (it represents the level of the functional hierarchy the component belongs to). Finally, the description attribute is optional and it must have a string value.

The root element of the language is the ctrlEng element which is composed by a sequence of Components. This is defined through the ctrlEng SUC having an Internal Element Component that is an instance of the Component SUC commented

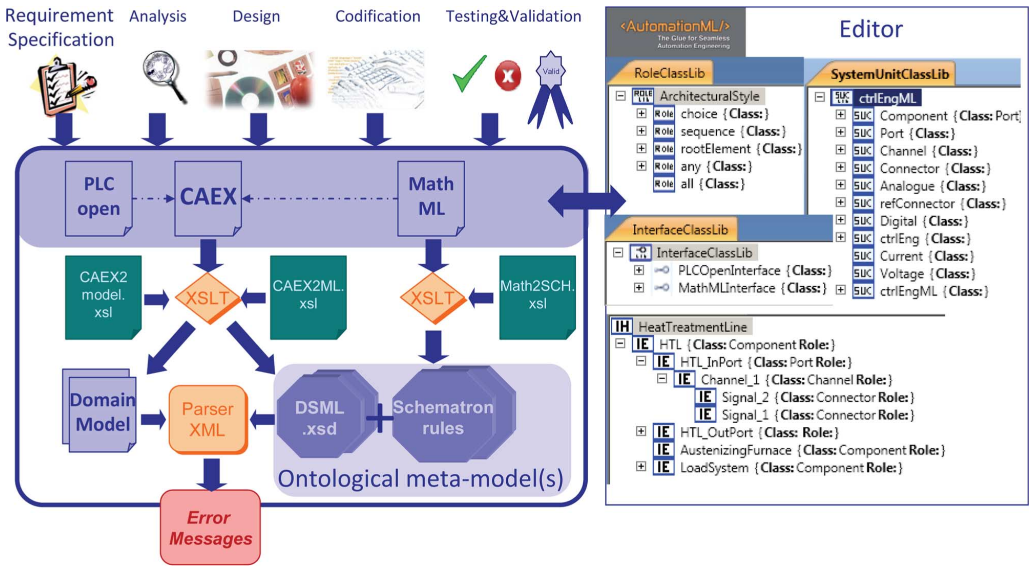


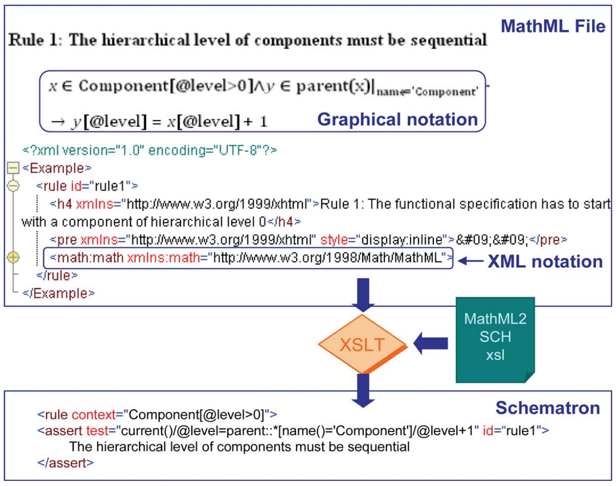
Fig. 3. CAEX-XML schema and model mappings.

above. Furthermore, the internal element is enriched with one sequence role (the value of minOccurs and maxOccurs attributes is one). The semantic associated to the Component SUC is defined in a mathML file (ctrlEng.mml) that is linked to the SUC via an external interface (MathMLInterface). The external interface has two attributes: the file name and the rule identifier. In this case, it corresponds to the Rule1 that checks if the hierarchy levels of the control engineering view are sequential (see Fig. 1).

Following this philosophy the other two domain languages have been defined. The software engineering language uses other external interface (PLCopenInterface) for linking the POU element to a PLCopen file containing the corresponding source code [1]. This external interface is also characterized by two attributes: the file (following the PLCopen interface) and the referenced POU in such file.

In order to map descriptions of the control system belonging to different domains, a fourth SUC library is used. Six types of SUC has been defined; three for establishing mappings among functional components, processing nodes and software components and another three for defining the mapping between field signals, hardware IOs, and global variables. Note that every domain model is managed by a particular tool that is used in a different phase of the development cycle. Thus, the mapping defined by means of this SUC library allows checking the coherence of the complete model. Besides using a separate SUC for the mappings makes easier the extension of any domain model without affecting the others.

One SUC is used to link functional components to configurations or to resources in the software architecture; there also are SUC types to link field signals to I/Os in the hardware architecture to variables at configuration level in the software architecture. As in the other views, a set of MathML rules have been defined that correct models must meet. The use of MathML for specifying semantics allows displaying to the user a graphical view of the constraints to be met, expressed in a mathematical editor. Thus, the definition of constraints could be done directly using such tools.



Rule 1: The hierarchical level of components must be sequential

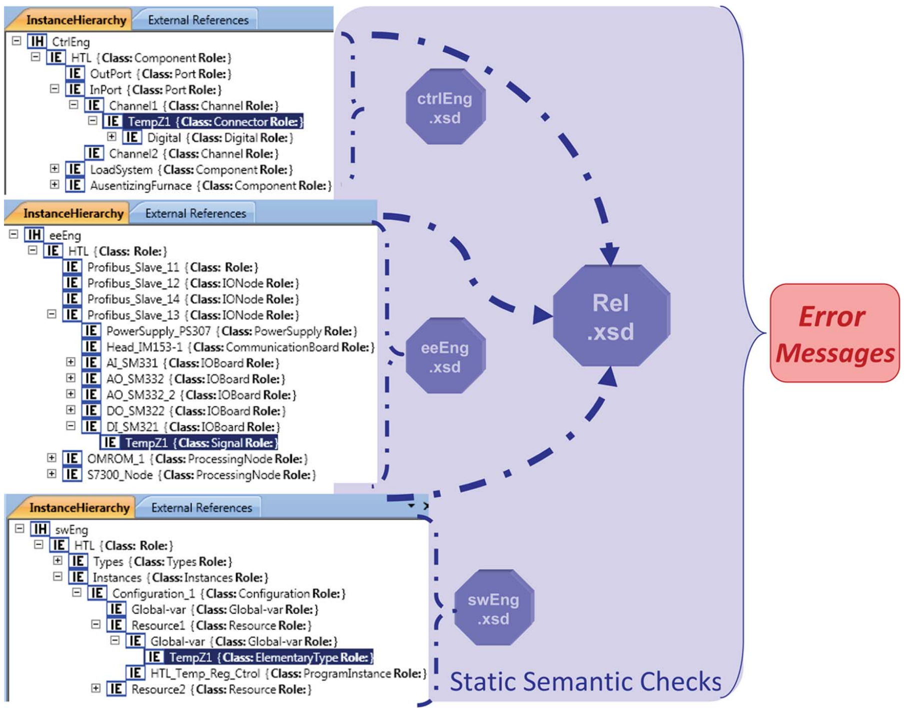
$$x \in \text{Component}[@\text{level}>0] \wedge y \in \text{parent}(x)|_{\text{name='Component'}} \rightarrow y[@\text{level}] = x[@\text{level}] + 1$$

```
<?xml version="1.0" encoding="UTF-8"?>
<Example>
  <rule id="rule1">
    <h4 xmlns="http://www.w3.org/1999/xhtml">Rule 1: The functional specification has to start
with a component of hierarchical level 0</h4>
    <pre xmlns="http://www.w3.org/1999/xhtml" style="display:inline">&#09;&#09;</pre>
    <math:math xmlns:math="http://www.w3.org/1998/Math/MathML">
  </rule>
</Example>
```

```
<rule context="Component[@level>0]">
<assert test="current()/@level=parent::*[name()='Component']/@level+1" id="rule1">
      The hierarchical level of components must be sequential
</assert>
```

Fig. 4. MathML to schematron rules transformation's example.

Fig. 5. Load system of the heat treatment line.


```
InstanceHierarchy   External References
⊟ IH CtrlEng
  ⊟ IE HTL {Class: Component Role:}
      IE OutPort {Class: Port Role:}
    ⊟ IE InPort {Class: Port Role:}
      ⊟ IE Channel1 {Class: Channel Role:}
        ⊟ IE TempZ1 {Class: Connector Role:}
          ⊞ IE Digital {Class: Digital Role:}
        IE Channel2 {Class: Channel Role:}
  ⊞ IE LoadSystem {Class: Component Role:}
  ⊞ IE AusentizingFurnace {Class: Component Role:}

InstanceHierarchy   External References
⊟ IH eeEng
  ⊟ IE HTL {Class: Role:}
      IE Profibus_Slave_11 {Class: Role:}
      IE Profibus_Slave_12 {Class: IONode Role:}
      IE Profibus_Slave_14 {Class: IONode Role:}
    ⊟ IE Profibus_Slave_13 {Class: IONode Role:}
        IE PowerSupply_PS307 {Class: PowerSupply Role:}
        IE Head_IM153-1 {Class: CommunicationBoard Role:}
      ⊞ IE AI_SM331 {Class: IOBoard Role:}
      ⊞ IE AO_SM332 {Class: IOBoard Role:}
      ⊞ IE AO_SM332_2 {Class: IOBoard Role:}
      ⊞ IE DO_SM322 {Class: IOBoard Role:}
      ⊟ IE DI_SM321 {Class: IOBoard Role:}
          IE TempZ1 {Class: Signal Role:}
    ⊞ IE OMROM_1 {Class: ProcessingNode Role:}
    ⊞ IE S7300_Node {Class: ProcessingNode Role:}

InstanceHierarchy   External References
⊟ IH swEng
  ⊟ IE HTL {Class: Role:}
    ⊞ IE Types {Class: Types Role:}
    ⊟ IE Instances {Class: Instances Role:}
      ⊟ IE Configuration_1 {Class: Configuration Role:}
          IE Global-var {Class: Global-var Role:}
        ⊟ IE Resource1 {Class: Resource Role:}
          ⊟ IE Global-var {Class: Global-var Role:}
              IE TempZ1 {Class: ElementaryType Role:}
            IE HTL_Temp_Reg_Ctrol {Class: ProgramInstance Role:}
        ⊞ IE Resource2 {Class: Resource Role:}
```

Fig. 6. HTL domain specific models.

VI. Domain Specific Markup Languages for Distributed IPMCS

In order to perform model instances validation it is necessary to obtain a domain specific ML against which models can be analyzed. The domain specific ML for IPMCS can be automatically extracted following the guidelines established in Section IV, using CAEX and MathML for the Control Engineering and Electrical-Electronic views and also using the PLCopen interface for the case of the Software Engineering view.

To obtain the composition rules, the MathML2SCH XML stylesheet has been developed. Fig. 4 illustrates an example of this mapping. This rule checks if the hierarchical level of the components, that constitute the control engineering view, is sequential.

The derived MLs for distributed industrial control systems can be used to check the correctness of models. In this section, the model validation procedure is illustrated through the modeling of an industrial Heat Treatment Line. These lines are commonly used for manufacturing metallic elements in many industries and its main goal is to provide the required quality characteristics for the final product. A typical line is composed by: furnaces, transport systems to charge and discharge the material, cooling tanks, washing and degrease machines, and presses.

Although these applications are commonly composed by a load system, an austenizing furnace, a tempering tank, a washing tank, and an annealing furnace, the modeling approach has been applied to two representative subsystems of the complete line: the austenizing furnace and the load system. Fig. 5 illustrates the load system of a real plant. Notwithstanding this, more than 100 field signals are managed. The design of the control system functionality for the complete line involves four hierarchical levels.

- Level 0 represents the plant.
- Level 1 is composed by components corresponding to each independent subsystem of the plant.
- Level 2 defines the set of components within every component of level 1. For instance, the austenizing furnace, a level 1 component, includes six level 2 components: the gas train control, the burner combustion control, the zone fan control, the combustion fan control, the temperature regulation, and the movements control.
- Level 3 is composed by elementary functional components. For instance, the level 2 component Zone Control contains three elementary blocks: the control of the temperature, the burners and the fan.

The control engineering view that represents both subsystems includes 50 internal elements, 20 for the load system and 30 for the austenizing furnace. Furthermore, 100 field signals have been defined. The inputs of the control system are defined at the level 0 component that represents the overall control system of the plant. These field signals can come from process, the human machine interface or the operator desk.

Fig. 6 illustrates the instance hierarchy elements for the three domain models. Note that these models may come from different engineering tools. In this example, they have been imported from AutomationML editor. The modeling approach proposed in this paper allows assuring the correctness of every domain model. This is achieved by checking the model instance elements against the corresponding XML schema and composition rules, as Fig. 6 shows. Besides and in order to check the correctness of the whole model, represented by the three views, it is also necessary to check the external references among elements in different domain models. For instance, one of these cases is highlighted in Fig. 6, where the process signal in the control engineering view corresponding to the temperature in zone 1 of the austenizing furnace (TempZ1 process signal) must be mapped to data within a IO board in the electronic-electrical view and to a global variable of a Resource in the software engineering model. This also implies checking that the PLC that manages the IO board is mapped to the Configuration that contains such Resource.

The right part of Fig. 6 shows that the domain markup language and the composition rules are extracted from the CAEX model and how the analysis of the model instances is performed. Every model instance is checked against the domain schema and composition rules that define the ontological metamodel of the domain. The complete application consisting of the three views and the mapping among them is checked against the relationships schema (Rel.xsd) and composition rules, through the information defined in the external reference elements.

Using the same technologies, it is possible to generate the code skeleton for the automation projects of the PLCs present in the application as well as any type of documentation contained in the model [26]. The code skeleton would consist of the definition of the structure of the automation project in terms of configuration, resources, access variables, and global variables (at configuration and resource level), as well as the complete definition of the program instance interface.

VII. Conclusion and Future Work

The work presented here applies the MDD concepts in the Industrial Automation field. The main contribution of the work is a tool-independent approach that supports model collaboration along the application development cycle. The guidelines to define domain markup languages from standard common representation formats have been established supporting domain model definition and validation. The commented approach has as input the domain languages implemented by means of three well spread standards: CAEX, MathML and PLCopen and it directly derives the markup languages composed by the abstract syntax, the composition rules and the static semantics. Using a set of powerful technologies related to XML, the correctness of application models is assured.

This approach has been successfully applied to the design of an industrial heat treatment line. The definition of the complete application consists of the definition of three views: the high-level design, the hardware architecture, and the software architecture and the mappings among them.

The use of this model-based approach during the different phases of the development cycle allows detecting errors at early stages of the design through model validation, reducing cost and development time. Furthermore, the modeling approach allows checking the consistency of domain models coming from heterogeneous domain tools. Hence, the use of models improves the design of systems.

In order to complete the model driven development process, the four ML can be used as the core of an integrated design framework in order to achieve tool integration through model collaboration.

References

[1] E. van der Wal, "PLCopen," *IEEE Ind. Electron. Mag.*, vol. 3, p. 25, Dec. 2009.

[2] A. Lüder, E. Estévez, L. Hundt, and M. Marcos, "Automatic transformation of logic models within engineering of embedded mechatronical units," *Int. J. Adv. Manuf. Technol.*, pp. 1077–1089, Nov. 2011, DOI: 10.1007/s00170-010-3010-y.

[3] *Programmable Controllers, Part 3: Programming Languages*, IEC International Standard IEC 61131-3:2003, 2003, International Electrotechnical Commission.

[4] B. Selic, "The pragmatics of model-driven development," *IEEE Software*, vol. 20, no. 5, pp. 19–25, Sep./Oct. 2003.

[5] J. Miller and J. Mukerji, "Model Driven Architecture (MDA)," Architecture Board ORMSC1, Jul. 2001, OMG, ormsc/2001-07-01.

[6] J. Sztipanvits and G. Karsai, "Model-integrated computing," *IEEE Comput.*, vol. 20, pp. 110–112, 1997.

[7] G. Karsai, "A configurable visual programming environment: Toolf or domain-specific programming," *Computer*, vol. 28, no. 3, pp. 36–44, 1995.

[8] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing domain-specific design environments," *Computer*, vol. 34, no. 11, pp. 44–51, 2001.

[9] L. Lo Bello and G. Frey, "Guest Editorial (Special Section on Information Technology)," *IEEE Trans. Ind. Informat*, vol. 7, no. 4, pp. 688–688, Nov. 2011, DOI: 10.1109/TII.2011.2167341.

[10] R. Khare and A. Rifkin, "XML: A door to automated Web applications," *IEEE Internet Comput.*, vol. 1, no. 4, pp. 78–87, Jul./Aug. 1997.

[11] M. Fedai and R. Drath, "CAEX—A neutral data exchange format for engineering data," ATP International Automation Technology 01/2005 3 2005, pp. 43–51.

[12] N. F. Polys, D. Brutzman, A. Steed, and J. Behr, "Future standards for immersive VR: Report on the IEEE virtual reality 2007 workshop," in *IEEE Comput. Graphics Appl.*, 2008, vol. 28, no. 2, pp. 94–99.

[13] J. El-Khoury, "A model management and integration platform for mechatronic product development," Ph.D. dissertation, School of Ind. Eng. Manage., Stockholm, Sweden, 2006.

[14] L. Baresi, M. Mauri, and M. Pezze, "PLCTools: Graph transformation meets PLC design," *Electronic Notes in Theoret. Comput. Sci.*, vol. 72, no. 2, pp. 1–10, 2002.

[15] K. Thramboulidis, D. Perdikis, and S. Kantas, "Model driven development of distributed control applications," *Int. J. Adv. Manuf. Technol.*, vol. 33, pp. 233–242, DOI 10.1007/s00170-006-0455-0.

[16] K. Thramboulidis, "Model-integrated mechatronics—Toward a new paradigm in the development of manufacturing systems," *IEEE Trans. Ind. Informat.*, vol. 1, no. 1, pp. 54–61, Feb. 2005.

[17] *IEC, IEC/TR 61499-3: Function Blocks—Part 3 Tutorial Information*, International Standard, International Electrotechnical Commission, Geneva, 2005-01, 2005, 1st ed..

[18] K. Thramboulidis, "The $3+1$ SysML view-model in model integrated mechatronics," *J. Softw. Eng. Appl.*, vol. 3, pp. 109–118, 2010.

[19] V. V. Vyatkin, J. H. Christensen, and J. L. M. Lastra, "OOONEIDA: An open, object-oriented knowledge economy for intelligent industrial automation," *IEEE Trans. Ind. Informat.*, vol. 1, no. 1, pp. 4–17, Feb. 2005.

[20] G. Cengic and K. Akesson, "On formal analysis of IEC 61499 applications, part A: Modeling," *IEEE Trans. Ind. Informat.*, vol. 6, no. 2, pp. 136–144, May 2010.

[21] G. Cengic and K. Akesson, "On formal analysis of IEC 61499 applications, part B: Execution semantics," *IEEE Trans. Ind. Informat.*, vol. 6, no. 2, pp. 145–154, May 2010.

[22] T. Strasser, C. Sunder, and A. Valentini, "Model-driven embedded systems design environment for the industrial automation sector," in *Proc. 6th IEEE Int. Conf. Ind. Informat.*, Daejeon, Korea, 2008, pp. 1120–1125.

[23] K. R. Foster, "Math on the Internet," *IEEE Spectrum*, vol. 36, no. 4, pp. 36–40, 1999.

[24] P. Kruchten, "Architectural blueprints—The "$4+1$" view model of software architecture," *IEEE Software* vol. 12, no. 6, pp. 42–50, Nov. 1995.

[25] E. Estévez and M. Marcos, "Model-driven approach for designing industrial control systems," *Lecture Notes on Computer Science*, vol. 4758/2007, pp. 284–287, DOI: 10.1007/978-3-540-75132-8_25.2007.

[26] M. Marcos, E. Estévez, F. Pérez, and E. van der Wal, "XML exchange of control programs," *IEEE Ind. Electron. Mag.*, vol. 3, pp. 32–35, Dec. 2009.

[27] E. Estévez, M. Marcos, and D. Orive, "Automatic generation of PLC automation projects from component-based models," *Int. J. Adv. Manuf. Technol.*, vol. 35, no. 6, pp. 527–540, 2007.

E. Estevez received the Degree in telecommunications engineering in 2002 and the Ph.D. degree in automatic control in 2007 from the University of the Basque Country (UPV/EHU), Bilbao, Spain.

She is a Lecturer of Electronic and Automatic Engineering at the University of the Jaen, Jaen, Spain. She has coauthored more than 70 technical papers in international journals and conference proceedings in the field of distributed industrial control systems. She has also been involved in several research projects funded by National and European R&D programs. Her main expertise deals with applying software engineering concepts to the industrial control field. She has carried out review work for various conferences and technical journals.

M. Marcos (M'88) is a Professor of Automatic Control and Systems Engineering at the University of the Basque Country, Bilbao, Spain, where she has been Chairman of the Automatic Control and Systems Engineering Department for more than ten years. She has authored and coauthored more than 80 technical papers in international journals and conference proceedings. She has acted as the main researcher of more than 60 research projects funded by National and European R&D programmes. She has carried out review work for various conferences and technical journals.

Prof. Marcos has served on technical committees of IFAC (AARTC, WRTP, CC), IEEE (NBCS), and she has been a member of the European Control Association Council, the National Organizing Committee of IFAC Spain, and Publication Co-Chair for the IEEE Control and Decision Conference (CDC2005). She has been the General Co-Chair of the IEEE International Conference on Emerging Technologies for Factory Automation 2010.