FORMAL MODELLING OF INDUSTRIAL DISTRIBUTED CONTROL SYSTEMS

*** M. Marcos, E. Estévez**

**Departamento de Ingeniería de Sistemas y Automática,*
Universidad del País Vasco, UPV/EHU (Spain)

Abstract: Industrial Process Measurement and Control Systems (IPMCS) are used in most of the industrial sectors to achieve production improvement, process optimisation and time and cost reduction. Integration, reuse, flexibility and optimisation are demanded to adapt to a rapidly changing and competitive market. There is also a growing requirement that all software tools that support the different phases of the development process (design, configuration, management) can be integrated as well. Thus, a consolidation of modelling methodologies for achieving this goal is needed. This paper presents a formal modelling for IPMCS that captures all aspects of the system to design in terms of functionality and implementation (hardware and software). The modelling language (eXtensible Markup Language, XML) allow to implement model validation as well as to easily transform information coming from / going to different software tools, achieving tool integration.

Keywords: distributed control systems, formal modelling, programmable logic controllers, industrial applications.

1. INTRODUCTION

Nowadays most of the industrial sectors use Programmable Logic Controllers (PLCs) to achieve the control of their productive systems. In the last years, technological advances in these controllers allow the production improvement, process optimisation and time and cost reduction. The application of standards has also a great force in the fast growth of the control and instrumentation of industrial processes. The International Electrotechnical Commission has published several standards promoting interchangeable and open systems. The IEC 61131-3 standard deals with the software model and programming languages for IPCMS. The evolving standard IEC 61499 standard (IEC, 2000) focuses on distributed IPMCS based on an extended definition of Function Blocks and it provides the requirements for software tools to support the specification, analysis and validation of Distributed IPMCS as well as the configuration, implementation, operation and maintenance of such systems. However, as fast as industry reaches a greater maturity level, a consolidation of the modelling methodologies becomes necessary. Therefore, modelling languages, that allow system description and definition before their construction, must be used. The modelling methodology should allow to model the system from different viewpoints as well as to involve the entire system lifecycle.

In this sense, PLCopen (PLCopen, 2003) 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 this field. For this, PLCopen has several technical and promotional committees (TCs). In particular, TC6 for XML had as original goal to define an open interface to communicate different programming tools. But, from the beginning, TC6 members realized that having XML as a common road, other tools, like simulation and modelling tools, or documentation and version control tools could be integrated.

Some attempts have been done by different authors towards the use of well known modelling languages for designing IPMCS. Bonfé and Fantuzzi (2000), Heverhagen and Tracht, (2001) propose the use of UML (Jacobson, et al., 1992), (Rumbaugh, et al., 1996) for specifying components of control systems following the IEC 61131-3 standard. But they only model the implementation issues for this software model and their goal is to generate source code. There are other works that focus on the design of

applications using the IEC 61131 standard. Gonzalez et al. (2003) propose a methodology for the analysis and modelling of discrete event systems applied to the development of the control logic based on the IEC 61131-3 standard. In Kandare (2001), a graphical tool that generates IEC 61131-3 structured text (ST) code is described.

On the other hand, work is being done towards the design and development of a framework to support the engineering process of distributed IPMCS (Thramboulidis and Tranoris 2001, Thramboulidis 2003). The design is object oriented and uses UML to define a 4-layered architecture for designing such type of systems.

Previous works of authors Marcos et al. (2004) propose a model for industrial distributed control systems also using UML and the Model Driven Architecture (MDA) methodology (Millar and Mukerji, 2001), proposed by OMG (OMG 2002). Following the MDA concept, the distributed IPMCS is modelled in two parts that are complimentary: the functionality and the implementation of this functionality following the IEC 61131-3 standard software model. The model is completed by mapping the functionality to the concrete IEC 61131-3 architecture. The main problem that arises in this UML-based modelling is to check the consistence (the definition of the parts must follow a strict formal model) and the coherence of the overall system being modelled (the parts, functionality and hardware and software architecture, refers to the same industrial distributed control system). The modelling language must offer tools to define formal models and to perform syntactic and semantic analysis of model instances (applications). The current version of UML does not support this.

This paper presents a formal modelling for distributed IPMCS using XML as modelling language, given that XML offers powerful technologies for defining formal grammars and performing the necessary coherence and consistence analysis. This model of an application will be generated / consumed in part by different software tools involved in the life cycle of the applications.

The layout of the paper is as follows: section 2 briefly describes the modelling requirements from which arise the requirements that must be met by the modelling language. Section 3 presents a formal modelling for distributed IPCS using XML, the modelling language selected. Finally, section 4 illustrates the proposed modelling methodology in an industrial application: a Heat Treatment Line.

2. MODELLING OF DISTRIBUTED IMPCS

Usually, an important part of the control system corresponds to the modules that control the basic mechanical components. In addition, different communication modules exist between the part which controls the basic components and the main plant controllers. For this reason, the definition of the system in a modularized and hierarchical way presents clear advantages. Within a hierarchical design, each module represents the control of a set of lower level components.

In order to feed documentation and configuration tools with the information from the distributed IPMCS system model, other independent part of this model should inform about the hardware components that are used to implement the system (controllers e.g. PLCs, networks, industrial buses, etc.). This part of the model captures the information related to the hardware characteristics of the implementation.

To encourage the use of standards, as providers of hardware independent methods for implementing the control, measurement and monitoring functions, a third part of the model should referred to the software architecture following the IEC 61131-3 standard. This assures modularity and reusability of applications.

In summary, the main requirements in the design and development of industrial distributed control applications are: related to functionality, a modular and hierarchical specification of the control system. Related to implementation, a model of the hardware components and a model of the software architecture for each of the processing elements. This modelling, based in three separate parts, allows partial or full reuse of both, applications and software. The model has to be completed with the relationship between the three different parts, as the components of the functional specification are mapped to specific processing elements (hardware components) and each processing element has a specific software architecture.

From the requirements that the modelling of distributed IPMCS must meet, the requirements of the modelling language can be identified:

Firstly, it must allow to define a formal grammar for each of the separate parts of the model: functionality, hardware components and software architecture. Besides that, it has to provide means for achieving syntactic and semantic analysis of each part.

Secondly, it must allow to check if the application model specifies a coherent mapping between the parts.

Finally, as each part of the model can be generated / consumed by different software tools, the modelling language should provide means for extracting part of the information contained in the model instance.

2.1 XML as modelling Language

XML is a formal language and it comes from SGML (Standard Generalized Markup Language). XML documents contain data that are organized following a tree hierarchical structure. The minimal information units of this structure are XML elements.

Each element can be characterized by XML attributes. There are different standards in XML technologies (Arciniegas 2001, Simpson 2001) that can be used for modelling distributed IPMCS as well as for performing model coherence checks and for obtaining part of the information contained in the model. In particular, the XML schema standard (Van der Vlist, 2002) can be used to define the formal grammar for specifying the functionality and the hardware and software architecture as established in the previous sub-section. In order to perform coherency and consistency checks of the different parts of the model (semantic analysis) as well as the overall model consistency, the XML schema standard can be complemented with XML *schematron* rules (Schematron, 2001). These are necessaries in order to assure a correct mapping between the model functionality and implementation as well as to guarantee a correct value of critical fields from either functionality or implementation. Finally, the XML *stylesheets* (XSL) (Tidwell, 2001) can be used to perform transformations of XML documents.

3. FORMAL MODELLING OF DISTRIBUTED IPMCS

The XML Technologies have been selected for the modelling of distributed IPMCS. In particular, the XML schema standard is used to define the formal grammar of each part of the model: the functionality (Platform Independent Model, PIM) that defines 'what' the control application has to do and the Platform Specific Model (PSM) that defines 'how' to do it in terms of the hardware components and the software architecture of each processing resource. Fig. 1 illustrates this separation of concerns proposed by MDA defined in XML.

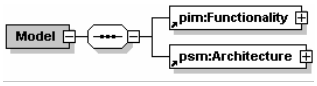

Fig. 1. Model general overview

The *Functionality* XML element defines how to model the hierarchical specification of the control system, independent from the technologies used for implementing it. On the other hand, the *Architecture* XML element defines how to specify a concrete implementation. It is constituted by the hardware architecture (a set of hardware components, nodes, interconnected via networks) and the software architecture following the IEC 61131-3 standard software model. The following sub-sections detail the main characteristics of each XML element.

3.1 Functionality (PIM)

The Functionality element defines a generic hierarchical specification. It is based on components (*Functional_Basic_Component*), characterized by its hierarchical level and its connectors (*Inputs* and *Outputs*). Fig. 2 shows the characteristics of these components.

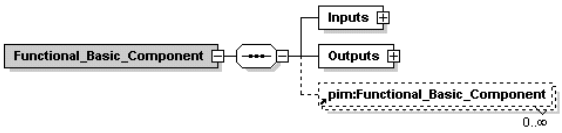

Fig. 2. Functional Basic Component

Each FBC is characterized by two XML attributes: *Name* and *Level*, and by an optional *Description* field. The *Name* identifies the FBC and the *Level* represents its location in the hierarchy. The latter XML attribute guarantees that the hierarchical level of the component is between 1 and N, but this does not assure that the level in the hierarchy is assigned sequentially. To do that, it is necessary a *schematron* rule as illustrated in Fig. 3.

```
<rule context="pim:Functional_Basic_Component">
    <assert test="current()/@Level=0 or
current()/@Level=parent::*[name()='pim:Functional_Basic_Component']/@Level+1"
priority="high">
    The value of level in the Functional Basic Component is not correct
    </assert>
</rule>
```

Fig. 3: *Schematron* rule Sequential hierarchical level

The other elements that compose the functionality are the connectors that can be either Input or Output to/from a component. A *Connector* is characterised by its Name, Type and optionally a Description. Fig. 4 shows the connectors characteristics expressed in XML.

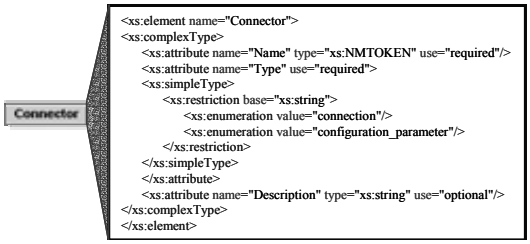

Fig. 4. *Connector* XML element

A *connector* can be a *connection* (field signal) or a *Configuration Parameter* (in case of an input Connector). Note that if a connector is an output from a FBC, the type of this connector can only be connection. In order to assure the correct type of the connector another *schematron* rule is developed.

3.2 Implementation (PSM)

In this sub-section the modelling of implementation issues is detailed. As commented before, the Architecture models the implementation issued in terms of the hardware architecture (nodes and networks) and the software architecture of each processing element (following the IEC 61131-3 software model). Fig. 5 illustrates the two XML elements that define the PSM.

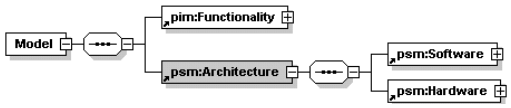

Fig. 5. Platform Specific Model

The IEC 61131-3 software model defines a set of elements to specify the execution of the software that is to run in a processing element (e.g. PLC). Fig. 6 illustrates the elements of this model expressed in XML.

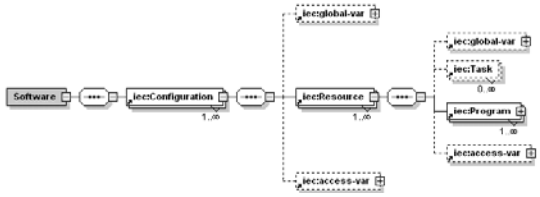

Fig. 6. IEC 61131-3 Software model in XML

The elements of the IEC 61131-3 software model are the following:
Configurations: processing elements. E.g. PLC or OpenPLC. *Resource*: It provides support for program execution. e.g. CPU or a Virtual Machine. *Task*: It allows the designer to control the execution rates of different parts of the program. *POU*: Program Organisation Units that are *Programs*, *Function Blocks* and *Functions*. They provide software reuse. *Variables*: Defined by their visibility: global in a configuration/resource level or local in programs. As any other high level programming language, they are characterised by their type and value.

The characteristics of each element have been expressed in XML. As an example the characteristics of a IEC 61131-3 Variables are illustrated in Fig. 7

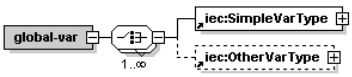

Fig. 7. IEC 61131-3 variable

The type of a variable can be either Elementary, as defined by the standard (*iec:SimpleVarType*), or user defined (*iec:OtherVarType*). The elementary types have also been defined and characterized in XML. Fig. 8 illustrates the TIME type.

```
<xs:simpleType name="TIME">
<xs:restriction base="xs:string">
    <xs:pattern
value="(T|t|TIME|time)(#\d*\p{P}?\d*h|#\d*h\d*\p{P}?\d*(m|s|ms)|#\d*h\d*m\d*\p{P}?
\d*s|#\d*h\d*(m|s)\d*\p{P}?\d*ms|#\d*h\d*m\d*s\d*\p{P}?\d*ms)"/>
    <xs:pattern value="(T|t|TIME|time)(#\d*\p{P}?\d*m|#\d*m\d*\p{P}?\d*(s|ms))"/>
    <xs:pattern value="(T|t|TIME|time)(#\d*\p{P}?\d*s|#\d*s\d*\p{P}?\d*ms)"/>
    <xs:pattern value="(T|t|TIME|time)#\d*\p{P}?\d*ms"/>
</xs:restriction>
</xs:simpleType>
```

Fig. 8. TIME (IEC 61131-3 Elementary) Data type

Hardware Architecture. The hardware architecture of an industrial control system can be defined as a set of network nodes (nodes) connected through a set of network segments (buses).

The Bus element represents a network segment characterized by its application layer. A Node element is connected to a network segment through a communication board. There can be two types of nodes: those that have processing resources (e.g. PCs, PLCs, OpenPLC) and those that are only used as input/output (e.g. PROFIBUS_DP slaves). The former contains processing resources (CPUs) and memory cards. Fig.9 illustrates the Node characteristics in XML.

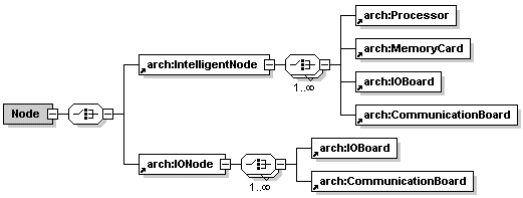

Fig. 9. Characteristics of the Node elements

Both types of node element have a *CommunicationBoard*, as Fig. 10 illustrates.

```
<xs:element name="CommunicationBoard">
<xs:complexType>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="type" use="required">
        <xs:simpleType>
        <xs:restriction base="xs:string">
        <xs:enumeration value="Ethernet"/>
        <xs:enumeration value="Profibus"/>
        <xs:enumeration value="#Others"/>
        </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="refBus" type="xs:string" use="required"/>
    <xs:attribute name="Address" use="required">
        <xs:simpleType>
        <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{1,3}\p{P}[0-9]{1,3}\p{P}[0-9]{1,3}\p{P}[0-9]{1,3}"/>
        <xs:pattern value="\d*"/>
        </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    ...
    <xs:attribute name="portAddress" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
```

Fig. 10. Communication Board

A *CommunicationBoard* is characterised by its name, type, the name of the network it belongs to (*refBus*) and Address. This attribute must follow a pattern that depends on its type. E.g. if it is an Ethernet *CommunicationBoard*, its address must follow the pattern *xxx.xxx.xxx.xxx* as Fig. 10 illustrates.

Relationship between both architectures. The XML *schema* assures a correct modelling of the hardware and software architectures separately. But it is also necessary to guarantee that both architectures are coherent and consistent between them. To do this, a set of *schematron* rules have been implemented. Fig. 11 shows two examples.

```
<rule context="iec:Configuration">
    <assert test="((count(//iec:Configuration)-1) -(count(//iec:Configuration/iec:Resource[@onProcessor
!=current()/iec:Resource/@onProcessor])- count(//iec:Configuration/iec:Resource[@onProcessor
=current()/iec:Resource/@onProcessor])) =0 or  ((count(//iec:Configuration/iec:Resource[@onProcessor
!=current()/iec:Resource/@onProcessor])- count(//iec:Configuration/iec:Resource[@onProcessor
=current()/iec:Resource/@onProcessor]))-(count(//iec:Configuration)-1))=0 or (count(//iec:Configuration) -
count(//iec:Configuration[@Name!=current()/@Name]))=1" priority="high">
        The mapping between a configuration and an intelligent node must be one to one
    </assert>
</rule>
<rule context="iec:Resource">
    <assert test="count(current()[@onProcessor=//arch:IntelligentNode/@name])=1" priority="high">
        A resource must be downloaded to an existing intelligent node
    </assert>
</rule>
```

Fig. 11. schematron rules for HW and SW mapping

For instance, it is necessary to check that a configuration is mapped to an intelligent node, or the resources of the same configuration are mapped to the same existing intelligent node.

3.3 Model coherence and consistence checks

A set of *schematron* rules have been implemented in order to assure that the overall modelled application is coherent and consistent. Thus, it is necessary to assure that the implementation is consistent and coherent with respect to the designed functionality. Some of the most important checks are (see Fig. 12):

A lowest level FBC is mapped to a POU instance.A program can only contain Function Blocks.A FBC belonging to level 1 to N-1 can optionally correspond to a POU instance.

```
<rule context="pim:Functional_Basic_Component">
    <assert test="count(pim:Functional_Basic_Component)=0 and
count(//iec:Program[@Name=current()/@refPSMElement]) +
count(//iec:FunctionBlock/iec:Repository/*[@Name=current()/@refPSMElement]) +
count(//iec:FunctionBlock/iec:NewPOU[@Name=current()/@refPSMElement])=1 or
count(./pim:Functional_Basic_Component)!=0" priority="high">
    A Level N Functional Basic Component must corresponds to a Program Organisation Unit instance
    </assert>
    <assert
test="count(parent::*[name()='pim:Functional_Basic_Component'][@refPSMElement=//iec:Program/@Name])
=1 and count(current()[@refPSMElement=//iec:FunctionBlock/iec:NewPOU/@Name]) +
count(current()[@refPSMElement=//iec:FunctionBlock/iec:Repository/*/@Name])=1 or
count(parent::*[name()='pim:Functional_Basic_Component'][@refPSMElement=//iec:Program/@Name])!=1"
priority="high">
    When the upper level FBC corresponds to a program, the FBC must corresponds to a Function Block
    </assert>
    <assert test="count(./pim:Functional_Basic_Component)!=0 and @refPSMElement and
count(current()[@refPSMElement=//iec:Program/@Name])+
count(current()[@refPSMElement=//iec:FunctionBlock/iec:Repository/*/@Name])+count(current()[@refPSMEl
ement=//iec:FunctionBlock/iec:NewPOU/@Name]) = 1 or not(@refPSMElement) or
count(./pim:Functional_Basic_Component)=0" priority="high">
    An FBC belonging to level 1 to N-1 only can corresponds to POU instance
    </assert>
</rule>
```

Fig. 12. Consistency analysis of the overall model

4. CASE STUDY: A HEAT TREATMENT LINE

This section illustrates the proposed modelling methodology as applied to an industrial case study: the distributed control system of a Heat Treatment Line.

Fig. 13 illustrates a typical Heat Treatment Line (HTL) that is composed by the following sub-systems: a Load System, an Austenizing Furnace, a Tempering Tank, a Washing Tank and an Annealing Furnace.

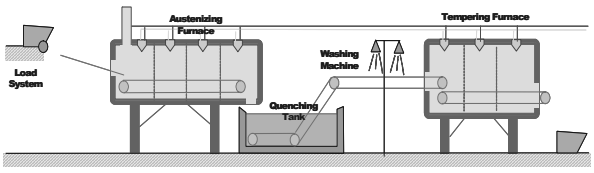

Fig. 13. General overview of a Heat Treatment Line

Let us apply the modelling methodology proposed in section 3 to two representative sub-systems of the complete line: the Austenizing Furnace and the Load System.

The *Austenizing Furnace* with four zones and two burners per zone. The temperature regulation is performed in each of the four zones, where the temperature should be around 850º C. A conveyor belt moves the pieces through the furnace. The speed of the conveyor depends on the required heating treatment.

The design of the control system functionality for the complete line involves four hierarchical levels:

Level 0: The plant. Level 1: Components corresponding to each independent subsystem of the plant. In this case, the Austenizing Furnace and the Load System. Level 2: Each level 1 component is composed by a set of level 2 components. For instance, the Austenizing Furnace, a level 1 component, includes 6 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: This level is composed by elementary functional components. For instance, the level 2 Movements Control Component contains three elementary blocks: the Conveyor Control, the Conveyor Movement Control and the Set Point level. All three belongs to the third level of the functional hierarchy. Fig. 14 illustrates this functionality expressed in XML.

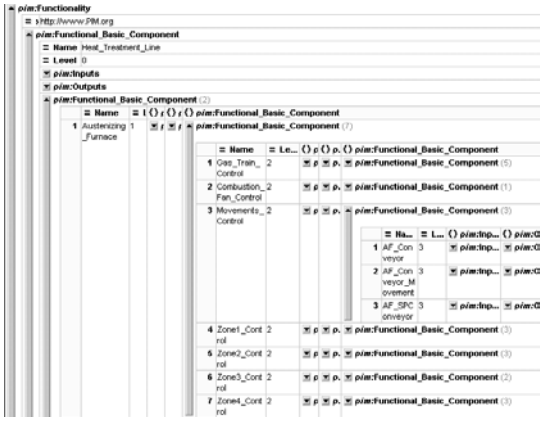

Fig. 14. HTL functionality

The control system is implemented in two OMRON Open Network Controller (a Configuration element of the IEC 61131-3 standard), one containing two Resources and the other one. Fig. 15 shows the hardware architecture for this implementation.

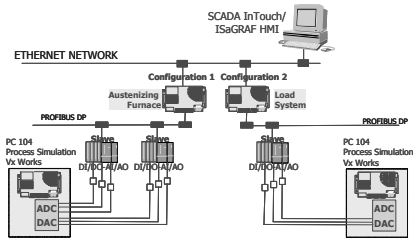

Fig. 15. General scenario of the HW components

Fig. 16 illustrates the part of the model instance that corresponds to the hardware architecture.

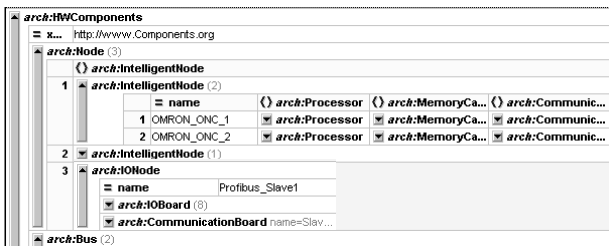

Fig. 16. Hardware architecture of HTL

Fig. 17 shows part of the software architecture of the model instance.

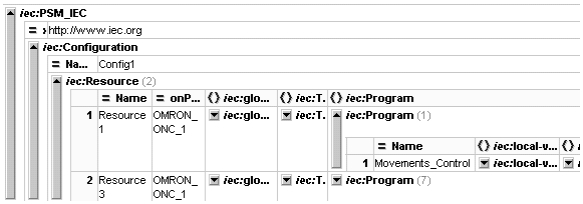

Fig. 17. Software architecture of HTL

This application has been developed within the FLEXICON project. The general goal of the project, financed by the European Union's Information and Science Technologies programme, is to develop methodologies that enable Commercial Off-The-Shelf (COTS) tools integration for the design and deployment of Distributed Control Systems (DCS) with high degree of flexibility, dependability and re-usability. Within the FLEXICON toolset, the modelling tool is UML. Once the application model is defined in terms of the functionality of the control system and the hardware and software architectures of the implementation, the information captured is expressed in XML as a model instance. This XML file is validated against the proposed schema and content and cross content checks are performed via the *schematron* rules commented in section 3. Finally, as one of the goals of the FLEXICON project is to generate IEC 61131-3 ST code, the necessary stylesheets have been developed in order to generate source code and download the application code into the target processing resources.

5. CONCLUSIONS

XML technologies have been proved to be very powerful to implement formal modelling of distributed IPMCS. In this paper, a formal model for this kind of applications has been described. XML schemas are used to formally described the model of the distributed system. This allows formal validation of model instances. *Schematron* rules can be used for achieving validation contents and cross contents. Finally, *stylesheets* allow transformation between XML documents and extract information from an XML document. Thus, the proposed formal model for distributed IPMCS and the XML technologies allow to integrate different software tools for generating /consuming part of the model instance. Besides that, the work presented here is completely compatible with the goals of PLCopen TC6 XML and it can also be extended to support the requirements of the evolving IEC 61499 standard to model distributed IPMCS.

6. ACKNOWLEDGEMENTS

This work has been supported in part by EU-IST programme under project IST-2001-37269 and by MCYT&FEDER under project DPI 2003-2399.

REFERENCES

Arciniegas, F. (2001). Programación avanzada con XML, McGraw-Hill.

Bonfé, M., Fantuzzi, C. (2000) "Mechatronic Objects encapsulation in IEC 1131-3 Norm". Proceedings of the 2000 IEEE Int. Conf. on C A, pp.598-603.

Gonzalez, V. M., F. Mateos, N. Amos (2003). MLAV. Object-Oriented Methodology for the Analysis and Modelling of the Control Logic of Discrete Event Systems, SSGRR 2003.

Heverhagen, T., Tracht, R. (2001) "Integrating UML-RealTime and IEC 61131-3 with Function Block Adapters". Proceedings of the IEEE International Symposium on OO RT Distributed Computing.

IEC Technical Committee TC65/WG6 (2000). IEC61499 Industrial-Process Measurement and Control – Specification. IEC Draft.

Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G., (1992) "Object - oriented software engineering.". Addison-Wesley.

Kandare, G. (2001). Model-based software design for procedural process control with programmable logic controllers. The 2nd Int. PhD student workshop on systems and control, [COBISS.SI-ID 16409639]

Lewis, R.W., (1998) "Programming Industrial Control Systems using IEC 1131-3". IEE Control Engineering Series.

Marcos M., Estévez E., Gangoiti U., Sarachaga I. (2004). UML Modelling of distributed control systems, Proc. of the 6th Portuguese Conference on Automatic Control CONTROLO 2004,.

Millar J, Mukerji J. (2001) Model Driven Architecture (MDA). OMG, ormsc/2001-07-01.

OMG (2002). *OMG IDL Syntax and Semantics*. www.omg.org/docs/formal/02-06-07.pdf .

PLCopen (2003). *PLCopen for efficiency in automation*. www.PLCopen.org

Power Douglas, B. (1998) "Real Time UML developing efficient objects for embedded systems". Addison Wesley.

Rumbaugh, J., Blaha, M., Premerlan, W., Eddy, F.,Lorensen, W., (1996) "Modelado y diseño orientados a objetos. Metodología OMT". Prentice Hall.

Schematron (2001). http://xml.ascc.net/schematron/

Simpson, J.E. (2001). Just XML. second Edition, Prentice Hall PTR.

Thramboulidis, K. and C. Tranoris (2001). An Architecture for the Development of Function Block Oriented Engineering Support Systems. CIRA 2001.

Thramboulidis, K. (2003). An Architecture to Extend the IEC 61499 Model for Distributed Control Applications. 7th Conference on Automation Technology (Automation 2003).

Tidwell, D. (2001). XSLT, Ed. O'REILLY.

Van der Vlist, E.(2002). XML Schema,. Ed. O'REILLY.