Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Towards the conformance Analysis of IEC 61131-3 PLC Programming Tools

M. Marcos, E. Estévez, I. Sarachaga, A. Burgos

Automatic Control and Systems Engineering Department, University of the Basque Country,
Bilbao, Spain (e-mail: marga.marcos@ehu.es, elisabet.estevez@ehu.es, isabel.sarachaga@ehu.es, ispbufea@ehu.es).

Abstract: Industrial Process Measurement and Control Systems are used in most of the industrial sectors to achieve production improvement, process optimization and time and cost reduction. Integration, reuse, flexibility and optimization are demanded to adapt to a rapidly changing and competitive market. In fact, standardization is a key goal to achieve these goals. The international standardization efforts have lead to the definition of the IEC 61131 standard. Part 3 of this standard defines a software model for defining automation projects as well as 5 programming languages. Nowadays, a major part of Programmable Logic Controllers (PLC) vendors follows this standard, although each programming tool adds particularities and stores the automation project in different manner. This work presents an approach for analyzing the conformance to the IEC 61131-3 standard of a programming tool. The conformance analyser is designed at different levels, covering the data types, programming languages and software architecture. Besides, within each level of conformance, different sublevels can be certified.

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 optimization and time and cost reduction. On the other hand, for many years, only proprietary programming languages could be used for vendor specific equipment. Although some languages, such as ladder diagram or instruction list were very widespread, their implementation used to be rather different. It was obvious the need of standardization in the field, covering from the hardware to configuration issues, up to the programming languages. In 1993, the International Electrotechnical Commission (IEC) published the IEC 61131, International Standard for Programmable Controllers (IEC, 2003).

The IEC 61131-3 standard deals with the software model and programming languages for Industrial Process Measurement and Control Systems (IPMCS) (Lewis, R.W, 1998), (John, K.H and Tiegelkamp M, 2001). In this sense, it has provoked a movement to Open Systems in this application field. Thus, the so-called Open PLCs that are open architecture controllers that replace a PLC with a computer, have begun to appear in the market.

Nowadays, most of the PLC vendors are doing a great effort for becoming IEC 61131-3 standard compliant. In fact, this offers great advantages to the control system engineers, as the programming techniques become vendor independent. Notwithstanding this, it is impossible to assure if a PLC programming tool is full IEC 61131 standard compliant.

Efforts are being done by international organizations in order to promote the use of standards of the field. The most important related to the IEC 61131-3 standard is PLCopen (1992), a vendor- and product-independent worldwide association, whose mission is to be the leading association resolving topics related to control programming. Its main goal is to support the use of international standards in this field. PLCopen has several technical and promotional committees (TCs).

In particular, TC6 (TC6, 1992) for XML has defined an open interface between all different kinds of software tools, which provides the ability to transfer the information that is on the screen to other platforms. The eXtensible Markup Language (XML) (W3C, 2006a) was selected for defining the interface format and in April 2004, the first XML schema for the graphical languages was released for comments (W3C, 2004). The goal is to achieve interchange of code in graphical language by expressing what is in the screen in XML, including the graphical information.

On the other hand, TC3 (TC3, 1992) defines a certification system for PLC programming environments, focussing on testing the features of a tool. Thus, PLCopen TC3 certifies IEC 61131-3 environments. It certifies three compliant levels: a Base Level (BL) a Conformity Level (CL) and Reusability Level (RL) that implies possibility of interchange Function and Function Block definition with other RL products. The compliance test is performed by a test laboratory accredited by PLCopen. Therefore, those PLC programming tools having the PLCopen CL could exchange source code.

The work presented here presents the design of a framework for analyzing the grade of conformance of models exported by PLC programming tools. It defines different levels of conformity: Data Types (elementary or extended), the Program Organization Units (POU interface & source code) and the automation project structure. This work extends and modifies the proposal of TC6. The extension consists of including the constraints that both, the elements of software architecture and the elements of the programming languages, must meet to define correct automation projects. The modification consists of eliminating any element or characteristic related to the visualization of code. This is achieved combining XML technologies: XML schema (W3C, 2004), *schematron* rules (Rick J., 2006), XML Stylesheets and the SAX (Simple API of XML) (SAX, 2004) or DOM (Document Object Model) (W3C, 2005) XML Application Program Interfaces. The conformance analyser is based on previous work of authors (Marcos and Estévez, 2005) that focussed on achieving the collaboration, along the development cycle of an application of a set of heterogeneous tools. To do that, the IEC 61131-3 software model was expressed as a XML schema. The current work extends the schema with a set of composition rules to check the constraints to be met.

The layout of the paper is as follows: section 2 briefly describes the elements of the IEC 61131-3 software model. Section 3 presents the definition of the IEC 61131-3 conformance templates. The conformance analysis is divided in different aspects and for each one various levels of conformity have been defined.

2. THE IEC 61131-3 SOFTWARE MODEL

The IEC 6131-3 standard is defines three different parts: Data types, source code in Program Organisation Units (POU) and the Software architecture in terms of the Automation project definition. The following sub-sections analyses the characteristics of each part.

2.1. Data Types

Within the common elements, the ***data types*** are defined. Data typing prevents errors in an early stage. It is used to define the type of any parameter used. This avoids, for instance, dividing a Date by an Integer. Table 1 illustrates the Elementary Data types of the IEC 61131-3 standard.

TABLE 1. IEC 61131-3 ELEMENTARY DATA TYPES

Key Word	DataType
BOOL	Boolean
SINT	Short Integer
INT	Integer
DINT	Double Integer
LINT	Long Integer
USINT	Unsigned Short Integer
UINT	Unsigned Integer
UDINT	Unsigned Double Integer
ULING	Unsigned Long Integer
REAL	Real
LREAL	Long Real
TIME	Duration
DATE	Data
TIME_OF_DAY (TOD)	Time of Day
DATE_AND_TIM (DT)	Data and time of Day
STRING	String
BYTE	Bit string of length 8
WORD	Bit string of length 16
DWORD	Bit string of length 32
LWORD	Bit string of length 64

The standard defines the representation patterns, range of values of each data type (IEC, 2003). It also associates a Key word to each type of data and the space of memory they require. The number of bits needed to define data types are tool dependent.

Programmers can define their own personal data types based on these elementary data types. These are known as ***derived data types*** and they also group structures, arrays and enumeration values.

2.2. Source Code

Part 3 of the IEC 61131 standard specifies the grammar, syntax and semantics of a suite of five programming languages for programmable controllers.

Concretely, two of them are textual and three graphical:

- The *Instruction List* (IL) is a low level textual language, similar to the assembler. It consists of text lines and each line describes an operation instruction.
- The *Structured Text* (ST) is a high level textual language, structured in blocks. It is close to Pascal, and it is influenced by ADA and C programming languages. It is commonly used for complex process automation.

Related to the graphical languages, three are provided by the standard:

- The *Ladder Diagram* (LD), based on graphical symbols laid out in networks in a similar way to a rung of relay ladder logic diagram. It is specially oriented to Boolean signals.
- The *Function Block Diagram* (FBD) is used for complex procedures programmed by graphical objects or blocks which represent functions, function blocks or programs, like in electronic circuit diagrams. It is widely used in the process industry.
- The *Sequential Function Chart* (SFC) focuses on structuring sequential tasks of an automation application through programs and function blocks. It can be programmed in a textual or graphical way.

TABLE 2: IEC 61131-3 POUS

POU Type	Interface	Body
Program	Input/output formal parameters and local variable characterised by name, type and initial values	ST,IL, LD,FBD, SFC
Function Block	Input/output formal parameters and local variables characterised by name, type and initial values	ST,IL, LD,FBD
Function	Input formal parameters, Only an output parameter and local variables. All of them characterised by name, type and initial values	ST,IL, LD,FBD

Furthermore, the IEC 61131-3 standard provides three type of *Program Organization Units* (POU). A POU is composed by the interface, which contains the formal parameters, and the body that contain the source code. Table 2 illustrates the characteristics of each type of POU in terms of formal parameters of the interface, and the language for programming the functionality or body

2.3. Structure of the Automation Project

The elements provided by the standard in order to define the automation project are:

Configurations: for instance they can be a PLC or an OpenPLC. **Resource**: It provides support for program execution. It can be a 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 which are Programs, Function Blocks and Functions and they provide software reuse. They are once programmed and they can be used whenever necessary. This assures modularity and reusability of applications.

Finally, the **Variables** represent the communication between software components. In fact, their visibility identifies the IEC 611313-3 elements that are involved in the communication:

- VAR_ACCESS identifies the communications between programs residing in different configurations.
- VAR_GLOBAL at Configuration level identifies communication between programs residing in different resources of the same configuration.
- VAR_GLOBAL at resource level identifies the communication between programs of the same resource.
- VAR_LOCAL identifies the communication among nested POUs.

As any other high level programming language, they are characterised by their type and initial value. In Estevez et al (2007a) a Component Based Technology is used for identifying the different components and connectors of the IEC 61131-3 software model. Two types of components can be distinguished: those that do not encapsulate code, Configurations, Resources and Tasks, and the Program Organization Unit (POU) that encapsulates code. These latter can be Programs, Function Blocks and Functions.

3. AN IEC 61131-3 CONFORMANCE ANALYSER

The XML schema (W3C, 2004) standard and schematron (Rick, 2006) technologies proposed by The World Wide Web Consortium have been selected for developing the IEC 61131-3 Conformance analyser.

In particular, the *simpleType* and *complexType* mechanisms, provided by W3C schema, have been used for defining in XML the IEC 61131-3 elements, taking into account all their characteristics and making use of *attributes*, *restrictions* and *representation patterns*. Other W3C XML schema elements, such as *sequence* and *choice*, have been used for defining the architectural style of the IEC 61131-3 software model.

On the other hand, and in order to check cross contents, the schema element for defining constraints (*key/keyref*), jointly with schematron rules are used. Fig. 1 illustrates different schemas defines to form the core of the conformance. These technologies have been used for defining the IEC 61131 grammar as a set of XML schemas (See Fig. 1). The conformance analyser uses these schemas in order to perform the different conformity checks.

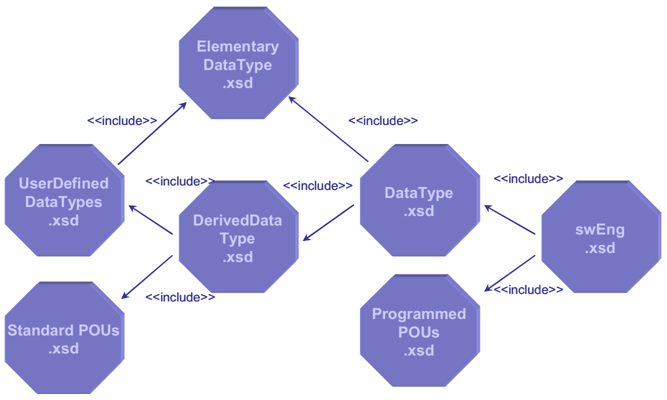


Fig. 1. General scenario of IEC 61131-3 conformance analyzer

The information related to types is defined separately from the automation project itself. The *DataType.xsd* and *ProgrammedPOUs.xsd* XML schemas define in XML the data types and POUs, respectively, which are used in the automation project. In particular, *DataType.xsd* schema includes the characterization of the IEC 61131-3 elementary data types, contained in *ElementaryDataType.xsd* as well as the application derived data types, i.e. new data defined by the programmer. The software architecture defined Markup language is available in (www.disa.bi.ehu.es/gcis/projects/merconidi). The Fig. 2 illustrates the control flow of the conformance analyzer.

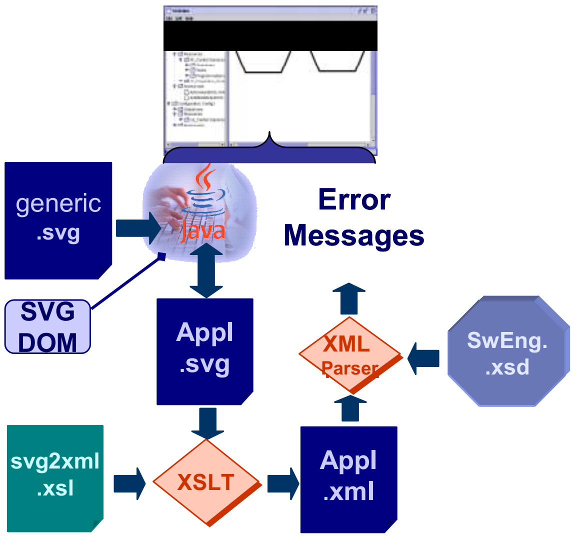


Fig. 2. The control flow of the conformance analyzer

To obtain a model to be analyzer from a tool, the integration techniques proposed in (Estévez, et al., 2007a) can be used.

4. DEFINITION OF CONFORMITY LEVELS

Three levels of conformance have been defined, one for each part of IEC 61131-3 standard identified in the previous section. The following sub-sections detail each level of the conformance analyzer.

4.1. Data Types conformance level

The *DataType.xsd* is responsible for checking the conformance of the PLC programming tool at data type level. It is composed by two XML schemas, one for the elementary data type and other for derived data type (see Fig. 1). *ElementaryDataTypes.xsd* schema contains a XML *simpleType* definition for each elementary data type as defined in the IEC 61131 standard. Thus, it includes the name, the range of values and the representation patterns. As a simple example, Fig. 3 illustrates the definition of SINT data type. As the IEC 61131 standard specifies, it is an integer value with or without sign being its range of values between -128 and 128, both inclusive.

```
<xs:simpleType name="SINT">
<xs:annotation>
        <xs:documentation>SINT: [-128, +127]</xs:documentation>
</xs:annotation>
<xs:restriction base="xs:int">
        <xs:maxInclusive value="+127"/>
        <xs:minInclusive value="-128"/>
        <xs:pattern value="(\-|\+)?\d*"/>
</xs:restriction>
</xs:simpleType>
```

Fig. 3. Characterization of SINT data type

Fig. 4 shows a more complex data type, the TIME type as the standard defines different representation patterns.

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

Fig. 4. Characterization of the TIME data type

As a few tools support them, the conformity of the TIME, DATE, TIME-OF_DATE and DATE-AND-TIME data types correspond to the advanced data type conformance level.

4.2. Source Code Conformance Level

The POU XML schema *(ProgramedPou.xsd* in Fig. 1) characterises in XML the three types of POUs that the IEC 61131-3 standard distinguishes.

Two sub-levels of conformance have been identified at POU level. The basic level is related to the POU interface. The goal of the advanced level is to assure that both, the interface and the source code contained in the body, are IEC 61131-3 compliant.

The three POU types are characterised by the interface and the functionality. The interface of a Function Block or Program is defined by the input and output formal parameters characterised by their name, position and the data type. Each formal parameter is defined in XML as a Variable XML element, and its characteristics are defined as XML attributes. The interface of a Function POU type is different as only one output formal parameter is defined that is the type of the function (see Fig. 5 and Fig. 6).

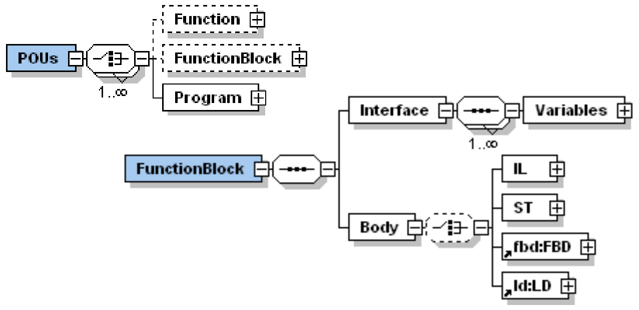


Fig. 5: POU Type definition

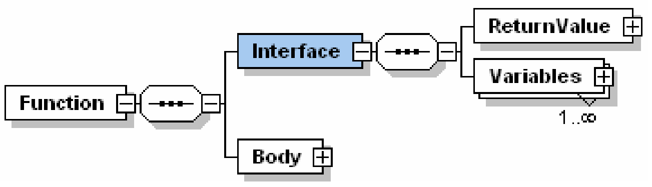


Fig. 6: Function interface

The advanced conformity level includes the analysis of the POU functionality. In order to add this possibility, a XML schema for each of the programming languages provided by the standard has been developed.

Fig. 7 illustrates the XML schema defined for characterizing the Function Block Diagram (FBD) graphical language (Estevez et al, 2007b).

A POU written in FBD language is composed at least by a network defined by an identifier (*id*), and optionally by a label and a comment. Each network could contain blocks, jumps, returns and/or connectors. For instance, each block is defined as a XML schema element characterised by a set of attributes: an identifier (*id*) and the POU from which it is an instance (typeName). If the POU is of FB Type, it is also necessary to indicate the name of the Instance (*instanceName*). Finally, it is also necessary to indicate the block appearance order (*line and position*) within the network.

The parameters (*inputs*, *outputs* and *inOuts*) are characterised by their order and they can be negated or not. The inputs have a connection that can come from a variable, a connector, or from an output of another block. On the other hand, the block outputs could have a connection that updates a variable value.

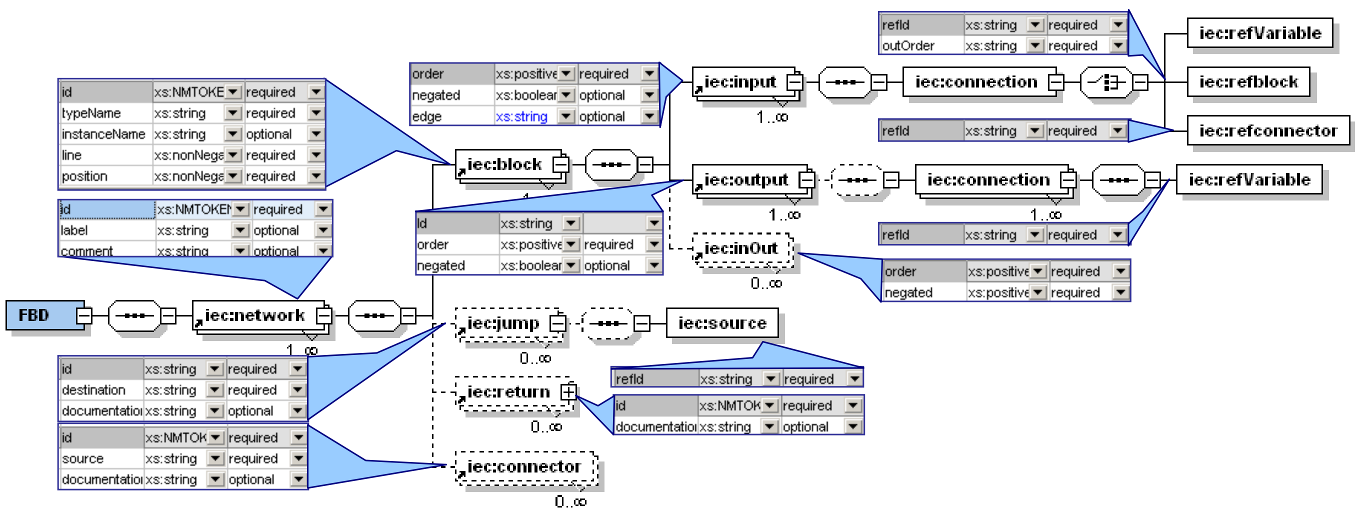


Fig. 7: FBD language definition in XML

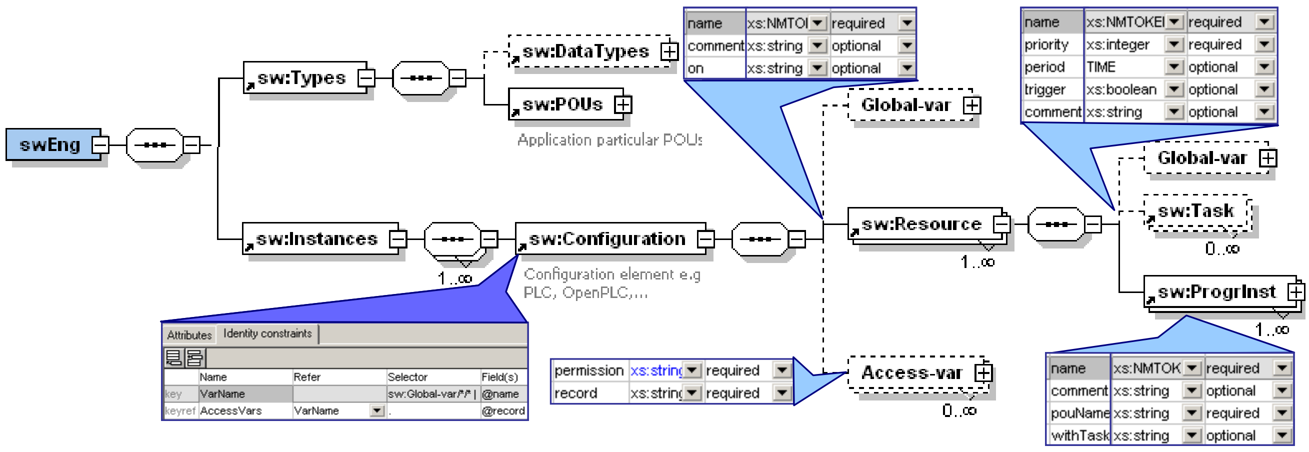


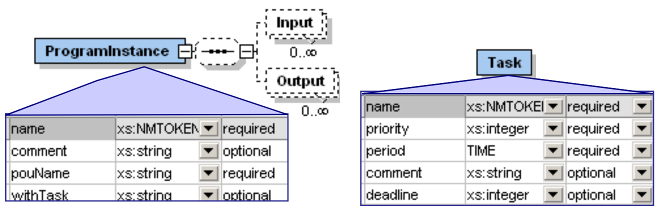
Fig. 8: General scenario of the Software Engineering Markup Language

4.3. Software Architecture conformance Level

The schema form overall software architecture is illustrated in Fig. 8. In particular, the architectural style of the software architecture is contained in the *swEng* XML schema. An Automation project must be composed at least by one configuration, containing one or more resources which contain the source code (*ProgramInstances*) that could be or not organized by tasks.

Each of the program instances contained in a resource correspond to a particular instance of a program Type POU characterised by its Name. If such program instance is organized by a task, it will have the *withTask* attribute with the name of the task.

Finally, the program instance will have actual parameters that must correspond in type and position to its formal parameters. Fig. 9 illustrates the characterization of the program instances and task elements.



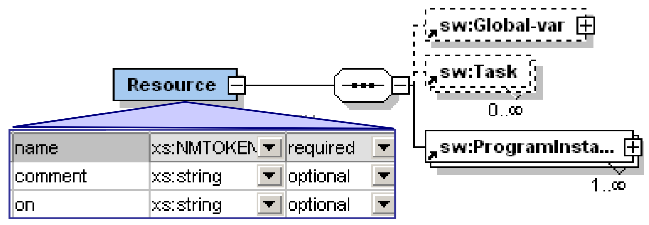
name	xs:NMTOKEN	required
comment	xs:string	optional
pouName	xs:string	required
withTask	xs:string	optional

name	xs:NMTOKEN	required
priority	xs:integer	required
period	TIME	required
comment	xs:string	optional
deadline	xs:integer	optional

Fig. 9: programInstance and Task elements

Task elements can represent a periodic or sporadic execution. In the first case they are characterised by the *period* attribute that must be of type TIME.

In the second case, the task is characterised by the event that triggers it, whose type must be BOOL. The event may be an external or internal interrupt.

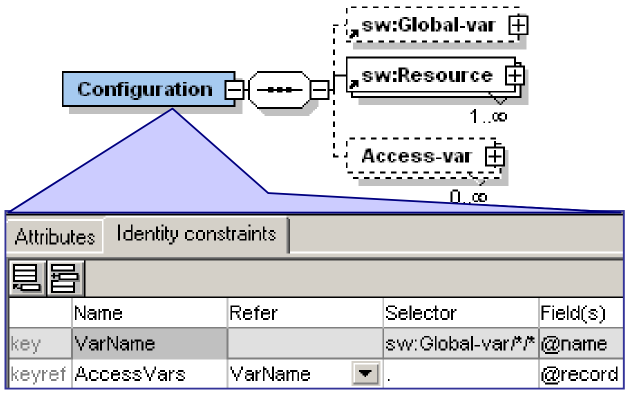


name	xs:NMTOKEN	required
comment	xs:string	optional
on	xs:string	optional

Fig. 10: Resource

The Resource elements could be downloaded to a processor; in this case, the name of this processor must be the value of the *on* attribute (see Fig. 10).

Finally, Fig. 11 illustrates the characterisation of the configuration element. It corresponds to a PLC and it is at least composed by one resource.



	Name	Refer	Selector	Field(s)
key	VarName		sw:Global-var/*/*	@name
keyref	AccessVars	VarName	.	@record

Fig. 11: Configuration

The automation project variables are defined at Configuration and at Resource level, depending on their visibility. Configurations exchange information through Access variables.

Fig. 11 illustrates how by means of the key/keyref mechanism provided by XML schema, it is assured that the access variable corresponds to a global variable defined in this configuration.

5. CONCLUSIONS

A conformance analyser has been proposed that allows assuring the level of conformance to the IEC 61131 standard of a PLC programming tool.

The analyser has been developed using the potentialities of the XML technologies, that allow defining not only the architectural style of the software architecture and programming languages but also to check cross contents. This conformance analyser can be very useful for the final users as it assures that the software developed within a PLC programming tool can or cannot be reusable in others.

6. ACKNOWLEDGEMENTS

This work was financed by the MCYT&FEDER under DPI-2006-4003 and by UPV/EHU under project DIPE 06-16.

REFERENCES

E. Estévez, M. Marcos and D. Orive (2007a). *Automatic Generation of PLC Automation Projects from Component-Based Models*. The International Journal of Advanced Manufacturing Technology. Springer London. Vol: 35, pp: 527:540. 2007

E. Estévez, M. Marcos, D. Orive, E. Irisarri and F. Lopez (2007b). *XML based Visualization of the IEC 61131-3 Graphical Languages*. Proc. of the 5th International Conference on Industrial Informatics, pp: 279-285 (INDIN 2007). Vienna, Austria.

IEC (2003). International Electrotechnical Commision. *IEC International Standard IEC 61131-3 Programmable Controllers. Par3: Programming Languages.*

John, K.H and Tiegelkamp M. (2001). *Programming Industrial Automation Systems.* Springer.

Lewis, R.W. (1998). *Programming Industrial Control Systems using IEC 61131-3.*IEE Control Engineering Series.

M. Marcos, E. Estévez, *"Formal Modelling of Industrial Distributed Control Systems"* 16th IFAC World Congress. Praha.

Medvidovic, N. and Taylor, R.N. (1997). *Exploiting architectural style to develop a family of applications*. IEE Proc. Software Eng. 144 (5–6), pp:237–248.

PLCopen (1992), Web-site: http://www.plcopen.org

Rick J. (2006). *Resource Directory (RDDL) for Schematron 1.5.* Web Site: http://xml.ascc.net/schematron/

SAX (2004). *Simple API of XML (SAX).*
Web Site: http://www.saxproject.org/

TC3 (1992), PLCopen Technical Committee 3.Web-site: http://www.plcopen.org/pages/tc3_certification/

TC6 (1992), PLCopen Technical Committee 6.Web.site: http://www.plcopen.org/pages/tc6_xml/

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

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

W3C (2004). *XML Schema Part 0: Primer (Second Edition), W3C REC-xmlschema-0-20041028.* Available at: http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/

W3C (2005). *Document Object Model (DOM)*,
Web Site http://www.w3.org/DOM/

W3C (2006a). *eXtenslble Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation.* Available at: http://www.w3.org/TR/2006/REC-xml-20060816/

W3C (2006b). Extensible Stylesheet Language (XSL) Version 1.1, W3C Proposed Recommendation PR-xsl11-20061006.