Using Industrial Standards on PLC Programming Learning

F.J. Molina*, J. Barbancho*, C. Leon*, A. Molina*, A. Gomez*
*University of Seville/Department of Electronic Technology, Seville, Spain

Abstract— **In this paper, we review aspects relevant to industrial standards related to PLC programming: IEC 61131, IEC 61499 and a work about safety developed by the PLCOpen organization based on IEC 61508. We propose to use these standards in PLC learning to fix a common know-how that allows one to reduce the gap between industry and education, and between different professionals. We show the application scope of these standards by analyzing the IEC 61131 limits. The IEC 61499 can be introduced in distributed control systems and in complex centralized systems with multiple operating modes. In critical applications, like safety functions, where functional safety is required, the IEC 61508 is a reference model.**

I. INTRODUCTION

In PLC programming, there is a gap between industry and education that has been increasing, especially in the last few years. We identify two main reasons. Traditionally, automation control systems have been developed by engineers or technicians. Their know-how, design methodology and working procedures are compiled into standards. But usually, these standards are quite complex to be introduced in education because they are written to transmit clear information to experts, not to teach to non-experienced students. The second reason is the increasing use and the integration, of programmable electronic systems (PES) and computers in all the levels of the automation hierarchy: sensor/actuator level, supervisory level and the company management tools (databases, information systems, decision tools, etc). Due to the introduction of communications technologies and new programming concepts with PES, professionals from computer science have started working with industrial PES. Their methodology and their knowledge about industrial processes are quite different from traditional engineers and conversely. There is a mutual misunderstanding.

The IEC 61131 standard was a first attempt to give a reference model for industrial PES. It was defined in 1993 and released in 2003. It unifies concepts and proposes a common standardized programming interface to allow people with different backgrounds to create different pieces of a program that can be joined to work together correctly. The standard also defines a set of programming languages and includes an easy way to apply new technologies like communication protocols and fuzzy-logic [1][2][3]. The PLCOpen association is working to update and promote the IEC-61131 standard. Their working groups have developed much material to better understand and teach the standard [4] [5]. They have also been developing recommendations and solutions to many industrial problems such as safety and motion control. The study of these works offers a clear perspective of the programming methodology that the standard draws. Currently, the IEC 61131 has been successfully introduced in the industry. Many commercial tools are compliant or, at least, include their main concepts. Nevertheless, several studies show that the languages and the execution model, defined in the standard, are incomplete. Because of this, frameworks usually define additional language elements or characteristic, and non-compliant program execution models [6] [7].

In section II, we analyze the main concepts of IEC 61131-3 standard. In certain applications, special reliability and availability are required for long periods of time (e.g. safety in process and machinery). Programming restrictions and different methodologies are necessary to achieve that [8] [9] . IEC 61508 standard introduces the main concepts on functional safety, recommendations and methods for programmable electronic devices in safety applications. In the section III, we present a work done by PLCOpen organization to adapt IEC 61131-3 to IEC 61508 recommendations.

The application of the IEC 61131 also has limitations. The study of these limitations allows us to introduce the standard IEC 61499. This new standard is better suited to program decentralized control with highly-coupled applications distributed on several devices [14] [15] . It is also adequate when the process and the controller have multiple operation modes and operating states. A comparative study of both is presented in section IV.

II. THE IEC 61131-3 MAIN CONCEPTS

The IEC 61131 part 3 defines a software model for industrial controllers based on a clear set of definitions about what is a program, how to construct one, and how the program interacts with the host machine and with another program. The model consists of:

- *High-level elements*: Configurations, Resources and Tasks. They describe the overall architecture of a program resident in a programmable controller.
- *Program Organization Units* - POU's. They are basic code containers. A program is structured in one or more POU's.
- Variables and Data Types.

The standard defines basic hardware-independent data types. The size and the arithmetic are strictly declared. Consequently, many errors, caused when a program is compiled in different platforms, can be avoided. From basic types, programmers can define derived data types, creating enumerations, sub-ranged types, arrays, or data structures.

POU's are the key of the program development based on 61131-3. There are three types: Functions (FUN), Function Blocks (FB) and Programs (PROG). A Function is defined as a program organization unit in which, when executed, yields exactly one data element. It does not contain internal memory. The same arguments always yield the same output.

A function block can process several outputs. It contains internal state information. Each function block instance has a structure with internal data, the inputs or default input values, and output, last output or default output values.

A Program is a function or a FB with access to the I/O variables.

Fig. 1 illustrates function or function blocks elements: interface, internal variables and code.

Two advantages of using blocks can be stood out. Firstly, the interface must be defined exactly and consequently the block operation/behavior. The second is related to the programmer's skills. The code can be written using any of the five languages that the standard defines. Ladder (LD), Functions (FUN) and Sequential Function Chart (SFC) are graphical languages. Instruction List (IL) and Structured Text (ST) are textual languages. The selection of these languages is guessed right because it allows the programmers with different abilities to program easily. Ladder language is widely used. Seventy percent of the programs in PLC's are written in it. It is inspired on relay logic formalism, so it is very popular amongst technicians. Instruction List language is similar to the assembler languages. It is better suited to solve problems that deal with mathematical algorithms, or to process data intensively. It is very popular amongst programmers accustomed to low level languages like embedded systems developers. Structured Text language is similar to Pascal language. It is attractive to computer science programmers, and it is better suited to solve math or algorithmic problems. Despite their importance, SFC and FUN languages are not very popular. In the standard, both languages are used to structure the main program. In fact, the main program is defined as a *logical assembly of programming languages elements*. Figures 2 and 3 illustrate the aspect of a main program structured with both languages

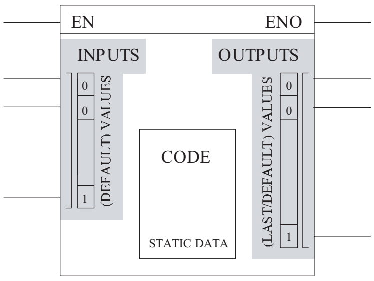

Fig. 1. Function Block elements.

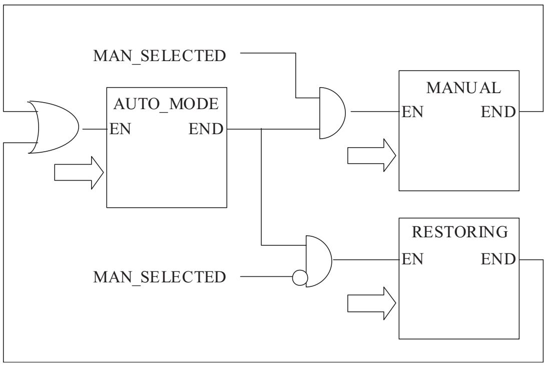

Fig. 2. Main program structured by FB's.

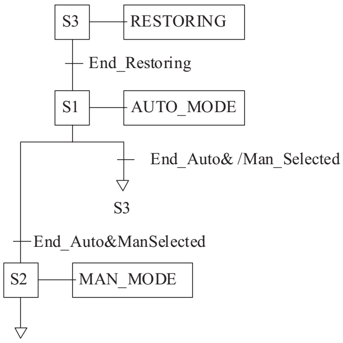

Fig. 3. SFC structured main program.

SFC - Sequential function chart is an evolution of IEC 648 language. But instead of a program documentation resource, SFC is a set of execution control elements for POU's. It is designed to structure sequential and concurrent algorithms. It can be considered a special case of a Petri Net, and it is better suited to describe Discrete Event Systems (DES). Many researchers have developed procedures to program SFCs from DES models [10] [11] [12] [13] .

Function Language is a graphical language in which POU's can be interconnected with in a similar way to an electronic circuit. It structures the program and manages concurrency easily.

The standard also defines high level elements to describe how the program is hosted and executed. The elements are Tasks, Resources and Configurations.

A Task defines the execution mode of program or a POU. Typically, there are three: cyclic execution, periodic, or triggered by an event. In this context an Event is a change in a variable. If this variable is associated to a physical I/O the event is called Alarm. A Configuration represents a programmable controller system as defined in IEC61131-1. A PLC is a configuration example. Another is a computer running a Soft-PLC. A soft-PLC is a specialized software able to run IEC programs. Nowadays, this technology is being applied more and more frequently. A Resource is a real o virtual machine where the programs are executed. Each CPU in a PLC is a resource. A configuration can contain one or more resources. Global user variables can be defined in the resource or a configuration level, and this

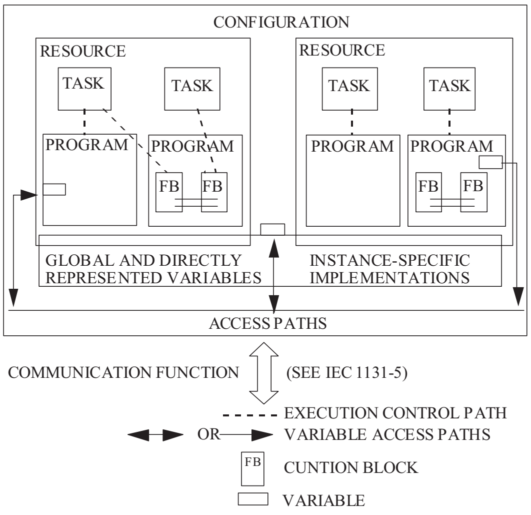

Fig. 4. The IEC 61131-3 software model

will be its scope. Special sort of global variables are Directly Represented Variables and Access paths. Directly Represented Variables are vendor-defined and represent physical I/O channels. Access Paths are variables that can be accessed by other configurations.

The standard software model (fig. 4) describes an overall automation structure consisting of several configurations connected by a communication bus, with programs residing in the resources and working in a coordinated way.

III. Programming restrictions and functional safety of PLC's

Reliability and functional safety are always problems present in an industrial controller. In fact, IEC 61131 defines hardware and software characteristics to achieve this goal. Although, programmers' habits and programming languages are likely sources of failures. Jointly and with a clear methodology, to reduce program failures it is necessary to limit the variability of the languages. Full Variability Languages (FVL) like Pascal, Java or C++, allow to the programmers great freedom to define the program structure, the data and the program flow, so the failure probability is greater compared to Limited Variability Languages (LVL) that are more restricted, and combine predefined and application specific functions. The IEC 61131-3 languages are good examples of LVL's. But the standard also includes additional restrictions to increase reliability, e.g. by fixing the program structure and by limiting the program access to hardware resources directly. Specifically, this late restrictions means:

- The I/O channels are updated through Directly Represented Variables, e.g. the program never read or write the I/O channels.
- The programs are not often compiled to a processor native code program. Instead of that, it is translated to Instruction List or to a pseudo assembler language that runs in a supervised or interpreted mode.

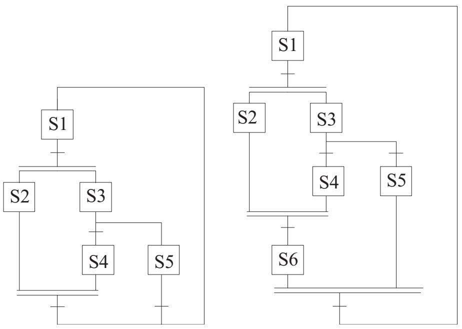

Fig. 5. Unsafe and impossible SFC's

IEC 61131-3 languages have been well studied in several papers, and they are known for their inconsistencies. For example in SFC languages the state evolution can fall into unsafe states or impossible conditions derived from jumps from simultaneous divergences (fig. 5).

Moreover, dynamic problems like critical races or non-deterministic execution time have to be avoided for better reliability, too. Critical races can be present in any language if simultaneous accessing to shared variables or feedbacks are used in FB's. In these cases, the result can depend on the execution order. Non-deterministic execution time or infinite loops can be caused by classical structuring instructions like WHILE or FOR. In consequence, language and execution model restrictions must be stronger in those cases where reliability should be higher. Safety is one of these cases. Safety can be defined as the expectation that a system will not cause anyone bodily harm or risk human life or health. Safety functions in a process or machinery require an extra reliability and availability that can only be achieved with special hardware devices, special PLC systems and/or special programming methodology. Safety in machinery or industrial processes is a very important, sometimes complex problem that can be considered at the design stage. Safety measures have to be included at the beginning of the development process, so the process and the safety elements are well integrated. Safety functions can be managed by standalone elements, wiring safety devices or specialized programmable electronic systems (PES). Complex safety functions are usually managed by safety PLC's, using centralized or distributed structures. Safety functions must be reliable to guarantee that safety measures are functional or to maintain a safe state for the equipment under control. This concept is described in the IEC 61508 standard as Functional Safety of a safety related system. The introduction of this standard is quite important to have a clear idea about how to estimate the risks derived from the failure of a PES, and the Safety Integrity Level (SIL) required to assure the risks are at an acceptable level. Furthermore, IEC 61508 offers a set of measures or recommendations to enhance SIL in a PES. For example:

- Using applicable programming languages and language subsets.
- Using validated software blocks.

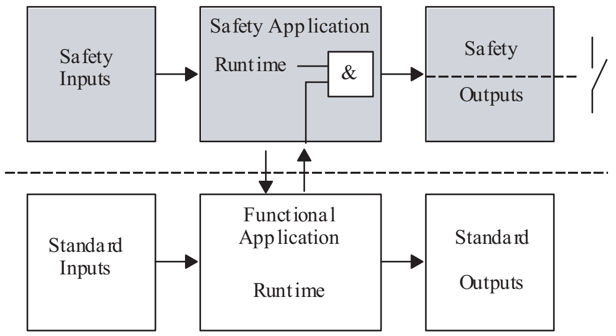

Fig. 6. PLCOpen software model for Safety applications

- Using applicable programming guidelines.
- Using recognized error-reducing measures for the lifecycle of the safety-related software.

More specific standards, derived from IEC 61508 have been proposed in the process industry (IEC 61511), machinery (IEC 62061), nuclear plants (IEC 61513), etc. In these standards the IEC 61508 philosophy is integrated with specific safety measures and functions, specific recommendations and specific failure estimation methods. PLC Open has developed a wide work to include the IEC 61508 and IEC 62061 strategies within the IEC 61131-3 programming languages. The work is organized into four topics:

- A software model.
- A set of recommended reductions in the Development framework.
- General rules for Safety-Related Function Blocks.
- A library of certified Safety Function Blocks.

A. The software model

The software model describes the functional process application and the safety application in a generic way in order to allow that existing and upcoming safety systems can be covered (Fig. 6). No safety control hardware architecture should be excluded by this specification. Both applications can be executed on one device or there could be several devices which are more or less coupled.

The main objective of PLC Open is to merge the developer environment for the functional part and with an integrated safety part, including reductions in language programming and functionality for safety section. This way, safety is integrated with process control functions at the beginning of the development stages. Safety I/Os and safety signal processing are clearly separate from the process I/Os and the functional application. The functional application can read safety inputs, but it can not be connected to the safety outputs directly, it only can control the data flow to them. To achieve this separation, a new data type with the designation SAFEBOOL was defined. SAFEBOOL is not a simple new boolean variable. It can include additional information in order to calculate the SIL with the programming tools. SAFEBOOL represents a single input or output channel, regardless of the internal hardware structure: 1oo1 (*"1 out of 1"*), 1002D, 2oo2 or 2oo3. The hardware which executes the FBs with the SAFEBOOL I/Os has to be certified separately. The safe value of a SAFEBOOL must be false. Application engineers must ensure that the safe behavior when set to FALSE. Additional recommendations have been included to process safety functions, such as: the safety application must runs only as a single task, or it must to have higher priority. A safety functions should not be interrupted by the functional application program.

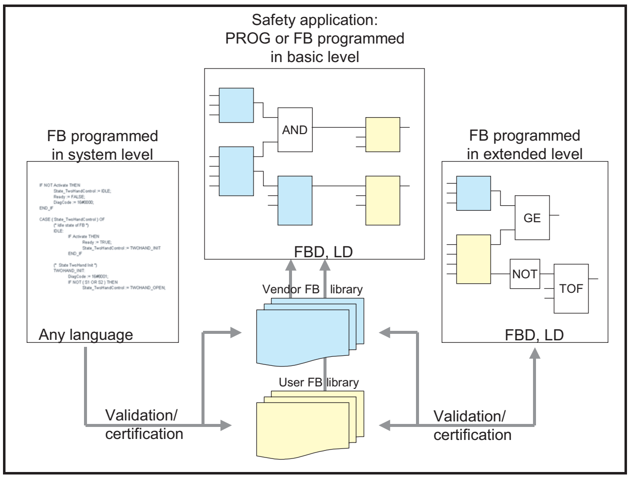

Fig. 7. Safety applications programming procedure

B. Recommended reductions in the development framework

The specifics proposed by PLCOpen for the framework differentiate between three user levels: Basic, Extended and System level. In the Basic Level, the program consists of certified interconnected blocks. The Extended Level allows one to create custom blocks, although they have to be validate/certificated before being used in the basic level. System Level is provided for suppliers of safety controls. The blocks can be programmed in any language, so this level is not part of the specification. The figure 7 illustrates these ideas.

IEC 61508 defines a reduction in the preferred programming languages for different SILs. Based on this, PLOpen has selected in the specification Ladder and Function Block IEC 61131-1 languages for Basic and Extended levels. SFC, Instruction Lists and Structured Text are more complex to test and validate.

Data types, functions and function blocks from the IEC 61131-3 are also reduced. The reduction is stronger in the basic level.

C. Safety related function blocks (SRFB's): General rules and certified library

The PLCOpen safety specification defines a generic SRFB (fig. 8). Specific safety related FB's should be derived from this one. The interface and the behavior of this FB are the following:

- An *Activate* input to enable the safety function.
- *A Reset* that can be used for different purposes: as *"error reset"*, restoring the initial state, or as a *"manual reset"* of a restart interlock by the operator.

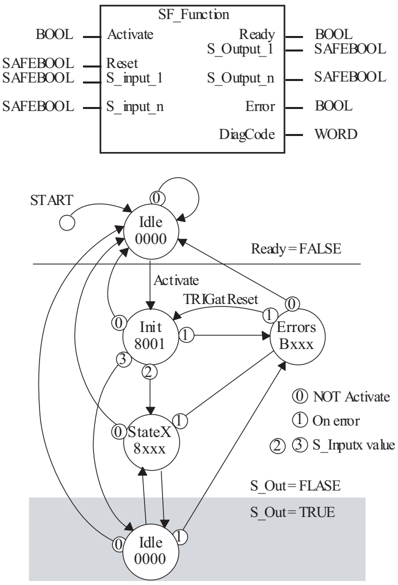

Fig. 8. Interface and behavior of the base SRFB.

- *S_Inputs* (process specific variables).
- *A Ready* output indicates if the FB is activated and the outputs are valid.
- *S_Outputs* (process specific variables)
- *Error* output indicates that the FB is in an error state.
- The *DiagCode* is very useful for debugging. It represents all the states (active, not active and error states).

Following this model, PLC Open has developed a library composed of 20 SFRB's (e.g. emergency stop, safe stop category 1 and 2, mode selector, two hand control, sequential and parallel muting, etc)

IV. The limits of the IEC 61131-3 programming model as an introduction to IEC 61499 standard programming

In complex controllers, the IEC 611313 model presents applicability problems derived from overall architecture model misconceptions, and FB specifications. A controlled system with a high number of control points (I/O channel) does not carry to a complex controller necessarily. In this paper, we refer to controller complexity as a functional complexity. Two aspects contribute to increase this functional complexity:

- Multiple operation modes or running states of the process and the controllers.
- The use of distributed control systems in highly-coupled applications.

The IEC 61131-3 describes a centralized, or *"multi-centralized"*, architecture, i.e. a control system composed by several configurations running different applications each one, but in a coordinated way. In the opposite, IEC 61499 proposes applications hosted and running in several devices. Obviously, Function Blocks running in different devices, within a distributed application, have to be strongly coupled, so it is required to have more sophisticated synchronization methods than IEC 61131-3 defines. E.g., in contrast with the Send/Receive functions or Networked Variables, the IEC 61499 offers Publisher/Subscriber and Client/Server services.

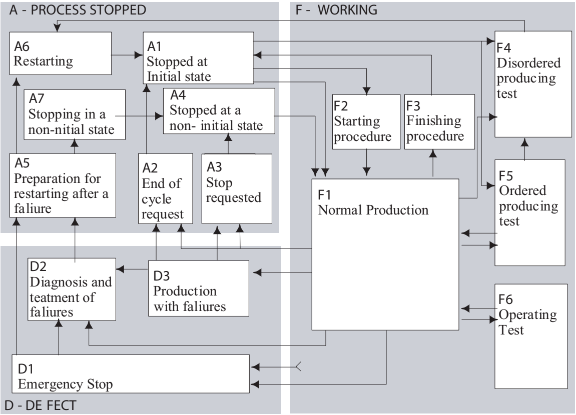

Fig. 9. The GEMMA guide.

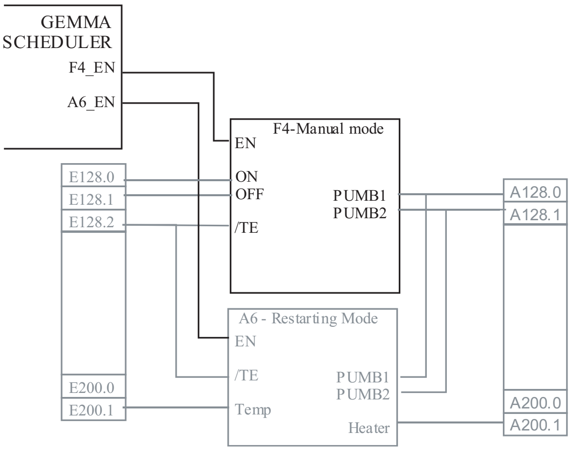

Fig. 10. GEMMA implementation example.

On the other hand, along its operational life, a machine or a process can be placed in many different operating modes and states. A very popular design reference to define them is GEMMA (Guide d'Etude des Modes de Marches et d'Arrts) (fig. 9). GEMMA is a general schedule that describes the process with up to 16 states. Engineers have to decide which states are present or not. Each state is a different automation problem and describes the process in a specific situation. For example: F1- is the normal production mode. F4 represents a manual mode, where some elements can be controlled by an operator's orders, A6 state signals a set of sequentially ordered operations for restarting the process, etc.

Following the traditional structuring methodology, and IEC 61131-3 FB's, each state will be programmed with a different FB. A scheduler FB will call the right FB depending on the operating process state. This can be done by using the EN input of FB's (Fig 10), or using actions in an SFC main program.

When an FB is deactivated it is not executed and stores the process state. When the block is activated again, the

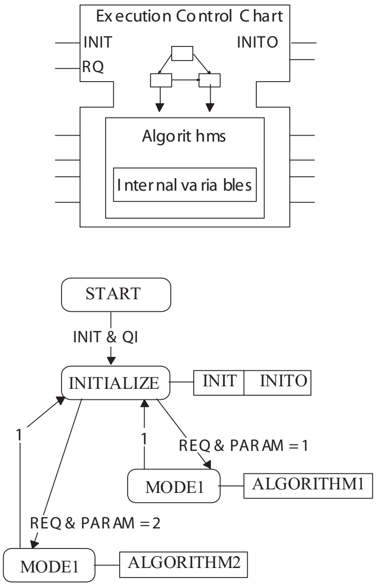

Fig. 11. IEC 61499 Function Blocks: interface and ECC.

real process state will likely not match with the stored one, so the FB's must be restarted. But FB's in the IEC 61131-3 have no special inputs to achieve that. Each vendor defines specific non-standarized inputs to control their execution. In consequence, the programs are not portable and the behavior of FB's can differ from different vendors. The FB's defined in IEC 61499 standard solve both problems. The FB interface makes a clear separation between process inputs and another special, and event triggered ones, called Events. These inputs control the functional state of the FB by means of a user defined Execution Control Chart (Fig. 11).

ECC guarantee the FB behavior, managing the execution and restarting of the algorithms written in it. When comparing ECC and GEMMA is clear that ECC can implement the complete system operation model (e.g. GEMMA) or, at least the relevant part of it. Nowadays, there are few IEC 61499 compliant frameworks. Although many elements like ECC's, can be implemented under certain restrictions with IEC 61131-3 tools [16] [17] (e.g using SFC's). SFC language is an evolution of GRAFCET (IEC-848). It is very usual that commercial tools support vendor specific implementations of GRAFCET orders do not include in SFC standard language. E.g. non-structured hierarchy actions like SET, KILL and FREEZE. With these orders, a master SFC can control the execution of another SFCs. SET order activates states, fixing them till the order is deactivated. KILL order deactivates all the states and actions of an SFC, so it can not keep on running. And, FREEZE pause the SFC evolution and actions execution till the order will be deactivated. Master SFC acts like ECC in IEC 61499 Function Blocks and the slaves like the algorithms.

V. Conclusions

IEC 61131-3 is a reference standard for PLC programming. It is designed to allow that technicians and engineers with different skills can work together. It is successfully introduced in the industry and there are many commercial frameworks. Vendor specific implementation can change, but all the tools have many common elements. That is an advantage in PLC programming learning, jointly with the efforts made by PLCOpen organization to extent the standard and make it grow and understandable. Analyzing the IEC 61131-3 limits, it is possible introduce new standards and new programming methodology. In applications like safety, reliability and availability can increase limiting the language variability. Programming methodology must be integrated with an overall design methodology such as it is described in IEC 61508 standard. Again, in this point, the work realized by PLCOpen is outstanding. IEC 61131-3 is also limited in distributed control systems. In this scope, the new IEC 61499 can be introduced. But, as we have shown, IEC 61499 is also better suited to be applied in centralized control systems, when the process and the controller have many different operating modes and operative stages.

References

[1] International Electrotechnical Commission, "IEC 61131-3. Second Edition" , *IEC publications*, 2003.

[2] R. Lewis, "Programming industrial control systems using IEC 61131-3", *IEE Control Engineering Series*, 1998.

[3] F. Bonfatti, "IEC 1131-3 Programming Methodology" , *CJ International*, France, 1997.

[4] www.plcopen.org

[5] E. van der Wal, "Introduction into IEC 1131-3 and PLCopen" , *The Application of IEC 61131 to Industrial Control*, IEE Colloquium on. 1999.

[6] I. Plaza and C. Medrano, "A specific implementation of IEC 61131-3 software model", *IEEE World Automation Congress*, 2004.

[7] N. Bauer, R.Huuck, B. Lukoschus, S. Engell, "A Unifying Semantics for Sequential Function Charts",*Integration of Software Specification Techniques for Applications in Engineering*, LNCS 3147, 2004, pag 400-418.

[8] Lewis,R., "Can IEC 61131 graphical languages be used for safety related PLC applications?",*IEE - The Application of IEC 61131 in Industrial Control*, 2002.

[9] K. Toon, " IEC 61131-3 in Safety Applications"*IEE - The Application of IEC 61131 in Industrial Control*, 2002.

[10] J. Flochova, " A Petri net based supervisory control implementation",*Systems, Man and Cybernetics, 2003.IEEE Conf.*, 2003.

[11] G. Music, D. Gradisar, D. Matko, "IEC 61131-3 Compliant Control Code Generation from Discrete Event Models",*Intelligent Control. Mediterrean Conference on Control and Automation"*, 2005.

[12] S.Klein, G.Frey; M. Minas, "PLC Programming with Signal Interpreted Petri Nets",*IEEE - Applications and Theory of Petri Nets 2003*, ICATPN, 2003.

[13] D. Thapa, S. Dangol, Wang, "Transformation from Petri Nets Model to Programmable Logic Controller using One-to-One Mapping Technique",*Computational Intelligence for Modelling, Control and Automation*, IEEE - CIMCA, 2005.

[14] G. Frey, T. Hussain, " Modeling techniques for distributed control systems based on the IEC 61499 standard - current approaches and open problems",*Discrete Event Systems, 2006 8th International Workshop on*, IEEE Proc., 2006.

[15] F. Vyatkin, S. Karras, T. Pfeiffer, "Architecture for automation system development based on IEC 61499 standard",*Industrial Informatics*, IEEE Conf., 2005.

[16] L. Ferrarini, M. RomanoC. Veber, " Automatic Generation of AWL Code from IEC 61499 Applications"*Industrial Informatics*, IEEE Conf., 2006.

[17] Isagraf Inc.,"IEC 61499 Execution Model"*www.isagraf.com.*