INCLUDING OBJECT-ORIENTED PROPERTIES IN THE PLC's PROGRAMMING LANGUAGES

Benítez Pina, I., Vazquez Seisdedos, L., Villafruela Loperena, L.

Faculty of Electrical Engineering. University of Oriente C.P. 90900. Santiago de Cuba, Cuba.
E-mail: {Ibenitez Lvazquez lvilla}@fie.uo.edu.cu Telef-fax: 53 (226) 43710

Abstract: If the use of orientation to object is generalized in the industrial automation programming, why not to mix the simplicity of the languages of the IEC 61131-3 with features of orientation to object?

KeyWords: Object-oriented programming, Programmable logic controllers (PLCs), Process Control

1. EVOLUTION OF THE LANGUAGES OF PLCS

The evolution of PLCs programming languages has evolved a similar way as the high level languages. Initially small applications were carried out using lineal programming by means of literal or contact languages (ladder diagram). Then the application field was not only enlarged toward areas of sequential control but also of the process control, economic control and government, passing to the arboreal structured programming with functional modules or organized procedures from a main program.

In France and Japan PLCs languages were standardized to some more diffused types such as Grafcet and MFG (Mark Flow Graph), respectively, but keeping its structured root. The IEC-1131 normalized the design of these equipment internationally but only including structured languages [15]. This norm is in constant evolution and its more recent modification is the inclusion of part 7 on Fuzzy Logic in PLCs [3]. Until the past couple of years, there were two distinct product segments; PLCs and DCSs (distributed control systems). Beginning about 1995 there emerged a third product designed to blend PLCs and DCSs. Moore's APAC's and Foxboro's I/A went through extensive modifications to combine PLC and DCS functionality into a single environment. Fisher-Rosemount introduced Delta V a brand new control system designed to bridge the PLC and DCS void. Each of these systems embraced IEC-1131 standards and allow users to mix languages.

Many articles were presented in international prestige journals in the last few years [1-4, 10, 12-16] demonstrating the importance of this topic at international scientific level. The paper [15] of Mr. Jeremy Pollard (Programmable Controller Support Systems, Barrie, Ont., Canada) was especially pointed out. This article comments the results of a survey carried out in 1995 from the main PLCs designers in the North American market. The lines developed and their perspectives are explained in that moment by each one. A tendency to the "open hardware and firmware" was presented as well as the introduction of Windows environments and the proliferation of graphic PLCs programming languages (Ladder Diagram, Function Block, Sequential Function Chart). These languages and those of text Instruction List and Structured Text, are included in the international standard of PLCs design (IEC-1131-3). All of these demonstrate that introduction of Windows facilities in the PLCs programming is something recent. These also demonstrate that graphic languages dominate the PLCs programming and orientation to object had not been introduced in that year by any international maker of PLCs. Today they are delivering on 1995 vision. Open systems and these general languages are here. But all of them use structured programming propierties. We only find references to the use from orientation to object in STEP7 language from SIEMENS [14], but the classic properties of this method are not fulfilled. This line has a new programming and operation system supported on Windows and with some object-oriented facilities in its programming language, specifically the encapsulation of old functional blocks in "objects" (according to definition of the SIEMENS) [14], but without existing a hierarchy of classes with real facilities of encapsulation, inheritance, polymorphism, etc. Thus STEP7 language doesn't use facilities of those properties hence it doesn't have the object-oriented programming and design advantages, such as safety and reusability of its applications.

The current trend of PLC development was published in the Control Engineering journal at the beginnings of this year [12]: "Trends in PLCs:

0-7803-5670-5/99/$10.00 ©1999 IEEE

· Open Systems.

· Communications.

· Meeting PC competition.

· Increasing analog capabilities". [12]

The work areas indicated in the 1995 show [15] the importance of keeping simplicity and friendliness of PLCs languages and even increasing them, as well as the desire of achieving bigger possibilities not allowed in any language at present. On the other hand, these trends [12] demonstrate that those areas are up to date and even looking for much more possibilities in the competition against PCs. The object-oriented properties are more effective to achieve these purposes than structured properties.

The IEC 1131-3 includes all the common use operands in PLCs. In their section 2.2 (External Representation of the data) it states that this representation will consist in numeric literal (integer and real), time literal and literal of chains of characters. Starting from it the ISaGRAF system (compatible IEC 1131-3) of CJ International are gathered in four basic types: Boolean, Analog, Temporized and Message. The norm as well as the ISaGRAF state the following programming languages:

· LD: Ladder Diagram.

· IL: Instruction List.

· FBD: Function Block Diagram.

· ST: Structured text.

· SFC: Sequential Function Chart.

The use of the parametric functional blocks is very useful in these languages to support reusability and extensibility facilities in these structured programs. These are not more than sections of the program the user can create or receive precompiled in order to execute common operations in applications such as: treatments of analog signals, control algorithms, etc. All of these use a memory area to transfer parameters and signals as well as results, allowing its use under different application situations.

In spite of their potentialities, the parametric functional blocks stay inside the characteristic limitations of structured programming, since they can not thoroughly fulfill the reusability and extensibility requirements, and the modeling and integration facilities as the object-oriented programming does perform.

According to Meyer [11] there are five fundamental requirements to obtain an appropriate structure of reusable components. These are the ones that admit:

1. Variation in types.
2. Variation in structures of data and algorithms.
3. Related routines.
4. Independence of the representation.
5. Union inside the subgroups.

Let's evaluate these aspects in PLC structured languages:

1. To vary the types, another functional block should be written or else a very big one with all the types.
2. The functional blocks to be able to admit variations in structures of data and algorithms are programmed with a complex modular structure and many selection switches such as the PID module IP-262 of SIEMENS. This burdens their use greatly.
3. The relationships among the functional blocks don't follow a fixed rule depending much on the programmer.
4. The representation also varies greatly in the programmer's dependence.
5. There is not coherence among groups of functional blocks.

This demonstrates that limitations exist in PLC programming using the current international languages, and that the development of modern automation requires a higher level in PLC programming offering advantages in the competition with PCs.

High reusability and extensibility levels in a PLCs language are factors that help safety and efficiency of the applications developed with the same ones. But nothing can be achieved if these advantage don't combine with other design and programming facilities of the automated system.

Generally speaking a system is wanted when it doesn't diminish reliability and has great simplicity, so that it maintains its popularity like the rest of the PLC systems. It should also be friendly and contribute to eliminate the programmer's fatigue.

The introduction of a new technique always requires the incorporation of positive aspects from the previous ones and at the same time solve the problems that had the old techniques. Hence, the proposed system uses the advantages of the IL language (IEC-1131-3) for programming methods in classes of the first level of abstraction (Nets) as well as the facilities of the ST language (IEC 1131-3) for programming methods of classes on second and third levels (macronets and users programs). All of these classes and objects should fulfill the object-oriented properties.

The facilities to create the macronets library maintain the advantages of the libraries of traditional functional blocks, but enlarging, modifying and using it with the object-oriented resources.

2. FROM THE STRUCTURED LANGUAGES TO THE ORIENTATION TO OBJECTS IN PLCS

Starting from the study of PLC programming languages and generalizing the features of different types of instructions they include, two types of groups

can be defined: the operand group (O) to use this instructions and the function group (operators) (F) executed with those operands. Therefore, the instructions of the PLC languages (figure 1) could be considered as elements of a Cartesian Product F X O formed by the couples of functions and operands (f,o). The group of instructions that execute a binary function is called Digital Net like P1 in figure 1. Similarly the word functions are grouped in Analog Nets (P2 in fig. 1)

Taking into account that:

"Object-oriented programming is an implementation method in which the programs are organized as cooperative collections of objects, each one of which represents an instance of some class, and whose classes are all members of a hierarchy of classes linked by inheritance relationships". [6]

Then it interests us to create that hierarchy of classes starting from the structural generalities of any PLC programming.

Each programming style is also based on its own conceptual structure. The conceptual base of object-oriented design is the "object model" [6] whose elements are: "Abstraction, Encapsulation Modularity, Hierarchy, Typing, Concurrence, Persistence". [6]. These properties should also be present in a true orientation to object on PLCs.

Each user program can be defined as a succession of precedents and consequent groupings (conditionals and actions) with a common objective called "nets" which model the objects of the real world governed in a specific application.

Starting from the study of generalities of the PLCs programming languages and using the facilities of the Theory of Groups we arrive to an efficient partition (Figures 1) of the group of these nets in which each partition has an equivalence class (RD to Digital Nets, RA to Analog Nets).

Then, if we have classes of programming nets representing groups of functions to carry out in the PLC having their distinctive properties and being disjoints. A mathematical base is given for a basic structure of classes for an object-oriented programming in PLCs languages where the own disjunction assures the encapsulation property.

In the ALS 4.0 language a hierarchy of classes exists (figures 2) arising from the abstraction of common qualities of any application type with PLCs. In the same way similar partitions are also created in higher levels of abstraction. Therefore the general basic classes (RG, PUG, MRG classes) are defined. The RG class contains the common ' qualities of any programming net models an object of a real process. The PUG class contains the common qualities of any user program and which models the control system of those objects of the real world in a small functional unit of the process (node of distributed control network). The MRG class makes it for the groups of nets that model the objects intervening in the execution of a responsibility in the node of distributed control network.

The figure 2 represents in a summarized way the *different object-oriented properties in* ALS 4.0 language.

Summarizing, we already have three aspects differenting classes of nets are obtained. We can also relate it with the grammar of the language. This conforms a set (A, P, M) representing the class pattern for the creation of programming nets, which will also be extended to the other two levels of orientation to object on the ALS 4.0 language.

In the class patterns terminal symbols, non terminals and production rules of the grammar of the IL language (in nets) and ST language (in MR and PU) are used for the creation of the attributes and methods of theses classes with some adaptations.

To highlight the differences between the structured programming in IL and ST and the use of these same ones an object-oriented language, the production rules of the ALS 4.0 language [5] were elaborated. As an example, the most general rules are given here.

Declaration of a class:

head_of_class ' {' attributes '; ' methods '; ' '} '

Implementation of a method of a net class:

derived_net_class_name'.'method_name'('parameter_list') '
' {' instruction_list '} '

For the methods of the classes of nets:

instruction_list:: = instruction {instruction}

For the MR and PU methods:

sentences_list:: = sentences '; ' { sentences '; '}

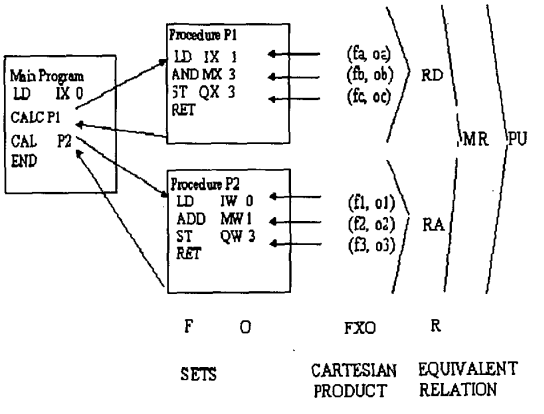

Figure 1. From the structured languages to the object's orientation in PLCs

Summarizing the necessary theoretical details are defined to begin the creation of the system that supports these conclusions.

Therefore, starting from those details a SALS 4.0 software package [5] was created allowing the programming of a cuban PLC (NOVA of the ICID line in Havana City). A low level interpreter is also included to execute on the PLC microcontroller the intermediate code which is received from this package. Both the SALS 4.0 package and the low level interpreter fulfill the object oriented properties.

The ALS 4.0 package has text and graphic editor choosen according to the user's preference. The text editor uses the whole syntax indicated in the production rules. The graphic editor uses only the groups of rules to edit the class methods; the rest (declaration and implementation of classes) are carried out by means of simple dialogue boxes. This is part of the facilities for a low level user.

The example in figure 2 indicates how you can create a general class that governs any motor (RMotor), and how it can be inherited modifying only the necessary aspects to adapt it to pumps, fans, etc. This class is available for any future modification as a general class in the head file (*.HAL). The motor control program (created in a macronet) can also be particularized for a specific type as pumps and fans (figure 2) or to be made directly using the RMotor basic class which makes it adaptable to any motor type at the execution time (polymorphism). It is also stored as a reusable MR class and can be used even for motor types, not implemented yet, without needing any reprogrammation of its methods. In the same way, this can also be carried out with analog macronets and nets for control algorithms (PID, Fuzzy, etc). These are the advantages given by inheritance and polymorphism in

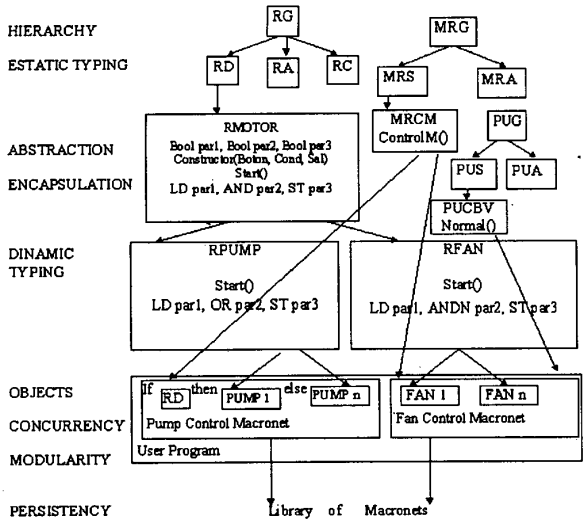

Figure 2. Object-oriented features in ALS 4.

the reusability and extensibility of the PLC applications.

The Fisher-Rosemount DeltaV control system works in similar way, but only to create elements of the library. It doesn't permit change in execution time. For that reason, it doesn't have an important feature of object-oriented programming, the polymorphism. This is the advantage of ALS 4.0.

The ALS 4.0 language is created to allow the advantages from the object-oriented on these PLC. Therefore, as these they are more evident in middle and big applications, it is not easy to show them with simple examples. Nevertheless, they will be summarized in a table the fundamental syntactic aspects of the language and they were explained by means of an example.

As in all object-oriented language there are two essential programmatic aspects:

- Classes definition or classes declaration.
- Classes implementation, that is to say, programming of their methods.

When they are carried out the first programs, the first aspect (definition of classes) it could be considered as something additional that one didn't have in the structured programming on PLCs. However, when one already has a hierarchy of classes with general methods of wide utility in the most usual applications and they are used the facilities for their reuse and extensibility it is seen that it is forever an early work.

The table 1 indicates the syntax for the definition of each one of the types of classes allowed in ALS 4.0.

TYPES OF CLASSES	DEFINITION OF CLASSES	IMPLEMENTATION OF CLASSES

Nets (R)
 	
Attributes ->:
Methods ->:
	Derived_Net : Mother_Net
	{
	Name: Type according to IEC 1131-3;
	NameMethod1();
	}
		Implementation of methods:
		Derived_Net. NameMethods1
		{
		Programming in IL (1131-3)
		}

Macronets (MR)
 	
Attributes ->:
Methods ->:
	Derived_MR: Mother_MR
	{
	Name: Type according to IEC 1131-3;
	OBJD ObjectName : NetName;
	NameMethod1();
	}
		Implementation of methods:
		Derived_MR. NameMeth
		{
		Programming in ST (1131-3) with calls to member objects' methods (only nets)
		}

Programs User (PU)
 	
Attributes ->:
Methods ->:
	Derived_PU: Mother_PU
	{
	Name: Type according to IEC 1131-3;
	OBJD ObjectName : NetName;
	OBJD ObjectName : MRName;
	NameMethod1();
	}
		Implementación de métodos:
		Derived_MR. NameMethod1
		{
		Programming in ST (1131-3) with calls to member objects' methods (Rs and MRs)
		}

Table 1. Syntactic summary of the ALS 4.0 language.

All take the name of the new class (where the first letter (R,M,P) it corresponds to the types classification) followed by ":" and the class name of which is inherited. This way internally the facilities for the operation like classes of that specified type are inherited. For the above-mentioned to create untied classes of some of these three hierarchical trees isn't allowed.

The class body is formed by the attributes (types of internal data of the class) and methods (functions to execute with this data).

In the case of the classes of nets, the possibility to use single types of attributes of the related ones in the IEC 1131-3 is inherited, that is to say, it can write himself a name anyone that represents to a fact boolean, byte or word, entrance or exit (digital or analogical), etc. Also, the possibility to elaborate methods using the syntactic rules of the language IL of this norm, for the manipulation of those attributes is inherited.

The simplified example of the creation of the class of Reserves Net is given in the figure 3. This class would alow to watch over a failure in the motor. If this failure happens and the change to reserve is authorized, the signal to use the reserve motor is activated. For this it has the attributes Failure, Authoriz and Persist of the type Boolean. It also uses the IL instructions for the programming of their method ActivatesRes().

In the case of the macronets classes the possibility to use types of attributes of the related ones in the IEC 1131-3 is inherited, but to create objects of nets as new attributes to manipulate for the methods of that macronet are also allowed. Besides, the possibility to

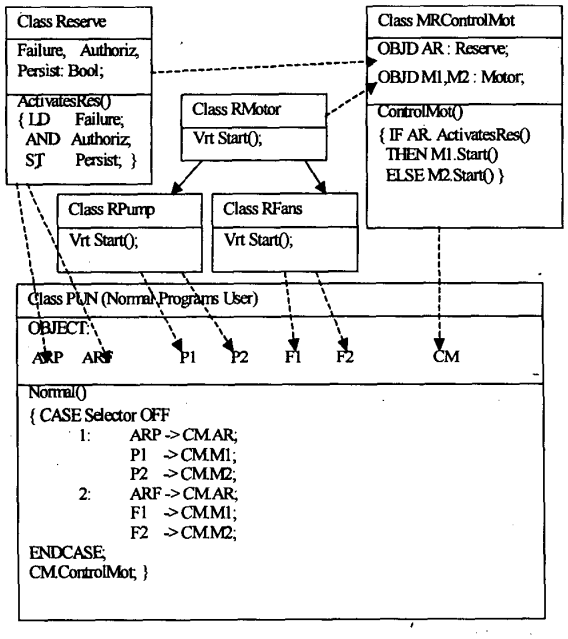

Figure 3. Polymorphism in control of working of motors with fault tolerant for physical redundancy.

elaborate its methods using the syntactic rules of the ST language of this norm, for the manipulation of all those attributes is inherited. The declaration of attributes of the object types implies an extension of the syntactic rules of the ST language. It permits the use of the call to methods of those object members inside the expressions and instructions that are used in alternative or iteration function of this language. This gives big facilities for the encapsulation of operations characteristic of certain type of nets in those object members.

In the example of the figure 3 four types of classes of nets are created: the explained Reserve and the RMotor class, of which RPump and RFans are derived. This allows to group the common qualities to all the motors in the mother and alone to program the different details in the derived. To look for simplicity of the example they don't stand out all the attributes and methods of Rmotor, alone the Starts method is indicated, declared as virtual in RMotor and on-written in the derived classes. All use the rules of IL in their programming.

Also in this example a class of macronet (MRControlMot) is create. This class uses objects of the class Reserve and the RMotor mother class. The ControlMot() method is programmed with ST language but using the methods of the members objects inside a sentence IF.

In the Programs User class, it is indicated in the table 1 the additional possibility to use objects of macronets, that which is not allowed inside the classes of MRs. Taking advantage this, the example of figure 3 indicates the attributes of objects of each one of these classes that uses the Normal() method of the PUN class. Using the CASE sentence of the ST language combined with assignments of objects derived those of the mother (used in CM), you can obtain the use of the same program of the MRControlMot class for pumps and for fans, and it can even be continued using in a future for any others types of motors which will be added in the hierarchy of nets of the RMotor class. This demonstrates the advantages from the polymorphism when re-using programs proven for new applications.

All is programmed in a general way in classes inheritable and instance inside of other applications, guaranteeing this way the advantages of this programming type.

To carry out a comparison between the traditional languages and those object oriented in PLCs, this same example will be analyzed in structured programming. For it, one of the traditional languages is used (for example IL) for all the programming. The reusability is only achieved using the functional blocks with variable parameters. For the case of activation of the reserve, it would be a similar programming of the ActivatesRes() FB, but it doesn't encapsulate their attributes in each object but rather constantly it is necessary to change the parameters that are passed for the calculation with the same FB. In the case of the hierarchy of Rmotor, to create the own hierarchy would not be possible, but

rather they would be independent FBs for the Pumps and Fans starts up or a FB which was able to include all the types of motors that it would be longer and more interdependent. It would be also impossible the creation of a program as ControlMot() which that is equal for all the existent types and that it is expandable for the future types.

The other variant internationally used in the programming of PLCs is the tendency to the use of the super-languages (Java, C++). This variant has the inconvenience that it requires the user to have knowledge on this super-languages, that which is not of the domain of most of the medium users of PLCs. Also, preferably they are used to work the PCs like PLCs (e.g. SoftPLC), because in the case of the PLCs it can not guarantee that they are executed oriented to object. For that reason they become simple translators to structured programming in the case of wanting to apply them to the PLCs.

In the ALS 4.0, the interpreters of high and low level are modified, to achieve as much the programming as the execution of the object-oriented user program. The interpreter of high level (compiler) gives a different structure from the intermediate file (figure 4), where they group the blocks of programs according to the classes to that they belong. Also, the interpreter of low level is able to work on this structure and to execute the methods of the classes with the data of the object activated in that moment. This reduces the volume by memory occupied in the user program for big applications. The increases of the execution time are worthless in front of the big advantages already explained, since alone you increase in 2μs the time of execution of each instruction of IL and in 15μs every time that an object is activated (approximately each 20 or more instructions IL). This refers to a PLC with a micro-controller of medium capacity, for a more potent micro-controller they can decrease more these times, although in all ways the advantages that introduce the properties to object orientated justify it thoroughly.

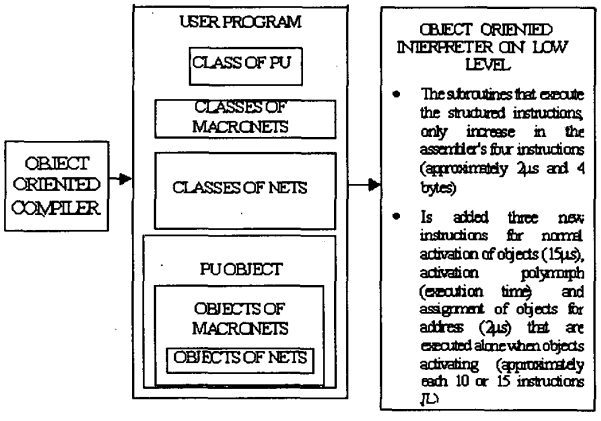

Figure 4. Modifications of the intermediate code and interpreter of low level.

CONCLUSIONS

The PLCs programming is designed to the possibilities of a medium user of this equipment. Hence object-oriented qualities can not be created in the same way established for higher languages like the C++, Java, Eiffel. However they should fulfill the theoretical principles that govern them.

In ALS 4.0 programming language a hierarchy of classes is set in three big trees which include the areas of application of this equipment. Its data and functions are encapsulated with three degrees of visibility (private, protected and public). The dynamic pass of types in execution time is allowed (polymorphism). Persistence is guaranteed outside the creation place and separated from ancestors using the macronets library. Nevertheless the whole normalized syntax of the structured languages of the IEC 61131-3 is almost kept.

Only the use of object-oriented advantages adapted to the simple level of the standard PLCs programming languages will permit the wide diffusion of their features among the PLCs users. ALS 4.0 achieve this combination. Future will say the last word.

REFERENCES

[1] Åström, K.J., Kheir, N.A., Auslander D., Cheok K.C., Franklin G.F., Masten M. and Rabins M.: "Control Systems Engineering Education". Automatica, Vol 32, No 2. pp 147-166. 1996.

[2] Achatz, R.: "Component for systems". SIMATIC report international 2/96 pp 7-9.. SIEMENS. Germany. 1996.

[3] Bartos, F. "Artificial intelligence: smart thinking for complex control". Control Eng, pp. 44-52. Jul. 1997.

[4] Besold, R.: "SIMATIC WinCC is Here". Revista SIMATIC report international 1/96. pp 18-19. SIEMENS. Germany. 1996.

[5] Benítez, I. "Manual de Usuario y Guía del Programador del ALS 4.0". Center for Automation Research. Electrical Engineering Faculty. Univ. of Oriente. 1998.

[6] Booch, G.: "Object Oriented Design with application" The Benjamin Cummming Publishing Company Inc. Redwood City. California, USA. 1991.

[7] Carmichael, A.: "Object Development Methods". SIIGS Books Inc. NewYork. 1994.

[8] Clayton, W. "Object Oriented Programming with Borland C++ 4". QUE Corporation. USA 1994.

[9] Guerraoui, R. "Strategic research directions in object-oriented programming". ACM Workshop on Strategic Directions in Computing. MIT, Jun. 1996.

[10] Harrold, D. "Object-oriented functional specs are key to surprise-free automation". Control Eng, 79-82. Jun. 1998

[11] Meyer, B. "Object-oriented software construction". 1th Edition. Prentice Hall. 1988.

[12] Morris, H. "PLCs aren't just older, they're better". Control Engineering, pp. 113-128. Feb. 1998.

[13] Montague, J. "Lulls are briefly slowing some controls markets". Control Engineering, p. 13 Aug. 1998.

[14] "Operation couldn't be easier". SIMATIC report international 2/96. pp 10-11. SIEMENS. Ge. 1996.

[15] Pollard, J. "The future of PLC programming". Control Engineering, pp. 83-88 Feb. 1995.

[16] VanDoren, V.J. "Advanced control software goes beyond PID". Control Engineering, pp. 73-78. Jan. 1998.