Proceedings of the 5th WSEAS Int. Conf. on Power Systems and Electromagnetic Compatibility, Corfu, Greece, August 23-25, 2005 (pp319-324)

Object Oriented Programming of Logic Controllers (OOPLC)

José Roger Folch, Juan Pérez Cruz, Manuel Pineda Sánchez, Rubén Puche Panadero, Emilio Aragó Ramos
Department of Electrical Engineering
Polytechnic University of Valencia Camino de Vera s/n 46022 Valencia
SPAIN

Abstract: -Object Oriented Programming of Logic Controller (OOPLC) is a new approach for developing control software for Programmable Logic Controllers. It is based upon the capabilities that the IEC-61131-3 standard offers, such as Function Blocks (FBs), extending them to support advanced OO features like inheritance, polymorphism and generic components. This extension is performed using the construction of electrical cabinets as a guide, considered here as a paradigm of application of the object oriented techniques to the industrial world.

*Key-Words:*Industrial control, Knowledge engineering, Manufacturing automation, Mechatronics, Programming environments, Software, Standardization.

1 Introduction

The development of Control Software for Programmable Logic Controllers (PLC) is a crucial stage in the construction of modern automation installations. Old programming languages have had a negative impact in this process: programs tend to be obscure, with limited capabilities of flow control, and internal interdependencies that make any minor modification a nightmare. To solve this problem, a new standard has been established, the IEC-61131-3. It defines new blocks that can be used as the building units of the new programs: Function Blocks (FBs), Program Units (POUs), Tasks, etc. It brings some of the benefits of Object Oriented Programming (OOP), as encapsulation. Nevertheless, other major benefits of OOP, like inheritance, generic programming and polymorphism are not supported.

Initiatives like Open kNowledge Economy in Intelligent industrial Automation (OOONEIDA) [1] are currently applying OOP concepts to the development of PLC software. In this paper, a new framework called Object Oriented Programming of Logic Controller (OOPLC) is presented, with built in object oriented mechanisms. It is based upon the similarities between the process of assembling electrical control cabinets and PLC control programs, so that, in the end, building a new program could be seen as a process of installing and connecting pre-built, fully tested and reliable software modules.

The structure of this paper is as follows: in section 1 an introduction is presented. Section 2 establishes the main OOP features that a control PLC program should implement, and in section 3 the OOPLC framework, which implement these features, is presented. In section 4 the main conclusions are established.

2 Object Oriented Programming of PLC control programs

PLC is a key component in modern automation installations. From the point of view of its hardware components, its evolution has followed the trends depicted in Fig. 1:

- CPUs more advanced, due to the use of better microprocessors.
- Decentralization. The CPU doesn't control only direct inputs and outputs signals. It now communicates also with external intelligent modules via industrial buses
- Networking. PLCs are not isolated parts of the plant. They must perform not only control tasks, but also they must communicate with the factory information infrastructure, for quality control, just-in-time production, etc.

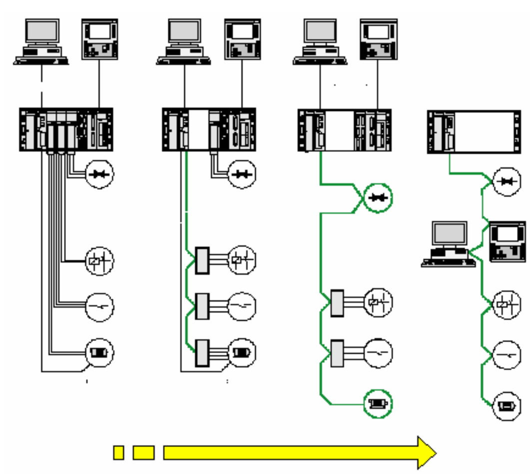


Fig. 1. Trends in PLC hardware development: better CPUs, decentralization and networking.

From the point of view of the software, there have been also significant developments, which has become IEC standards, or have been widely adopted by open vendor associations:

- IEC 61131-3 has defined advanced languages and common elements to program the PLC, independently from its brand [3],[5]. The standard enables two ways of developing a PLC program: top down and bottom up [8]. Either the whole application is specified and divided it into sub parts (declaring the variables, and so on), or the application is programmed starting at the bottom (for instance via derived functions and function blocks).

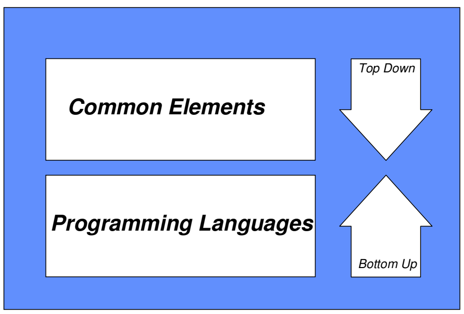


Fig 2. Common elements and Programming Languages defined in IEC 61131-3

- IEC 61149 is a standard about the development of distributed software, especially in the field of inter-operability of devices from different vendors [4].
- OPC (Ole for Process Control) has emerged as a de-facto standard, widely adopted for constructing "software networks", which enable the exchange of data between different vendor PLC's and commercial computer software like SCADA, databases, electronic worksheets, etc.

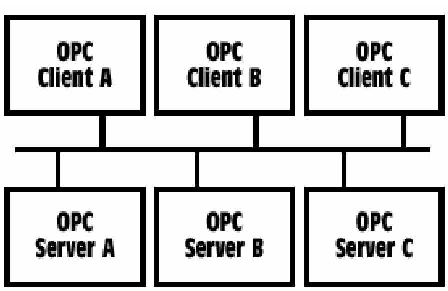


Fig. 3. OPC as a "software network".

These standards offer very powerful tools for developing new PLC programs. Nevertheless, a more general framework is needed to automate the development of the new software, as a way to achieve faster production rates, to ensure its reliability and to facilitate the maintenance of the installation. This situation is very similar to the one that the PC software industry had to face in the past:

- At first, the lack of programming languages slowed down the whole software industry. Only low level languages, such assembly language, were available, and they were specific of each microprocessor brand.
- In a second stage, high level language programs, such as Pascal, C, Basic were introduced, which offered powerful, unified and common resources for developing software for any brand of microprocessor. This is similar to the present situation in the PLC industry.
- But another significant advance came in the field of the PC software, which has contribute to the development of the internet, and the rich multimedia, powerful programs that we have nowadays: Object Oriented Programming (OOP).

With OOP technology, complex programs are developed using pre-built, fully tested software components. Its key features are:

- Encapsulation. Data and the algorithms needed to process it are encapsulated inside the same object.
- Inheritance. New objects can be derived for existing ones.
- Polymorphism. Different objects can implement the same methods.
- Generic programming. The use of generic objects and algorithms allows the compilers to generate functions and objects based on predefined templates.

Current standards in the field of PLC software development, such as IEC 61131-3, offer a limited support for these features. For example,

- The FB definition establishes a clear distinction between the interface and the program body, and permits having internal data encapsulated within the FB object.
- There is a limited support of polymorphism, in the sense that some functions can accept parameters of type "any" (a sum function, for example, can add integer and real numbers).
- There is no support neither for inheritance nor generic programming.

3 OOPLC

OOPLC is a new framework developed to enable the application of OOP technology to the construction of PLC software objects: polymorphism, inheritance and generic programming. The model for this framework is the construction of electrical control cabinets. So, every object of the OOPLC framework, defined in accordance with the IEC 61131 standard, will correspond to a physical one.

2.1 Polymorphism: the conductor object

Polymorphism is achieved using a new data-type, the conductor , defined as follows:

```
TYPE conductor:
STRUCT
b:BOOL; n:INT; re:REAL; t:TIME;
END_STRUCT
END_TYPE
```

Fig. 4. Definition of the conductor data-type.

The conductor data-type has a direct physical counterpart. A physical copper conductor, for example, can carry data of different types: digital, analog, even data frames multiplexed in the time or frequency domain. In the same way, the conductor object, as defined in fig. 4, can carry any type of the elemental data defined in the IEC 61131-3 standard.

There is a drawback in this definition: the size of a conductor structure is the sum of the size of each elemental data type that it can hold. The solution to this problem could be the addition of a new data structure to the standard, included in modern OOP languages such as C++: the “union” type, whose size is only the largest size of any of its elements.

2.2 Inheritance: the relay object

The relay object is defined as a FB with a fixed interface: three input conductors and an output one

```
FUNCTION_BLOCK relay_base
VAR_INPUT
      i,i1,i2:conductor;
END_VAR
VAR_OUTPUT
      q: conductor;
END_VAR
      ;
END_FUNCTION_BLOCK
```

Fig. 5. Definition of the relay_base object

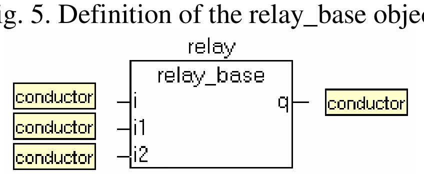


Fig. 6. Graphical representation of a relay_base object.

The program body of the relay_base object is empty. Any derived relay inherits this object, and therefore has the same interface, using the following three-step process, which constitutes the inheritance mechanism in OOPLC:

1. - Duplicate the relay_base component with the name of the new relay (Inheritance).
2.- Add the necessary components as internal variables of the new relay (Data encapsulation)
3.- Program the desired functionality in the body of the new relay. (Method encapsulation)

Fig. 7 shows a relay with a TON functionality generated using this mechanism.

```
FUNCTION_BLOCK ton_relay
VAR_INPUT
      i,i1,i2:conductor;
END_VAR
VAR_OUTPUT
      q:conductor;
END_VAR
VAR
      timer: TON;
END_VAR
      timer(IN:=i.b, PT:=i1.t, q=>q.b )
END_FUNCTION_BLOCK
```

Fig. 7. Definition of the relay_ton object. Bold font marks the additions to the relay_base object.

There is one major drawback with this method: any modification of the relay_base is not propagated to the inherited relays, as in OOP languages like C++. This could be achieved if any future amendment to the standard incorporates a definition of the inherited objects in a way similar to the represented in Fig. 8

```
FUNCTION_BLOCK ton_relay: relay_base
VAR
      timer: TON;
END_VAR
      timer(IN:=i.b,PT:=i1.t,q=>q.b);
END_FUNCTION_BLOCK
```

Fig. 8. Construction of an inherited object without having to repeat the common interface.

Having a common interface, any new relay, derived from the relay_base, can be substituted by another one, without needing to change any connections.

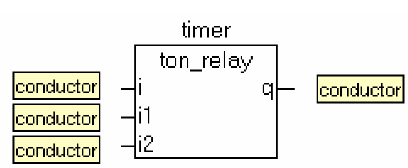


Fig. 9. Graphical representation of a relay_ton object.

Proceedings of the 5th WSEAS Int. Conf. on Power Systems and Electromagnetic Compatibility, Corfu, Greece, August 23-25, 2005 (pp319-324)

2.3 Generic programming: the card object

Connections of different elements inside an electrical control cabinet are made with conductors, but these are normally grouped, forming cables. This technique helps to speed up the building of new systems and to reduce connection-related bugs.

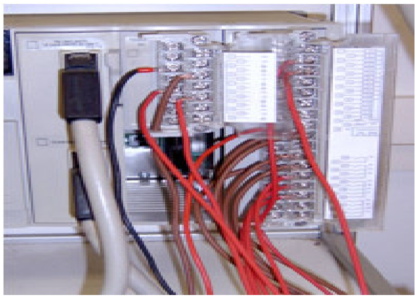
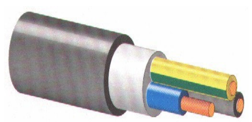


Fig. 10. PLC cards connected using separate conductors (right) and cables with connectors (left)

A new data type is defined for representing the cable object. There are two possibilities:

1. To represent a cable as a collection of different conductors



```
TYPE cable : STRUCT
yellow, blue, black: conductor;
END_STRUCT END_TYPE
```

Fig. 11. A cable as a union of conductors

2. To represent a cable as an array of similar conductors, numbered by its relative position.

```
TYPE cable :
        ARRAY [0..9] OF
        conductor;
END_TYPE
```

Fig. 12. A cable as an array of conductors

In OOPLC, the second option has been chosen because it facilitates the use of loop instructions.

In the same way as single conductors are grouped forming cables, single relays can be assembled together to form a higher level unit, the card object. This is the system that modern PLC's hardware use to achieve a high degree of modularity: they are assembled connecting different cards: CPU, I/O, etc.

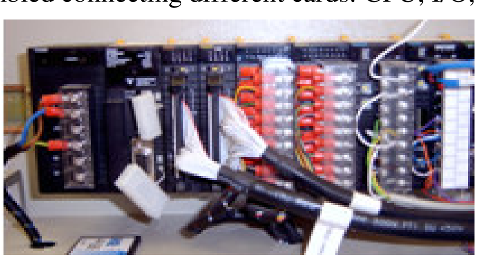


Fig. 13. A modular PLC assembled with cards

In OOPLC, the card object is defined as in Fig. 14

```
FUNCTION_BLOCK generic_card
VAR
   socket: ARRAY[0..9] OF relay_base;
   k: INT;
END_VAR
VAR_INPUT   ci,ci1,ci2:cable; END_VAR
VAR_OUTPUT  cq:cable;         END_VAR

FOR k:=0 TO 9 DO
socket[k](i:=ci[k],i1:=ci1[k],i2:=ci2[k]
         ,q=>cq[k] );
END_FOR
END_FUNCTION_BLOCK
```

Fig. 14. Definition of a generic card object

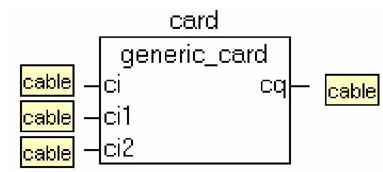


Fig. 15. Graphical representation of a generic_card

A generic_card has ten sockets of type relay_base, where any derived relay can be plugged, because all of them share the same interface. The program body of the generic card connects the conductors of the input/output cables to each of the sockets.

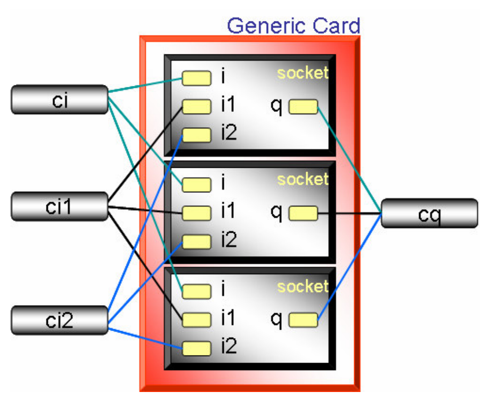


Fig. 16. Connection of the input/output cables to the internal relay sockets in a generic card

A card with a given functionality can be derived from this generic one simply by plugging in the sockets a relay with the desired functionality, through the following two-step process, which constitutes the generic mechanism in OOPLC:

1.- Duplicate the generic_card component with the name of the new card.
2.- Change the name of the relay object plugged into the socket

Proceedings of the 5th WSEAS Int. Conf. on Power Systems and Electromagnetic Compatibility, Corfu, Greece, August 23-25, 2005 (pp319-324)

Fig. 17 shows the application of the generic mechanism to the generation of a card with TON relays
.

```
FUNCTION_BLOCK ton_card
VAR_INPUT   ci,ci1,ci2:cable;  END_VAR
VAR_OUTPUT  cq:cable;          END_VAR
VAR
   socket: ARRAY[0..9] OF relay_ton;
   k: INT;
END_VAR

FOR k:=0 TO 9 DO
socket[k](i:=ci[k],i1:=ci1[k],
          i2:=ci2[k],q=>cq[k] );
END_FOR
END_FUNCTION_BLOCK
```

Fig. 17. Card with ton relays derived from the generic card. In bold face are marked the names that change from Fig. 14.

Because only the names of the elements are changed (marked in bold face both in Fig. 14 and 17), no modification must be made to the body program of the function. This minimizes the possibility of bugs.

There is one major drawback with this method: a card must be derived for each of the relays defined within a project, so the number of elements can grow very quickly. In OOP languages like C++ there is a special construction, called template, that allows the compiler to create "on the fly" objects derived from a generic one. If such construction could be applied to IEC-61131-3 elements, in a way similar to that shown in Fig. 18, it would greatly simplify the development of complex projects.

```
Template <T> FUNCTION_BLOCK generic_card
VAR
      socket: ARRAY[0..9] OF T;
                    ...
```

Fig. 18. Proposal of a Template constructor to support generic programming

With such an extension, the creation of a new card like the ton_card shown in Fig. 17 would be made internally by the compiler, from a simple declaration like the one presented in Fig. 19

```
VAR
 ton_card1,ton_card2:  generic_card <ton_relay>
                    . . .
```

Fig. 19. Construction of new cards from a generic one with the proposed Template constructor.

2.4 Specialization: the connector object

The card object allow for a high degree of parallelization of PLC programs. There are many applications with repeated elements that can be very effectively treated with this technique. For example, control programs in machines like injection molding ones or numerical control machines normally have to control three or more axis. Digital inputs and outputs can also be grouped to apply filtering, etc. in a parallel way. In these cases, one card can perform the same operation in parallel with up to ten different relays, so the development of the program is made in a faster way than having to program individually each element. The electrical equivalent is the unifilar representation of electrical circuits, in which each connection represents several parallel conductors.

Nevertheless, as the input and output signals must be connected individually, a new element is needed to assemble and disassemble them into a cable object. Its physical counterpart is the screw connector used to feed the control signals of a machine to and from the electrical cabinet, as the one shown in Fig. 20

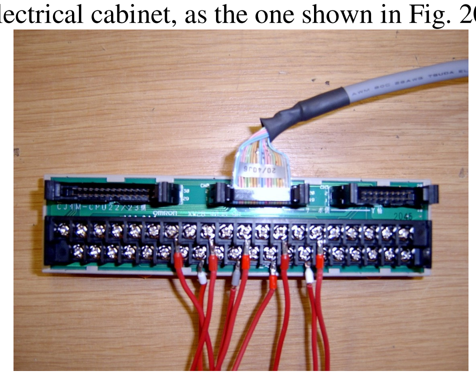
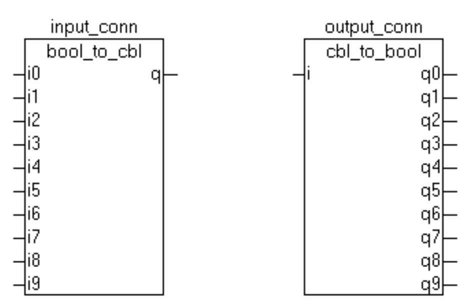


Fig. 20. Connector board for connecting several input/output signals to a single cable

The connector object is defined to perform such an operation. Fig. 21 shows the graphical representation of an input connector and an output one, both unspecialized in the sense that the identifiers of their terminal are general ones.



(a) (b)

Fig. 21. Graphical representation of an input (a) and an output (b) unspecialized connector object

The problem with this representation is that it doesn't offer enough information about the signals to be connected. Besides, in many occasions only few of the available pins are used. Physical connector pins are normally labeled to improve the legibility of the system. To mimic this behavior, a new, more specialized connector object can be derived for a specific application, with a two-step process which constitutes the specialization mechanism in OOPLC:

1.- Generate a new connector object with the desired interface : elementary data types with application identifiers and a cable.
2.- Add an unspecialized connection object and connect it to the new interface

Fig. 22 shows an input connector for an application which controls a three axis machine, which has been constructed following the described specialization mechanism.

```
FUNCTION_BLOCK  bool_to_cbl_3axis
VAR_INPUT
        axis1,axis2,axis3:BOOL;
END_VAR
VAR_OUTPUT
        q:cbl;
END_VAR
VAR
        conn:bool_a_cbl;
END_VAR
        conn(i0:=axis1,i1:=axis2,i2:=axis3 );
END_FUNCTION_BLOCK
```

Fig. 22. Input connector of type bool_to_cbl specialized for an application which controls a three axis machine

This specialized connector can be used repeatedly for connecting the program to the externa process inputs and outputs. For example, in Fig. 23, one element is used for connecting digital inputs coming from the edge switches of the axis, and another one for the command push buttons.

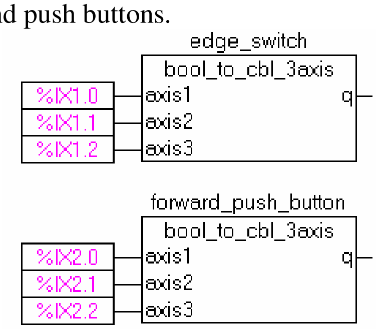


Fig.23. Specialized connectors used for a 3 axis machine

3 Conclusion

In this paper a new framework for developing PLC control programs has been presented. It is based upon Object Oriented techniques, implemented using IEC-61131-3 function blocks, through three basic mechanisms for deriving new objects:

- Inheritance: deriving a new object form a base one and adding a new program, without modifying its interface.
- Generic programming: copying a new object from a generic one and changing the type of the internal generic element, without modifying its program.
- Specialization: generating a new object with the desired interface and connecting it to a base object in its program

New extension to the standard has been proposed to implement more effectively these OOP features.

The development of the OOPLC framework has been made using a as model the electrical cabinet construction process, which has led to the definition of the conductor, cable, relay, card and connector objects. Its use allows for added benefits when using this framework:

- Polymorphism, achieved through the use of the conductor data type, that can carry signals of different types.
- Parallelism, integrated in the definition of the cable data structure and the card object, which allows for a simplified layout of the program and a reduce development time.

The OOPLC framework has been successfully applied to control complex machines such as molding injection, three axis controllers, etc.

References:

[1] Vyatkin, V.V. Christensen, J.H. Lastra, J.L.M., *OOONEIDA: An Open, Object-Oriented Knowledge Economy*, IEEE Transactions on Industrial Informatics, Vol I, No I, 2005, pp. 4-17
[2] Deen, S.M., *Agent-Based manufacturing*, Ed. Springer, 1998.
[3] Lewis, R. W.,*Programing industrial control systems using IEC 1131-3. Revised edition*, Ed. IEE, 1998.
[4] Lewis, R.W.,*Modeling control systems using IEC 61499*, Ed. IEE, 2001.
[5] John, K.H., Tiegelkamp, M., *IEC 61161-3: Programing industrial automation systems*, Ed. Springer, 2001.
[6] http://www.industry.net
[7] http://www.softwaretoolbox.com
[8] http://www.plcopen.org