2001 IEEE/ASME International Conference on
Advanced Intelligent Mechatronics Proceedings
8–12 July 2001 • Como, Italy

Object–Oriented approach to PLC software design for a manufacture machinery using IEC 61131-3 Norm Languages

Marcello Bonfè and Cesare Fantuzzi

University of Ferrara, v. Saragat 1, 44100 Ferrara - Italy.
E-mail: {mbonfe,cfantuzzi}@ing.unife.it

Abstract—
This paper presents an application of object–oriented methodology to the development of Programmable Logic Controller (PLC) programs. PLC is widely used as computer controller of manufacturing machinery, principally because its robustness and programming simplicity.

During the last years, IEC 61131-3 norm has been introduced aiming to provide standard languages and structure to the development environments of PLC programmes, that are instead strictly bound up to now to proprietary languages.

Following IEC 61131-3 prescription, is it also possible to develop well structured, object–oriented control software, which was hardly possible with the former vendor–specific low–level languages.

This paper describes an application of the novel standard IEC 61131-3 to the development of the control software of a medium complexity manufacturing machinery. Software development methodology is reported and comparison regarding development time, software re–usability and application structuring, is then discussed.

Paper for Industry Oriented Session "Theory and application of IEC 61131-3 Standard to PLC programming: a mechatronic approach".

I. Introduction

The design of manufacturing machineries is a task that requires an accurate knowledge of several kind of technologies, including mechanical engineering, electrical engineering, control systems and computer science, as these kind of systems are composed by mechanical parts for product handling and by several electrical sensors and actuators for motion control and overall supervision, performed by digital processing devices.

Most of the times, projects for new machines can hardly take advantage of previous work on similar systems, especially with regards to control software. This is mainly because software developing tools for industrial programmable controllers, generally PLCs, are very primitive, especially when compared to those designed for high level languages such as C/C++ or Java. As matter of fact, industry standard PLCs rely on low level programming languages (ladder logic, assembly code) for control program implementation, with evident limits with to software structuring and modularization, not mentioning lack of parameterizable procedures. Moreover, the development tools are fully vendor dependent and migration of the software application from one hardware platform to another is virtually impossible.

In the last years, the demand for new languages and tools for software development raised dramatically among PLC users, that is why the International Electrotechnical Committee published a normative for PLC programming languages standardization called IEC 61131-3 [1]. This norm introduces software engineering principles applicable to PLC programming, that have some interesting features in terms of software modularization and re–usability. An important issue to discuss is how to use these new languages and structures from IEC 61131-3 for improving the current PLC software development and maintenance cycle.

So far, theoretical analysis and control design of manufacturing machinery has been the object of several academic studies, mainly concerning discrete–event system and their description via Finite State Machines or Petri Nets [2]. However, those approaches have been seldom applied in practice, mainly because it is difficult to derive an accurate model of the manufacture machinery, which must includes, among others, alarms management and asynchronous manual operations.

In this paper, the focus is instead on how to introduce the object–oriented philosophy to the programming of industrial controller (PLC), taking advantage of IEC 61131-3 basics, so that the methodology described can be platform independent. Object-oriented (O-O) applications in industrial software have been reported by several authors, among which we can cite [3], [4], [5].

0-7803-6736-7/01/$10.00 © 2001 IEEE

A leading common theme between these works is the use of O-O methodology in system modeling and designing phase, that leads to a very powerful description of the physical devices in which the system can be split. Unfortunately, the description so obtained is hard to translate in a PLC–based environment, without introducing limitations or loss of effectiveness or without developing a specific intermediate tool between the high–level objects model and the low–level programming environment for the controlling device chosen.

On IEC 61131-3 compatible devices, instead, a programming structure named "Functional Block" (FB), can be exploited to develop control software with an O-O approach. In fact, Function Blocks permit to define generic software modules which, correctly programmed, can be used to implement "mechatronic objects", meaning with this term a combination of hardware (electrical and mechanical part of the machinery) and its control software running on the PLC. Mechatronic objects can then be developed independently from the main program, possibly forming an object library, that results in increasing application maintainability and software reuse. The development of mechatronic objects requires the structural decomposition of the machinery in small parts, each of them related to a particular action or product manipulation. The paper describes a methodology developed for the definition of mechatronic objects starting from a structural decomposition of machinery hardware as well as the use of Function Block structure for mechatronic objects implementation. The paper introduces also an application of such methodology to the control software development for a foodstuff freezing machinery, discussing benefits and possible pitfalls to avoid.

In the next Section, a complete definition of what is the meaning of mechatronic objects is introduced, while section III concerns with the methodology used to isolate mechatronic objects in a complex manufacturing machinery and how those can be implemented using IEC 61131-3 constructs. Section IV describes an example of the application of the above methodology. Concluding remarks are addressed in Section V.

II. Object Oriented approach to Mechatronic control design

The main difference between the traditional process of program construction, more likely procedure–oriented, and the O-O paradigm is a shift in the focus point of the development process, that is resumed in Meyer's Object Motto: "ask not first what the system does, but what it does it to" [6]. In a Personal Computer application, so far the most common playground for O-O programming, this statement refers mainly to data and data structures. Following this motto, in the O-O approach a data structure is only considered together with all the operations that can be executed over it, and this tighten combination is encapsulated in a generalizable software structure called "class". A class can be used in a program by means of its specific instances, named "objects". So an object encapsulate code and variables that all have a common functionality. Only objects proper operations, "methods" in the O-O terminology, can manipulate objects variables, so specific "interface methods" must be considered in the definition of a class, to allow interactions with other objects and the rest of the program.

Programs developed via classes definitions are intrinsically modular. Moreover, thanks to identification of generically useful classes, a high level of abstraction is achieved. As a final result, a great part of software developed is fully reusable.

The O-O approach is applicable not only to programming tasks, that require languages with particular facilities[1], but also to modeling and analysis of physical complex systems, as is reported in [7] and [8] For this purpose, we will focus on a powerful formalism named Unified Modeling Language (UML [9]), which is a graphical O-O language that integrates concepts derived from several object modeling and analysis techniques. With UML, classes of a O-O program and relationships between them can be very well described, and it also has features to define accurately objects behaviour and sequences of operations, with an implementation independent syntax. Thanks to these peculiarities UML has been applied successfully to analysis of real-time systems[10], which requires exactly this kind of unambiguous specifications.

UML can be used, of course, to describe also manufacturing systems and their real-time control needs, with particular regards to operational features of physical devices. Guidelines given in [10] are helpful for the identification of objects and their relationships. In manufacturing machines, main candidate classes are sensors, actuators and control algorithms. As an example, Figure 1 shows a UML model for a simple and generic digital control application. Lines between objects in the diagram shows different kinds of relationships, like *association* between controller,

[1] first of all inheritance of operations and attributes between classes.

sensors and actuators, *aggregation* for A/D converter contained in sensors, and *generalization*, that means an object is a specification of another one and it inherits from the latter operations and attributes.

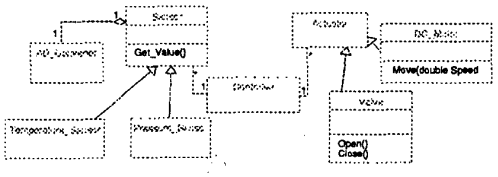

Fig. 1. UML model for a generic control application.

Once defined the objects structure, their behavior must be analyzed and described. Object behaviour can be simple (the object performs services on request without memory of previous services), continuous (e.g. digital filters or PID regulators) or state-driven. In last case, it is described in UML by means of Harel State-charts [11], as is shown in Figure 2. It is worth to note that this formalism allow to describe reactions of objects, to internal or external events, that don't depend only on the state, but also on transition's target.

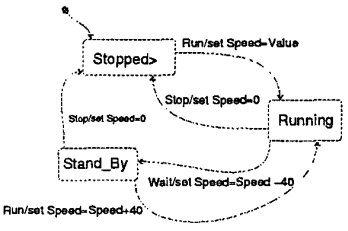

Fig. 2. An example of state-driven behaviour description with UML Statechart

When considering PLC–based control of manufacturing machines, a further abstraction is needed. In fact, PLCs operative system take charge of some operations modeled in Figure 1 so that they are transparent for the programmer (e.g. `Get_Value()` operations for every kind of sensor). This means that it would be more effective to consider objects as sets of several physical devices (mechanical parts, sensors, actuators) that together perform specific functionalities in the manufacturing process. The mechanical-electronic whole together with its directly related control software is what we refer to as a **mechatronic object**. In a complex manufacturing system, after analysis of both physical structures and functional areas, several "mechatronic objects" can be identified and isolated (e.g. conveyors, hydraulic bars, "pick and place" robots, etc.), some of which can be decomposed again in lower–level objects, cooperating in a nested architecture.

Finally, recalling the parallelism with O-O programming in Personal Computer application, the control algorithm can be viewed as a *method* that operate on the physical part of the object (sensors, actuators), storing the physical state (position, operations already executed etc.) into a specific portion of controller memory. It is evident that objects software components (algorithm and data memory) must be isolated from the rest of the program and encapsulated in a software structure, so that if a mechatronic object with the same features is needed for another machine project, or in another part of the same machine, the software block can be easily reused.

III. IEC 61131-3 FEATURES AND MECHATRONIC OBJECTS

The implementation aspects concerning control software for mechatronic objects described above, regard mainly the modularization and encapsulation facilities that can be used in the programming tools. Another important matter is the necessity to reuse or, at least, adapt software parts developed for previous projects on different vendors PLC platforms.

Unfortunately, major Programmable Controller producers has so far developed proprietary languages and programming tools with poor support to software reuse. For example, symbolic and parametric programming is not always possible, as instead is required by IEC 61131-3 Standard for programmable controllers [1]. Another great improvement coming from the IEC Norm is the syntactical definition of four different programming languages, with the aim to reduce dramatically the number of industrial controllers dialects. However, for the purpose of this paper, the most important aspect to discuss is the IEC 61131-3 "Common Elements", in which concepts derived from software engineering principles can be found, with the aim to describe an efficient and modular software model of control applications, with strong facilities for code reuse.

The IEC 61131-3 software model allows the decomposition of the application into independent modules that are defined *Program Organization Units* (POUs). POUs are divided into:

- *Programs*: the larger container of all programming elements, variables and constructs necessary for the control of a machine or a process.
- *Function Blocks*: software elements defined by two parts: (i) a **data set**, which describes the in-

put/output parameters that permit connections to other POUs (its external interface) and internal variables that can maintain their value between two subsequent execution cycles (this allows storing the state of function block elaboration); (ii) an **algorithm**, which is the code executed every time the function block is called and elaborates the data set described above. A Function Block can be programmed in any of the IEC languages, also using calls to other Function Blocks, that results in a strong hierarchical decomposition.

• *Functions*: software elements analogous to Function Blocks, but returning a single output value and without permanent internal variables, so that they can't maintain an internal state and can perform only services without memory.

The parameterized and generic definition of POUs is the major point indicated in the IEC norm for software reuse. For example, a program may contain several instances of the same Function Block (FB) type, each one independent from the others, because it is executed upon different variables and its internal data are allocated in a reserved memory area. Furthermore, the same FB can be used in several different programs, reducing time spent on rewriting code tested for a preceding project. This means that a growing library of user–defined software blocks can be constructed as programs end their "on–the–field" test phase and show effectiveness.

Another important part of IEC 61131-3 common elements is the definition of a formalism named **Sequential Function Chart (SFC)**, which is a state diagram derived from Grafcet and Petri Nets[2]. SFC can be used both to describe state-driven behaviour of machine parts and for Programs and Function Blocks programming[2], expressing state actions and transitions in any of IEC languages.

The software component appointed to implement a mechatronic objects model according to definitions in section II is of course the FB. This one is in fact the structure that can encapsulate the functional control of the behavior (continuous or state-driven) of a physical object. Its data section permits to keep memory of preceding executions, so the operative state of the object is well identifiable. Input and output parameters can represent signals (from sensors and to actuators) needed by the algorithm of the FB to control the object it is related to. Furthermore, input and output parameters are also the interface of the object with the rest of the program. This means that some parameters represent interlocks and relationships with other objects, that means, in this mechatronic approach, the whole of machine

Another important aspect of implementing a machine control program with IEC 61131-3 software model is related to its nested architecture, that suggests a top–down approach to the designing phase. In practice, the system is decomposed step by step until elemental devices are recognized. For a manufacturing machine these steps could be determined by the following decomposition levels:

(i) **Cell**: this level corresponds to a work-cell in which, for example, raw materials are processed or transported by a belt system between two neighboring work-cell.

(ii) **Unit**: it is a subset of devices in a cell which are related to a particular function. For example, considering a filling system for a can machine, the set of devices (actuators and sensors) used by the filling control.

(iii) **Device**: the bottom level corresponding to "atomic" devices, provided directly "off–the–shelf" by manufacturers, even if composed themselves by mechatronic components (e.g. pneumatic actuators with embedded end-stroke sensors).

Once determined the definition of the nested mechatronic components, the external interface of related FBs is a map of input and output signals connected with the specific machine part plus interactions with other FBs. The FBs internal algorithm is then programmed, that could be done with the help of user–defined libraries of FBs related to standard machine components defined in previous projects. This means that the designing and programming process for control application results in a combination of a top–down and a bottom–up approach. The first one is more oriented to the complexity management of the high–level abstract view of the project, while the second is suitable to deal with the low–level implementation phase, making easier modules maintainability and reuse. Figure 3 shows the projecting steps on the left and their results on the right. Vertical arrows should be followed according to the top–down or bottom–up approach chosen.

IV. A PRACTICAL EXAMPLE

The programming methodology described in previous section has been applied on several projects, for machines with different peculiarities. The application described in the following refers to a foodstuff freezing system, whose schematic layout is shown in Figure 4. The system is con-

[2]Functions cannot have a permanent internal state

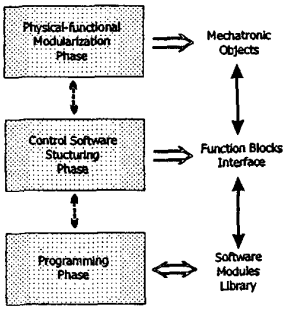

Fig. 3. The projecting methodology oriented to mechatronic objects.

trolled by a Siemens S7-400 PLC, and its control program has been developed on Siemens Step7, a software tool quite compatible with IEC 61131-3.

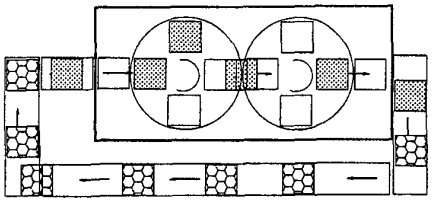

Fig. 4. Schematic layout of freezing machine analyzed

The machine working cycle can be resumed as follows: products are placed on trays that move along stepping conveyors, which realize a route around the freezing cell (drawn as a big rectangle). Inside the freezing cell, trays are stacked and pushed onto circular buffers, the two great circles inside the freezer. Trays remain on each buffer half of the total freezing time and then are pushed out of the second buffer, unstacked and transferred on a conveyor where products are taken from the following machine. Empty trays move along the conveyors line outside the freezing cell, going back to the loading cell where they are reused for another freezing cycle.

The machine has parts very similar to each other, e.g. conveyors, buffers, stacker and unstacker, and pushing bars that transfers tray stacks from the stacker to the first buffer, from the first to the second buffer and from the latter to the unstacker. Of course, the control software realized take advantage of similarities between mechatronic components. For example, every stepping conveyor is controlled by a different instance of the same FB type. Moreover, some simple and generic components, like conveyors or hydraulic actuators control, are suitable to build a project-independent library. It is instead difficult to export higher level objects, e.g. "cell" objects, to different projects, because they are constructed according to architectural peculiarities of each project.

From the functional point of view, the most interesting part of the machine is the freezing cell, with particular regards to the management of information needed to keep trace of the number of trays on each stack[3], buffer sectors occupied, elapsed freezing time and so on. For the memorization of these data, a specific structured data type has been defined, together with specific Functions to manipulate it. This structure stores information about the freezing cell total content, divided onto specific locations for every sector of the two buffers. When a stack transfer is performed, opportune Functions are executed to transfer also stack information on the right data structure location. As can be seen, freezing cell data are attributes of the physical object, while transfer operations are its methods.

Another interesting aspect of the application, is controlling of pushing bars, for two reasons. One is their relationship with the freezing cell object. Of course, as follows from physical analysis of the machine, *pushing bar* objects are aggregated to the higher level object *freezing cell*, as is shown in Figure 5. The other peculiarity of these objects is their particular state-driven behaviour. For example, in their *pushing forward* state, state evolution can lead to *stopped* state for emergency reasons, or *backward* state, after transfer task completion. It is only in the latter case that the stack data transfer operation should be done. To comply with this analysis, *pushing bar* FB implementation relied on the textual high–level language named in IEC 61131-3 Structured Text (ST), rather than IEC 61131-3 SFC (even if available on Siemens Step7) which seems at first sight the "natural" Statechart implementation language. Actually, Statechart formalism allow the execution of transition dependent and state dependent operations, while only the latter type is permitted in SFC diagrams.

Statechart realization in ST can be done by means of the selection construct `CASE .. OF` that permits switching to the correct state case and performing only operations and transitions control needed in that particular state. For example, the selection may look like:

```
CASE State OF
 ...
 StateB: Continous_Action();
         IF Transition1
```

[3]some stacks may not be complete, because of human operators loading request.

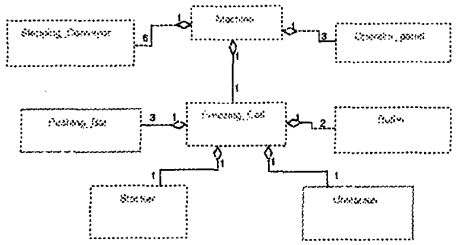

Fig. 5. Object Model of the freezing machine considered

```
            THEN Transition1_Action();
                 Exit_Action();
                 State := StateC;
         ELSIF Transition 2
            THEN Transition2_Action();
                 Exit_Action();
                 State := StateD;
 StateC: ....
 ...
END_CASE
```

This implementation has been preferred also for efficiency reasons to SFC, that, on Step7 tool, require more resources than compiled ST, especially with regards to memory occupation. In fact, the Step7 SFC editor produces FBs with a peculiar internal data structure, independently from of user defined variables, needed by the system to perform correct control of state transitions, computing of state elapsed time and other operations. However, most of these operations are unnecessary in the tested application, so the ST realization, in which only required operations are performed, prevents superfluous waste of computational time and user memory.

Thanks to its modular architecture and the utilization of high level languages and structured data, the readability and, consequently, maintainability of the control software realized is significantly increased with respect to previous realizations, that relied on low level languages and bitwise operations. So far, no upgrades has yet been applied to the application, but it is highly expectable that even different programmers will be able to manage it easily.

One last note regards platform independence. Unfortunately, current technology prevents the direct reuse of software modules on different PLC programming tools, even converting them in a low level textual language, because a lot of instructions are in fact hardware dependent. However, the application model, that means mechatronic objects description via Function Blocks interfaces and functional Statecharts, based on standard formalism, is absolutely independent from a particular implementation, so at least the project skeleton and preliminary documentation are fully reusable. Nevertheless, the programming phase is also faster than it is with traditional methods, because it requires only specific encoding of the standard model.

V. Conclusion

This paper has treated the implementation of the "object–oriented" technology for industrial automation environment, focusing on those aspects that are strictly related to the peculiar characteristics of the software for industrial automation, like the necessity of combining together mechanical and electrical devices, leading to "mechatronic" components. Then, some characteristics of the industry standard IEC 61131-3 have been analyzed with respect to the realization of an object–oriented methodology. In particular we focused on the use of Function Block software structure, that facilitate modularity and code encapsulation, so that a practical software reuse can be realized. The paper ends with a simple, real example showing the effectiveness of the concepts here treated.

References

[1] Iec 61131-3, "Programmable controllers - part 3: Programming languages," 1993.

[2] R. David and H. Alla, *Petri Nets and Grafcet: Tools for modelling discrete events systems.* Prentice–Hall, 1992.

[3] O. Duran and A. Batocchio, "A high–level object–oriented programmable controller programming interface," in *ISIE '94* (I. E. S. IEEE, ed.), pp. 226–230, 1994.

[4] F. Bonfatti, P. Monari, and G. Gadda, "Bridging structural and software design of PLC–based system families," in *ICECCS '95* (IEEE, ed.), pp. 377–384, 1995.

[5] C. Maffezzoni, L. Ferrarini, and E. Carpanzano, "Object–oriented models for advanced automation engineering," *Control Engineering Practice*, vol. 7, no. 8, pp. 957–968, 1999.

[6] B. Meyer, *Object–oriented software construction.* Prentice–Hall, 1997.

[7] R. Joannis and M. Krieger, "Object oriented approach to the specification of manufacturing system," *Computer–Integrated Manufacturing System*, vol. 5, no. 2, pp. 27–35, 1992.

[8] B. Abou-Haider, E. Fernandez, and T. Horton, "An object–oriented methodology for the design of control software for flexible manufacturing systems," in *2nd Workshop on Parallel and Distributed Real–Time Systems*, pp. 144–149, 1994.

[9] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide.* Addison Wesley, 1999.

[10] B. Douglass, *Real–Time UML, developing efficient objects for embedded systems.* Addison Wesley Longman, 1999.

[11] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, pp. 231–274, 1987.