A Design Pattern for translating UML software models into IEC 61131-3 Programming Languages

Cesare Fantuzzi, Francesca Fanfoni and Cristian Secchi
University of Modena and Reggio Emilia
Via Amendola, 2
42122 Reggio Emilia (Italy)
{name.surname}@unimore.it

Marcello Bonfè
University of Ferrara
Via Saragat, 1
44100 Ferrara (Italy)
marcello.bonfe@unife.it

Abstract

The paper deals with a deep analysis and application of object-oriented methodologies for the design process of industrial machine controllers. The process passes through the phases of (1) conceptual model development of the software artifacts, and (2) model code deployment for the automation system target, which is in general a Programmable Logic Controller (PLC) compliant with the international standard IEC 61131-3 for programming languages.

The paper describes a design pattern for the software conceptual model deployment, with a particular emphasis on practical requirements enforced by PLC of different brands. In fact, IEC 61131-3 compliance provides a common “look-and-feel” for programming languages, but does not guarantee straightforward code portability between different PLC vendors.

The paper has a great focus on applications and provides an example, based on a generic flow-pack machine, to better explain the proposed methodology.

1. Introduction

In the recent years, engineering problems related to the design, realization and maintenance of automatic machineries for advanced manufacturing processes were rapidly becoming more and more complex.

In fact, requirements for system flexibility, cost reduction and performance improvement have driven massive introduction of intelligent components (motion control, smart sensors, etc.) into the machine design. The integration of such devices with the mechanical parts and the control software development requires advanced methodologies to cope with the increasing system complexity, preserving efficiency under real-time constraints [15].

For these reasons design engineering methodologies had grow so much in the recent years. Conceptual modeling and structured engineering processes, mainly based on object-oriented approaches, have been studied and developed by several authors to provide the necessary support to engineers and developers to cope with these increasing technological demands.

In particular, several authors exploited the ‘mechatronic object’ approach to tackle project complexity by decomposing machine control system into modular components. The key concept of a “mechatronic object” is based on the definition of modular control functions that interacts through private signal interface with a mechanical part of the machine, to which it has an exclusive access, and with a public signal interface with other software modules. Therefore, mechatronic objects can be used to compound more complex systems, in which the internal complexity is hidden behind a public interface.

For example, paper [16] introduces the concept of MTC (Mechatronic Components and Connectors) to describe the development of mechatronic objects with “low granularity”, which can be used to build a complete application from basic elements of automation like sensors and valve, that communicates via well defined interfaces called MTConnectors.

The development of the “mechatronic object” approach made quite straightforward the introduction of system engineering methodologies that are based on conceptual object-oriented model of the artifact. In this research area, we can cite the paper [13], in which the authors describe the implementation of “mechatronic object” in the framework of the Unified Modeling Language (UML, see [10]). Protocol pattern with mechatronic object interface implementation issue is discussed in [14].

Design patterns are then useful to fill the gap between the conceptual model of the artifact and its deployment, as they provide common approaches for the solution of recurrent problem in the implementation of an applications (ed. state-chart translation to real time code, motion control handling, etc.).

A good overview on such a topic it can be found in the paper [12], while a specific design patterns for packaging machinery is described in [3].

From this point of view, we found a lack in the study of specific design patterns for the deployment of mechatronic
2. A Review on model based approaches for software development of manufacturing systems

The modern methodologies for software life-cycle management are based on engineering processes and procedures that provide guidelines for the software development and maintenance.

Most of these procedures requires the development of a model of the target system in order to address precisely system functionality to be implemented and push to surface any critical point. Models are also well worth in team projects as they make easier to share projects requirements and development steps between the engineering staff.

Many graphical notations have been proposed to support the model development, see for example [9] for a commented list of the principal graphical representations.

Unified Modeling Language (UML, [10]) distinguished itself in this wide range of graphical languages as it is an open method used to specify, visualize, construct and document the artifacts of a system under development.

UML in the Engineering Development Process is used to clearly communicate requirements, architecture and design, which can be applied in all iteration phases and used for Object-Oriented (OO) analysis and design. Object oriented modeling can be represent a complex system and its external relations. An object encapsulates both data and behaviors, and in the UML model it is associated to an abstract type, called class ([12]).

For the Machine Automation field, with respect to Object Oriented concepts the designer principal focus shifts from data to mechanics and control hardware. Thus the Software Object becomes a “Mechatronic Objects”, a unit that embodies the mechanical part, the electrical part and the control software ([13]).

A complex automation application (i.e. an automatic machine) is then build up of mechatronic objects that interacts through well defined interfaces (mechanics coupling, product exchange, software communication protocols, etc.).

The engineering process is based on capture of all the relevant information which are required to develop the control software. For the particular domain of packaging machineries, the following information should be specified by the model:

- The structural decomposition of the system in interacting objects. This description often begins with a walk through the “cases of use” (Use Cases) of the machine, which consist of an informal description of the machine functions and behaviors. The identification of machine functions leads to the development of an hierarchical structure of the system, in which each complex function of the machine is mapped into an object or a control module.

This structural decomposition is described using class diagrams (for conceptual design) or component diagrams (at deployment level).

- The reactive internal behavior of each object, which can be detailed using Statechart diagrams exploiting the richness of its graphical features (i.e. hierarchy, concurrency, inter-level transitions).

- The mutual interaction between mechatronic objects, or the interaction protocol, that has a static part describing the list of signals and their meaning, and a dynamic part, that describes the interaction protocol between two or more objects.

The static part can be described in a textual manner (useful in that case can be an excel file), where the dynamic part is well depicted by means of a sequence diagrams.

- Interface between the machine control system and the Human Machine Interface, usually a PC with touch screen panel. The interface is used to (i) supervise machine operations (i.e. to command machine start, machine stop, etc.), (ii) to set up machine operational parameters (usually named “production recipes”), and (iii) to perform troubleshooting in case of machine fault. The specification of system interface requires the definition of signals, which are generally defined using textual notation (e.g. list of variables), and they should be considered when classes (components) are defined.

In the next section it will be described how use the above mentioned diagrams to develop a conceptual model for an automated machine.
3. The development of the control software for a packaging machine.

As example, we consider the control model design of a generic film wrapper machine (see picture 1), that wraps a plastic film around food goods (e.g. chocolate bars, etc.).

The system use cases can be described as follows: the product enters the machine from the right side (see picture 1), then each piece of good is handled by a set of speed controlled belts to set the input in ordered pattern.

Figure 1. A generic film wrapper machine

Once the product item is set to the processing position, it is then wrapped by a plastic strip (film) that is unwound from a roll stock. The film is then sealed around the product using a melting heater, and cut to free the processed product from the remaining strip. Finally, the product item is carried out by means a conveyor (on the left side in the picture).

A motion control system keeps the motion of the strip stock synchronized with the product item shift. Usually, in an automatic machine, the motion control system is formed by a set of electrical motors (often Brushless electrical motor) which motion is synchronized to the position of a master, which can be either a real mechanical axis or (more often) a virtual axis. In the latter case, the master position is set by a software variable (a counter incremented by the control software to take into account the motion of product items thought the machine). In the example machine, the carrying belts (in-feed and out-feed) and the rolls stock for the plastic material movements are driven by the motion control system.

As shown by the example, an automated machine can be viewed as a serial machine in which the product is handled by a sequence of mechatronic group each of them performing a single action (or step) out of the whole process. In the example, the actions are: (1) grab the product form the machine input, (2) move and order the product, (3) wrap the product with plastic pellicle, (4) seal and cut the pellicle, (5) drive the product out the machine.

Thus, the static structure of a packaging machine can be described using a decomposition obtained from the identification of the mechanical parts that perform a single process step, which are candidates to became mechatronic objects which are generalized and represented in a UML class diagram. We can identify three classes that generalize the behaviors of the following three mechatronic objects (see fig 2: “Product in-feed”, that controls the actions (1) and (2), “Product out-feed”, that controls the action (5) and the “Sealing” that controls the actions (3) and (4).

Figure 2. The decomposition in three mechatronic objects, the product in-feed, the product out-feed and the sealing, of the flow pack machine.

For each class is defined:

- a private method that handles reactive behavior of the mechatronic object in terms of input/output signals from sensors and actuators. These signals are represented by private attributes, and can not be accessed form outside to enforce information encapsulation property of object oriented approach. The behavioral specification of this method is described by a state/transition model in a Statechart diagram.

- public attributes that implements human-machine interfaces, as described Sec. 2, and interface signals with other classes.

Moreover, the behavior of the mechatronic objects should be synchronized to achieve the final productive target for the machine by enforcing the following rules:

1. The machine is ready to produce if and only if all the mechatronic objects are ready to perform correctly.
2. The mechatronic objects starts the production simultaneously, or, in some particular case, one by one in the production arrival order.
3. If a mechatronic object fails, e.g. because a product jam or a system failure, all the previous (in the sense of product arrival) mechatronic object must stop, and the subsequent mechatronic object (if possible) can continue their job.
4. The machine finishes to produce if only all the mechatronic object stop as well.

The object synchronization can be achieved using two complementary approaches:
• **distributed control**, in which all the objects are at the same hierarchical level, and the above rules are enforced by a negotiation between them, following, for example, an approach defined by IEC 61499 norm ([4])

• **centralized control**, in which a system supervisor enforces the objects synchronization.

We decided to use the centralized approach because it comes naturally from the analysis of *machine use case* for the operator point of view. A complete list of use case of a packaging machine is out of the scope of this paper, but as example we can describe the machine start sequence:

The operator push the prepare button in the HMI panel, then all the machine mechatronic object starts its own preparation sequence. On the sequence completion, each object sends back a ready signal, so that the operator con proceed with the run command. At the run command, the motion control system initializes the master axis to move, and then, all the slave axes starts to move accordingly.

The above use case implies that there is a information exchange (preparation sequence, motion synchronization) between a common point and each mechatronic object.

From the above considerations, the structural decom- position of a packaging machine can be described using a UML class diagram in which each mechatronic object can be generalized in a class, linked by physical associations, that describe mechanical and information couplings points (see in figure 3 the class diagram of the example).

All the mechatronic object are linked with a information association with the machine supervisor, which has the stereotype of *machine controller*, as it supervises on the coordination over all the mechatronic objects compounding the machine.

![Figure 3. Class Diagram of the flow pack machine of the example](image)

In order to enforce object substitutability ([11]), we cho- sen to standardize the object information interfaces in two manners: (i) each object can be (information) associated only with the Machine Controller, and (ii) the object should present a common data structure for information exchange through this interface.

Each mechatronic object can be further decomposed in further detailed class diagram, at a lower level of hierarchy, depending on the complexity of the mechanics and control parts it describes. Being decomposed or not, the generic mechatronic object is described by:

1. A Statechart diagram that describes the dynamic behavior of the object belonging to a class. In each state, the actions (operation which last exactly one control cycle) or activities (operation lasting more controller cycles as the axis motion control) performed by the object are defined.

2. The interface with the Machine Controller, that is defined using:
   - A set of Sequence Diagram to define the communication protocol between the object and the Machine Controller.
   - A Data Structure described using a textual notation (IEC formalism or an excel file).

3. Description of non state-dependent behavior using plain text or target controller code fragments. As example, the handling of continuous process which is enforced in all the possible object state.

4. The alarm data base.

5. The production parameters data base for each machine recipe.

Once all the relevant information on the application has been collected and discussed among the development team, the system analysis work ends and begins the coding part. In this phase the high level information collected into the UML model must be translated to the target automation platform code. Since there is a plethora of different automation platform, the coding phase is not trivial, especially if we consider the portability among different PLC brands issue. To cope with this problem we propose a *control software deployment* design pattern, which is the main topic of the next section.

### 4. A design pattern for the control software deployment.

In this section we describe a design pattern we propose to translate the UML software model to the controller target code.

The principal PLC manufacturers (Siemens, Rockwell Automation, B & R, Elau, etc.) declare their systems compliant with the international normative that standardize the programming language for automation, the IEC 61131-3 ([5]). Unfortunately, this compatibility is very basic and it does not permit a straightforward portability of the source
code (to not mention the whole application) from different brands’ system.

In fact, each manufacturer provides a software tools for the PLC configuration (number and address of physical I/O, number and characteristics of the electrical motors, CPU, interface with the communication system, etc.) and for the code development. The configuration information are not standardized by the IEC norm, thus each vendor use a file format non interchangeable.

Concerning the code portability, only the text based (as Structured Text and Instruction List) can be ported with a cut and paste of the code, whereas the graphical languages (Ladder Diagram, Sequential Functional Chart, Functional Block Diagram) are not in generally portable.\(^1\)

For the above reason, our proposal does not refers directly to IEC normative, that in our opinion still be too generic concerning the portability issue, but in the following we report about the model deployment in relation to vendors, the B& R (in the following it will be named BR PLC) and Rockwell Automation (in the following RA PLC), that are taken as examples to describe a general approach to UML model deployment.

4.1 The class diagram deployment

The class diagram describes at the abstract design phase the structural decomposition of the system, that is mapped in the set of the mechatronic objects that form the overall system. As well as the system development proceed further to the deployment phase, the class diagram information are mapped into detailed software design specification and then into code. In this latter phase the system static structure is described by a Component Diagram.

The class diagram may have more levels of decomposition, as it can be broken down into several hierarchical levels, depending on complexity of each module. Recent drafts of IEC standard ([6]) introduced the class as a type of Program Organization Unit (POU), but at best of our knowledge, it still at draft stage and no manufacturer has yet implemented this POU.

Thus, the most natural IEC structure for the class diagram implementation is the concept of “program” since it implements a software module with internal states and static variables with local scope. Both BR and RA PLC implements the “program” POU as an application software module which has its own private data structure. Referring to our example, we can design an application with four programs, namely the machine_controller, product_infeed, product_sealing and product_outfeed.

Concerning the implementation of class’ private and public attributes, it should be stressed that they maps to the following machine signals:

- **Public attributes**: public interface variables between mechatronic objects and the machine controller and the HMI system.
- **Private attributes**: interface signals between the controller and the sensors and actuators of the mechatronic object. It should be noted that each PLC vendors has a proprietary way to define such a interface at controller configuration phase, and in general each interface signal is defined as a global variable (i.e. in general is not possible to associate a set of interface variable with a POU).

The most appropriate way to implement public attribute is by using VAR IN, VAR OUT, VAR IN_OUT definitions. Since both RA and BR PLCs doesn’t implement such a IEC constructs, we used global variable in case of RA PLC and data block definition in case of BR PLC. The data block in BR PLC is a software module that contains only variables, which visibility is limited to only programs who referenced that data block, in a way similar to the # include construct of the “C” language.

As discussed before, class private attributes must be declared as global variables in the configuration step of the control application. The turnaround for this fact, which violates the information hiding basilar concept, is by using the alias property of the RA PLC and, once again, the data block for BR PLC. The alias construct is a way to reference a variable by absolute address in RA PLC application, thus we use this construct as a pointer to private attributes in RA PLC application.

4.2 The motion control

Each PLC brands has a proprietary solution for motion control configuration and programming. The PLCopen [11] provides an open standard for the motion control handling. This standard is based on the concept of axis control. An axis is always in one of the defined states (Disabled, ErrorStop, StandStill, MC_Home, MC_Stop, MC_MoveSuperimposed, MC_GearOut, MC_CamOut). Any motion command that causes a transition changes the state of the axis and, as a consequence, modifies the way the current motion is computed.

The state diagram is an abstraction layer of what the real state of the axis is, comparable to the image of the I/O points within a cyclic (PLC) program. The idea here is to provide a PLCopen standard interface for any brand of motion control, by designing a function block that masks the underling specific motion control system, and translate the control and status signals form PLCopen standard to the specific brand.

4.3 The State chart deployment

Sequential Functional Chart, or SFC, defined by IEC as a common structure to organize POU (which it is indeed referred by almost all the PLC vendor as the fifth programming language), seems the most appropriate way to
implement UML Statechart Diagrams. However, the impossibility to represent hierarchical state machine in SFC forces to “flat” the Statechart structure into only one level, loosing almost all the original diagram clarity and effectiveness.

Thus our choice is to use Ladder Diagram (LD, an IEC graphical language that resembles the electrical schemes) or, better, the Structured Text (ST, a textual IEC language, which has a syntax close to Pascal) to implement the structure of the Statechart diagram, both in RA and BR PLC.

Coming back to our example, each Statechart defining the reactive parts of the above defined programs, is then translated to a LD or ST code. In both the cases, each Statechart diagram (e.g. taking as a reference the Statechart example depicted in figure 4) is decomposed into the following parts:

1. Setting of the initial conditions. In this part, executed only once at first PLC scan, the initial state is set. Each state, at any hierarchical level, corresponds to a boolean variable.

2. Evaluation of the enabled transitions. Each transition is marked by a boolean expression. A transition is “fireable” if the corresponded boolean expression evaluates to TRUE and its source state is active.

3. Evaluation of conflicts between transitions: because of hierarchy, more than one transition can be simultaneously fireable. However, the transition exiting from a higher-level state has an higher priority with respect transitions from an internal state.\(^2\)

4. Update of the states: states exited because of a transition fired are deactivated, including those contained in an exited super-state, and destination states are activated. The latter may be related to a History pseudo-state (see figure 4), the circled “H” symbol), which requires specific state bits management.

5. Evaluation of active super-states: in general, super-states may be AND-states or OR-states, so that their configuration can be managed with simple boolean expressions on lowest-level states.

6. Execution of the actions correspondent to each state or super-state.

5. Conclusion.

This paper describes a design pattern used to translate a packaging machine UML model into PLC code. The design pattern complies with practical issues in real implementation of IEC 61131-3 international standard. In particular, it is discussed here a methodology for the translation that permit an easy porting of the code from one vendor specific platform to another one.

References


