Design of IEC 61131-3 Function Blocks using SysML

Fabien Chiron*, Khalid Kouiss*

* Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS)
Blaise-Pascal University of Clermont-Ferrand, BP 10125, 63173 Aubière, France
fchiron@newtec-group.com,khalid.kouiss@ifma.fr

Abstract—UML is almost inevitable when dealing with software designing and intends to be extended to over many different disciplines. At the same time, control and automation designers integrate further object oriented concepts in their traditional programming behaviors in particular with Function Blocks. Whereas UML was not adapted to model these IEC 61131 compliant items, an important extension called SysML has been proposed. This paper deals with evaluating the new modeling abilities of this language according to programmable logical controllers specific programming rules.

Key Words—Control and Automation, SysML, IEC 61131-3, Function Blocks, MDA, UML.

I. INTRODUCTION

An important issue for companies while designing innovative products and applications is to ensure the capitalization of the specification work for future developments. One main difficulty often encountered is the target environment set of constraints that leads to take into account technological aspects in high level stages of specification. Problems occur when the technology evolves, and require modification in the previous design even if the global process remains the same.

To increase the reuse capacity of such developments, modeling techniques have been improved in parallel with object oriented (OO) design methodologies, particularly in computer science.

Our paper deals with these two aspects applied to automation, and represents a continuation of a more global research project about component-based design for distributed control systems [14].

A. Model-based engineering

Advantages brought by modeling techniques are nowadays widely recognized. They led to the release of efficient design methodologies in a vast range of domains (GRAI, CIMOSA, PERA, IDEF-3, etc...). However they usually stick to specific areas or levels of conception and remain difficult to use outside of their own context. So designers have to use many methods in various stages of design or they change the semantics of some concepts in order to better fit their needs.

Model Driven Architecture (MDA) paradigm [1] recommends the splitting up of design models into a succession of model refinements using Platform Independent Models (PIM) and then applying different mapping to obtain Platform Specific Models (PSM) at the last steps of design. According to this approach, integrity of the high level model is maintained regardless of the platform with its specific standards, patterns and constraints.

To give concrete expression to this paradigm, the Object Management Group (OMG) has put in a lot of effort for the specification of a Unified Modeling Language (UML) [2] allowing the design of software applications independently of a specific programming language. The final code implementation is mainly obtained using an UML profile and a modeling tool that allows the code generation process.

However, whereas UML was well-adapted to software design, modeling of real-time systems and of Programmable Logical Controllers (PLC) programs was hard to achieve. Many papers underlined these lacks and proposed extensions to UML, e.g. B. Selic in [5], where new stereotypes were introduced and widely reused in more recent papers that deal with time-driven systems modeling. According to all these works, an official extension to the version 2.0 of UML has recently been released by the OMG, the Systems Modeling Language (SysML) [4].

B. Object Oriented Design

In software development area, the OO programming has undoubtedly been a major revolution, allowing a great flexibility inside the code structure and breaking the traditional monolithic way of programming.

Later, the OO concept has been introduced in Control and Automation programming by the International Electro-technical Commission (IEC) through one of the standard concept described in the IEC 61131-3 specification [3], the Functional Block (FB). However, a FB differs by many aspects from a «class» concept and couldn’t be represented using the modeling tools used for software programming.

A new standard IEC 61499 [6] is currently under work and redefined the FB concept, including event management. Although this approach represents an interesting improvement for PLC programming, leading PLC solutions suppliers did not make important efforts on the integration of this new approach, except Rockwell Automation and its Function Block Development Kit (FBDK) by J. Christensen [14]. For the vast majority of control engineers, orienting development with this standard is not relevant at the moment.

In this paper we will focus on the IEC 61131-3 standard and we will show how the SysML extension provides great possibilities for integrating actual PLC programs design using UML.
II. FUNCTION BLOCK FEATURES

A FB encapsulates a known behaviour that will react depending on the input data values represented on the FB left side. An overall representation is given Fig. 1.

When instantiating a FB in a program, every input variable must be linked to an external variable and follows its evolution, the FB owning this variable cannot modify its value (read-only access). On the contrary, a FB owning an output variable can write or read it, while the execution environment can only read it. Input/Output variable inherits both features.

Intern variables can be read or write by the owning FB and can usually be set public or private. Intern behavior is described with the IEC 61131-3 languages and can reuse the owned variables according to the previous described rules.

Intern variables are defined as properties, like instances of blocks participating in the intern behavior of the owning block. Internal structure with intern relationships between properties is described using an “internal block diagram” (ibd). Connections between “flow port” are also represented on this diagram.

B. Flow Ports

Flow ports represent the major extension to UML 2.0 including new graphical symbols. According to the standard, a flow port specifies the input and output items that may flow between a block and its environment. It is possible to define multiple items flowing via a flow port, such a port is called “non-atomic” and is linked to a “flow specification” that lists all the items. In our case we prefer using the “atomic” flow port in order to stick with the IEC 61131-3 FB. An example is given Fig. 2, representing a FB that receives an input bit value «e_S» and sets an output bit value «e_S» to 1 if the time intervals between input changes are relevant, depending on an input sensibility value «Delay». To fulfill this task, the block defines the internal variables «_memSU» and «_memSD», and the internal block instances «top», «timerU» and «timerD». The block definition corresponding to these instances are also illustrated, «TopD» which detects rising edge and falling edge, and «TON» a timer with a preset value.

III. MODELING FB WITH UML

In PLC programs, code consists of sections started and executed sequentially. No call services are implemented by a FB and intern behaviour is activated depending on its position in a program section. UML “interface” stereotype is not relevant to express the external link of FB with its environment. Indeed, a FB does not expose services to the outside but static data. UML class attributes does not either fit to model Input variables due to its specific access rights.

If we first focus on a structural view, UML 2.0 introduced the “port” stereotype in the language and many papers dealt with the correspondence between this port and the FB variables interfaces like K. Thramboulidis in [8], [9] and [10] with the introduction of “data port” which are standard ports with attributes without services, and connected to the “capsule” stereotype introduced in [5]. T. Heverhagen in [7], [11], [12] proposes a new item, the Function Block Adapter (FBA), that may contain at the same time standard ports and input/output variables on the same representation. With this hybrid view, he underlines the lacks for the actual standard representation when trying to represent FB in UML. The SysML extension introduces a new “flow port” stereotype offering a standard way to express data interfaces with a certain similarity with FBA.

Until UML 2.0, to express the specific management of the tasks in a PLC, it was necessary to introduce new stereotypes. For instance, UML-PA [13] points out important representation needs in behavioral constructs according to Real-Time constraints in a PLC. UML 2.0 brought improvements at this level and SysML extended the UML 2.0 activity with flow inputs and outputs among actions, allowing deeper consideration when designing data flows. The next parts describe the SysML extensions enabling a FB description in UML.

IV. MODELING FB WITH SYSML

A. Block Stereotype

SysML extends the Class stereotype defined in the UML 2.0 Composite Structure Diagram by introducing a new stereotype called “Block”. In our context, blocks are a relevant manner to represent subparts of a system or functions that belong to a process, like Function Blocks in the IEC 61131 standard.

To express the structural relationships between blocks, a “block definition diagram” (bdd) is created. It can be useful to describe generalizations that are now allowed in some PLC programming tools when designing FB. Interface variables are also displayed on this diagram, using a “flow port” symbol.

Intern variables are defined as properties, like instances of blocks participating in the intern behavior of the owning block. Internal structure with intern relationships between properties is described using an “internal block diagram” (ibd). Connections between “flow port” are also represented on this diagram.

B. Flow Ports

Flow ports represent the major extension to UML 2.0 including new graphical symbols. According to the standard, a flow port specifies the input and output items that may flow between a block and its environment. It is possible to define multiple items flowing via a flow port, such a port is called “non-atomic” and is linked to a “flow specification” that lists all the items. In our case we prefer using the “atomic” flow port in order to stick with the IEC 61131-3 FB. An example is given Fig. 2, representing a FB that receives an input bit value «e_S» and sets an output bit value «e_S» to 1 if the time intervals between input changes are relevant, depending on an input sensibility value «Delay». To fulfill this task, the block defines the internal variables «_memSU» and «_memSD», and the internal block instances «top», «timerU» and «timerD». The block definition corresponding to these instances are also illustrated, «TopD» which detects rising edge and falling edge, and «TON» a timer with a preset value.
Completing the type specification of a flow port, a direction prefix must be explicitly specified (in|out|inout) before the flow port name. Fig. 3 displays the new graphical notation allowed in SysML. Inout flow ports are displayed with two arrows facing away from each other (<>).

We do not explain how to represent IEC 61499 FB with SysML in this paper, but a block can own additional standard ports within the same diagram, offering a way to precise event management on the previous view.

Flow ports inherits from UML 2.0 port and can be set as behavioral ports or not. In our case we choose to set the meta-attribute "isBehavioral" to false. Indeed, we try to stick to the FB structure conserving only one behaviour description in our model, the one owned by the block.

C. Program implementation

PLC program execution passes sequentially through sections containing subprograms described in one of the IEC 61131-3 standard languages. A FB can be used by any of these languages but the Functional Block Diagram (FBD) is particularly interesting when assembling many FBs together. It allows a graphical programming and developers can connect manually input and output between blocks as well as with system data like topological addresses.

The SysML Internal Block Diagram is a relevant way to represents FBD in UML. The example given in Fig. 4 illustrates the internal block diagram for «SensorCtrl» task. It describes how flow ports can be linked to internal parts and to external data, it also introduces a new stereotype "Item Flow". Whereas a flow port defines what can flow, an item flow defines what does really flow and describes connectors between flow ports.

V. Behavior Representation

After structural considerations, behavioral aspect can also be managed with SysML in order to model the execution sequences that operate in the PLC program. There are two main points to focus on.

First the global description of the different tasks and their arrangement inside the master task. In this paper we will not deal with insertion of auxiliary and fast tasks. In a vast majority of PLCs, execution mode can be cyclic, the master task restarts when it just ends, or periodic, if the master task ends before the periodic time that has been set up, system wait until time has been elapsed.

Other point is the internal behavior associated to a block stereotype.

A. Activities and Allocations

Various ways of representing behaviour are available in UML 2.0, use cases at a high level of description, interactions inside sequence diagram to express service and operation calls between entities, state machines diagram for the description of discrete behaviours and transitions between them, and finally activities that are a flexible way to specify and link tasks to blocks in SysML.

Another advantage of using activities in SysML is the possibility to exploit crosscutting constructs introduced in SysML. The "Allocations" management in particular permits to allocate a behavior item to a structural item. For these reasons we choose the activity diagram to display behavior description in the next sections of this paper.

One way to define these links is to use the compartment notation in a block definition diagram or internal block diagram as shown in Fig. 5.
In a similar way, it is possible to allocate Item Flows from Object Flows in activity diagrams, giving more coherence to the overall model when designing behavior and structure for FB in UML.

Activities are described with more details in activity diagrams mainly consisting in our case of action sequences and object nodes.

B. Master Task Decomposition

Considering the first decomposition level of a program, we aim at identifying major sub-tasks that belong to the execution model. Once this work achieved, important sub-programs that have been obtained will be represented as blocks elements. A block will be described with a block definition diagram, describing a kind of black box view with some additional information about the intern sub-elements participating in its realization. An internal block diagram is associated at the same time and a main activity is defined for this block during the same phase.

During the process, the master task gives execution control to every sub-task enabling its intern behavior until the end of the activity processing. Master task activity is described Fig. 6. The «SubTask» actions are marked with a rake-style symbol in order to specify that they call behavior more detailed at a lower level of design. For this reason they are associated with an activity which is itself allocated to a block.

This scenario is repeated for each level of decomposition contained in the block.

C. «SignalFilter» block example

We use the block "SignalFilter" described in Fig. 2 as a sample to illustrate a possible way to define FB behavior in UML, resulting in the Fig. 7 activity diagram. If we except the allocation specification, SysML semantics used in the activity diagram remain the same. Control and object flows included in the diagram activate different actions and activities in order to realize the owning block behavior.

Input variables associated with the FilterSignal block are represented as parameter nodes on the left side of the activity, output variables on the right side. Three sub-activities are included in the diagram. The «Top» is linked by an allocation relationship to the «TopD» block illustrated in Fig. 2.

We recommend only one activity allocation for one block. It offers a more coherent correspondence between inputs and outputs described on the structural interfaces and the dynamic view proposed in the activity. If various sections have to be defined separately for a block, an intermediate activity diagram with sub-activities must be refined in a similar way to Fig. 6.

Intern variables displayed for a block and expressed as block properties participate to the internal behavior. Activity attributes must be link to block attributes for better coherence in the model.

It may happen cases where intern variables must be linked to system values or to external blocks that are not in the intern structure. In this case activity which contains object flows corresponding to the variable should be allocated to item flows as shown Fig. 8.

VI. ABOUT THE DEPLOYMENT

A great interest when using UML to model software applications is its ability to be represented in a coherent xml description. A direct consequence is that models compliant to the UML standard can be exchanged from one modeling tool to others. Moreover, the correspondence between the xml description and a specific language description can be fulfilled in a UML profile. Many plug-ins have been implemented for UML.
modeling tools and allow generation of code in a wide range of programming language (C++, Java, Delphi, ...).

SysML is a recent specification and its integration in modeling tools is not completed at the moment. A potential code generator would offer the possibility to choose the IEC 61131 language targeted.

A example of such a possible generation for a block is illustrated Fig. 9. The first part describes the structural items of the blocks like it was explicitly displayed in the block definition diagram. The next parts translate the allocated activity depending on the selected PLC language, in our case some Structured Text (ST).

It is important to underline that some PLC suppliers already include xml representation of FB via export/import tools, Schneider and its UNITY platform for instance. That means that with a correct UML profile for the code automation, it is conceivable to import FB into PLC programming environment by the intermediate XML generated files. We currently work on this kind of approach and we evaluate its possibilities and relevance to develop control and automation programs.

<FBItem name="SignalFilter" version="0.01">
  <comment>FE used to filter signal variations</comment>
  <variables>
    <variable name="s_0" type="BOOL"/>
    <variable name="s_Delay" type="INT"/>
  </variables>
  <input>
    <variable name="s_IN" type="BOOL"/>
  </input>
  <output>
    <variable name="s_OUT" type="BOOL"/>
  </output>
  <interfaceName>
    <variable name="s_FRONT" type="BOOL"/>
  </interfaceName>
  <FBSource name="FBSignal">
    _TOP (s_0 || s_IN);
    _timerFD = INT_TO_TIME(s_Delay)
    IF _timeFD > 0 THEN _FM = 1;
    IF _timeFD = 0 THEN _FM = 0;
    _timerPD = INT_TO_TIME(s_Delay)
    IF _timePD = 0 THEN _FM = 0;
  </FBSource>
</FBItem>

Figure 9. FB XML description

VII. CONCLUSIONS

The main purpose of this approach is not to replace the traditional programming environments with a UML tool for persons that are currently developing PLC programs. Training of this staff category in UML utilization would be not a viable solution and the release of tools offering such possibilities is not planned at the moment among automation community.

However designers complain about a recurrent lack of coherence between the different steps of conception when creating new products or solutions and particularly when reaching the last conception stages. They also need methods and tools to capitalize their development for future projects. A way to avoid departing from the original plan is to use model-based engineering as far in the conception process as it is possible. This paper intended to rejoin this paradigm in order to integrate control and automation description at the modeling step, among other technical and high-level considerations.

We saw that efforts have been made by the OMG in order to integrate specificities of Real-Time systems in UML via the SysML extension. This work is recent but it offers new possibilities for modeling automated systems with component-based methodologies. Indeed, every subpart will be represented with the same set of diagrams, each covering its own development target via a specific model extension, but belonging to the same component global package.

REFERENCES