

## PLC PROGRAMMING LANGUAGES: A FORMAL APPROACH

Luciano Baresi<sup>(+)</sup>, Stefania Carmeli<sup>(++)</sup><sup>1</sup>, Antonello Monti<sup>(++)</sup>, and Mauro Pezzè<sup>(+)</sup>

<sup>(+)</sup>*Dipartimento di Elettronica e Informazione - Politecnico di Milano*

*Piazza L. da Vinci 32, I-20133 Milano, Italy*

*Phone: +39-02-23993400 Fax: +39-02-23993411*

*E-mail: baresi|pezze@elet.polimi.it*

<sup>(++)</sup>*Dipartimento di Elettrotecnica - Politecnico di Milano*

*Piazza L. da Vinci 32, I-20133 Milano, Italy*

*Phone: +39-02-23993702 Fax: +39-02-23993703*

*E-mail: carmeli|anto@etec.polimi.it*

### **Abstract**

This paper introduces describes how to integrate standard editing and code generation functionalities offered by most tools supporting the IEC standard 1131-3 with capabilities for modelling and simulating the plant and its interactions with the digital controller. The 1131-3 notations (in particular Functional Block Diagrams) are complemented with differential equations that describe the behaviour of the plant and with an underlying formal model, which supports the analysis of functional and timing properties.

### **Introduction**

Standards play a key role in industrial development. They enhance compatibility, openness and interoperability among different products, increase the confidence level of users and certification agencies, and promote the development of tools and methodologies by freezing notations, methods and techniques.

The IEC standard 1131-3 [5] defines a set of notations and methods for developing control systems. It is now adopted by many manufacturers and is increasingly supported by tools. Most existing tools support editing of the IEC 1131-3 notations and code generation, but provide little support to simulation and analysis of early specifications.

In this paper, we describe how to complement the main editing and code generation functionalities offered by most tools supporting the IEC standard 1131-3 with capabilities for modelling and simulating the plant and its interactions with the digital controller. Suitable simulation and analysis of the early specifications of the digital controller integrated with the plant can identify functional and timing errors that would be difficult to identify in and expensive to remove from the final system.

In this paper, the 1131-3 notations (in particular Functional Block Diagrams) are complemented with differential equations that describe the behaviour of the plant and an underlying formal model, which supports the analysis of functional and timing properties. The paper introduces the approach and describes the industrial experience within the ESPRIT project INFORMA<sup>2</sup>.

---

<sup>1</sup> Luciano Baresi and Stefania Carmeli are partially supported by Ansaldo Sistemi Industriali S.p.A., viale Sarca 336 - 20126 Milano (Italy)

<sup>2</sup> The work presented in this paper has been partially funded by European Community under ESPRIT Project INFORMA (EP23163). The consortium comprises IFAD (Denmark),

## **IEC Standard 1131-3**

The IEC standard 1131-3 [5] defines the software model of programmable controllers and the languages to program it. The many proposals and dialects from PLC vendors result in a suite of five programming languages:

- **Instruction List (IL):** a low-level textual language with a structure similar to assembler. IL is well suited for solving small straightforward problems and producing optimised code, but it does not support structured programming. IL can be interpreted directly by many IEC1131-3 compliant PLC. This is why IL is sometimes considered *the PLC language* in which all other IEC1131-3 languages can be translated.
- **Structured Text (ST):** a high-level procedural programming language. ST borrows its syntax from Pascal, augmenting it with some features from Ada. ST enforces data typing and support structured programming. Sometimes considered *the new PLC programming language*, it provides useful means to handle the complexity and modularity of modern programmable controllers.
- **Ladder Diagrams (LD):** an evolution of electrical wiring diagrams. LDs supply a programming style borrowed from electronic and electrical circuits. LDs are ill suited for the complexity of today's controllers and structured programming, but help legacy systems.
- **Function Block Diagram (FBD):** a graphical language similar to Structured Analysis [7]. Controllers are modelled as signal and data flows through processing elements (*function blocks*<sup>3</sup>). FBD transforms textual programming (ST) into connecting (already defined) building blocks, thus improving modularity and software reuse.
- **Sequential Function Chart (SFC):** a graphical language similar to Petri nets [8] and SDL [9]. SFC is used for structuring the internal organisation (behaviour) of PLC processing elements. SFC elements partition processing elements into sets of steps and transitions among them interconnected by directed links. Steps are associated with actions; transitions with predicates (conditions).

The IEC1131-3 standard fosters the development of well-structured software. Function blocks are the key concept toward enforcing modularity and designing for reuse. Well-defined algorithms or control strategies can be packaged as reusable software components by embedding them in suitable functional blocks. This paper focuses on reusable components, and thus on FBDs. IL, ST, and LD are seen as possible means to define the "internals" of function blocks, while SFC is used as *glue* to combine all processing elements of a programmable controller.

## **Approach**

The design of programmable controllers can be improved by an approach that integrates standard functionalities for editing and generating code with new functionalities for simulating, and analysing embedding (hybrid) systems, where the digital controller and the analog embedding are strictly intertwined. The approach described in this paper integrates

---

Ansaldo Ricerche (Italy), Politecnico di Milano (Italy), Odense Steelshipyard (Denmark), Ansaldo Sistemi Industriali (Italy), and Scientific Software Group (France).

<sup>3</sup> The term *function block* is used here with a broader meaning with respect to the standard, to consider all the kind of processing elements of the software model

widely used notations and tools to deliver a toolbox to be employed in industrial practice. The digital controller is specified with FBD<sup>4</sup>; the analog embedding is modelled with Matlab/Simulink [6].

FBD models are automatically translated in high-level timed Petri nets, hereafter HLTPNs, by means of suitable rules. These rules define the HLTPNs that are functionally equivalent to the function blocks. The creation of new function blocks imposes the definition of either the HLTPN or the C code that specify its behaviour. The newly defined element can be used as if it were a library component. The dual language approach described in [1,2] allows control engineers to automatically build the HLTPN corresponding to their FBD models. HLTPNs remain hidden to control engineers. They act as formal engine that provides specific services through the FBD formalism. HLTPNs are used to:

- **Execute (simulate) FBD models.** HLTPN execution events are shown as visualisations (animations) of FBD elements.
- **Debug FBD models.** The execution described so far can suitably be controlled to let control engineers debug their specifications.
- **Analyse (validate) FBD models.** The corresponding HLTPNs are analysed by exploiting the usual techniques and algorithms for Petri nets (for example, reachability analysis, model checking, place invariant) and results are translated in visualisation understandable by FBD experts.
- **Generate C code from FBD models.** HLTPNs serve as intermediate representation to automatically generate the C code corresponding to an FBD specification. The generated code meets all the properties proved by analysing the HLTPN.
- **Derive test cases.** The HLTPN is used to guide the generation of test cases that will be used during the testing activity on the final target CPU.

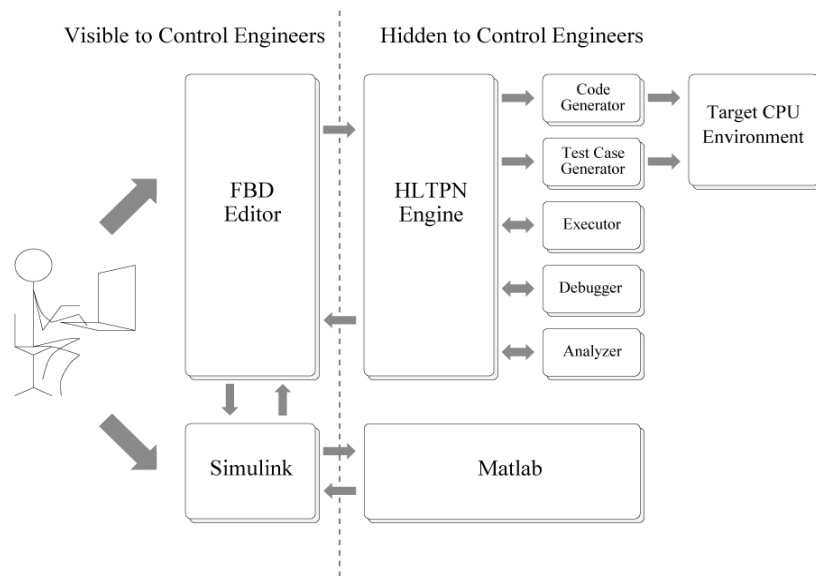
The logical components of the INFORMA toolbox are presented in Figure 1. Control engineers define the whole embedded system with both Simulink and the INFORMA FBD Editor. Then, they exploit both Matlab to execute the analog embedding (solve the equations that describe the embedding in the time domain) and HLTPNs to perform the tasks described so far.

Simulation of both the embedding and the controller is carried out in an integrated framework, thus, control engineers can get the overall behaviour of their systems at a glance. The same integrated framework supplies the controller with significant analog inputs during debugging activity. Figure 2 shows a sample INFORMA simulation in the Simulink environment: the block *Plant* hides the definition of the plant model in terms of standard Simulink blocks while the block *Control System* hides the control specification in terms of FBD and VDM<sup>++</sup> specification linked through tailored INFORMA blocks.

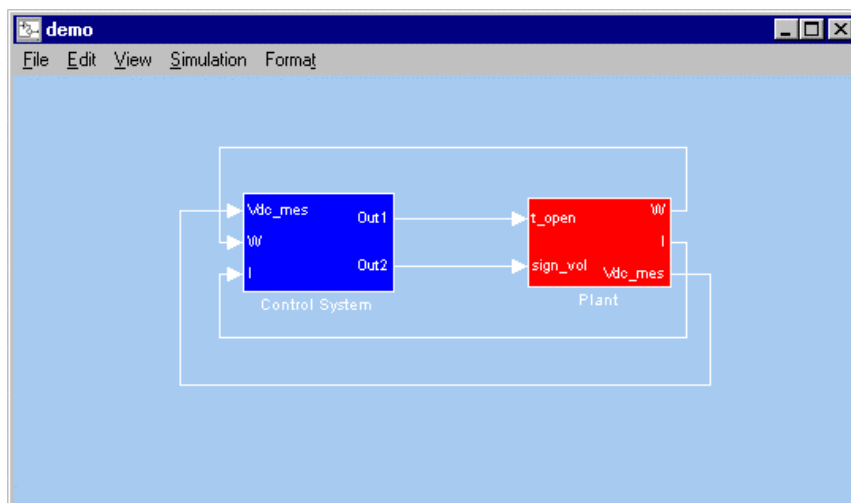
The INFORMA approach provides the *glue* to make industrial users work with international or de-facto standards (FBD, VDM<sup>++</sup>) and widely used tools (Matlab/Simulink) in an integrated framework, exploiting the benefits of formal methodologies. The formal methodology (HLTPN) is the core of the toolbox, but domain experts (control engineers) do not interact with it directly. They go on using their notations and techniques, they gain its services and benefits, but they can even ignore it exists.

---

<sup>4</sup> Soft Real Time components can be specified with VDM<sup>++</sup> as well. VDM<sup>++</sup> is not considered in this paper, interested readers can refer to [3] for further details



**Figure 1: High level components of the INFORMA toolbox**



**Figure 2: a sample of integrated simulation in a Simulink model**

In the following paragraphs a detailed description of the PLC editor contained in the INFORMA package as well as an introduction to a real application of the methodology will be presented.

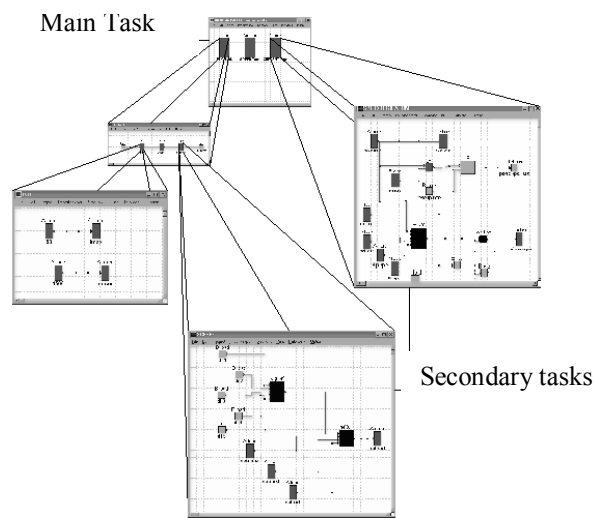
### **The INFORMA ISO-PLC editor**

Starting the INFORMA package from the Matlab Command window is possible to activate the INFORMA ISO-PLC Editor that allows the user to quickly develop new specification using the FBD language. From the user viewpoint the main steps of an ISO-PLC model design are:

- opening of an existing model or creation of a new one
- adding blocks to the scheme dragging them from the available libraries
- eventually definition of new blocks:

- generic block: the user can specify the block functionality using a high level language (i.e. C code), while the graphical aspect is standard;
- customised block: the user can specify both the aspect and the functionality, acting on the Petri Net specification;
- library block: a customised block can be added to the libraries and made available for future employment;
- saving of the specification carried out.

In what follows the main editing element are introduced together with their syntactic role.



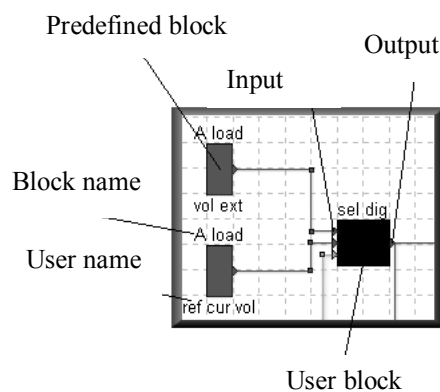
**Figure 3: Typical hierarchical structure of the sessions.**

### Session

A SESSION is a working window where it is possible to draw block diagrams. The whole of sessions is structured dynamically (see Figure 3).

### Block

Blocks are the basic elements of the notations. There are three kinds of BLOCKS (see Figure 4): *normal*, *task*, and *session*, all based on the same structure.



**Figure 4: The object block**

*Normal blocks* are operational. They perform logical or mathematical operations (e.g., *AND*, +), or they control input/output (e.g., *load*, *store*). *Task blocks* describe the software architecture in terms of scheduling strategy. *Session blocks* represent new sessions.

*Task* are operative units; while sessions graphically describe the code to be executed. Predefined task types are: *Main*, *Daemon*, *Interrupt*, and *Exception*.

The *main task* is executed periodically. At each cycle it calls the related sessions. A main task is always present in a specification and can be interrupted only by tasks of higher priority, e.g., a daemon task.

A *daemon task* has the highest priority level and is executed according to a predefined timing. It is not possible to assign the same priority to more than one daemon task.

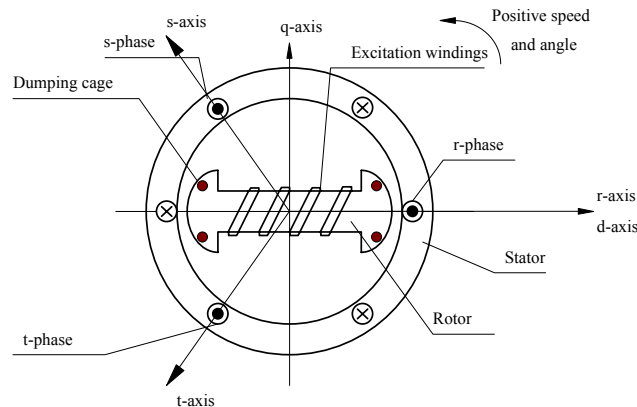
An *interrupt task* model an interrupt handler.

An *exception task* takes control of the CPU on an exception. No more than one exception task is allowed.

For each task it is possible to specify several sessions, obtaining a tree structure.

The information flow among the blocks is represented by lines displayed using the colour of the related type of data.

## A synchronous machine speed control



**Figure 5: Section of an AC synchronous machine**

From the structural viewpoint a synchronous machine is composed of (see Figure 5):

- an external stator with a three phase winding acting as induced element
- an internal rotor with a DC winding acting as inductor element. the rotor presents a short-circuited winding to limit the transient under quick variation of the load (dumping cage).

In speed control applications, the synchronous machine is equipped with an encoder that allows to know rotor-position and rotor speed (*act\_speed*), the stator currents (*ir\_act*, *is\_act*, *it\_act*) and the excitation current (*ie\_act*) are also monitored. The control circuits utilize the rotor speed, rotor position and monitored currents to control machine speed and torque by modifying the voltages of a cycloconverter (i.e. a direct AC/AC converter) and the excitation current in the rotor winding.

In Figure 6, the reference rotor speed (*n\_ref*) and the actual monitored value of the rotor speed (*n\_act*) are compared and their difference is fed into the speed controller, which is a PI controller. The output voltage of the speed controller is proportional to the developed electromagnetic torque (*t<sub>e</sub>*) of the synchronous machine, and thus the reference torque (*t<sub>e\_ref</sub>*)

is obtained. This is divided by the modulus of the magnetizing flux-linkage space vector ( $\psi_{act}$ ) to yield the reference value of the torque-producing stator current component  $iy_{ref}$ . The monitored rotor speed serves as input to the field weakening block which below base speed yields a constant value of the magnetizing flux reference ( $\psi_{ref}$ ); above base speed this flux is reduced. The flux-linkage reference ( $\psi_{ref}$ ) is compared with the actual value of the magnetizing flux linkage ( $\psi_{act}$ ) and their difference is fed into the dynamic flux regulator, which is also a PI-controller.

The flux controller maintains the magnetizing flux linkage in the machine at a preset value, independent of the load. To ensure optimal utilization of the motor, the magnetizing flux-linkage reference ( $\psi_{ref}$ ) is raised to a very high value during starting, to yield a high breakdown torque, but for normal operating conditions it is set to the rated value. The output of the dynamics flux regulator is the reference value of the magnetizing-producing stator current component ( $ix_{act}$ ).

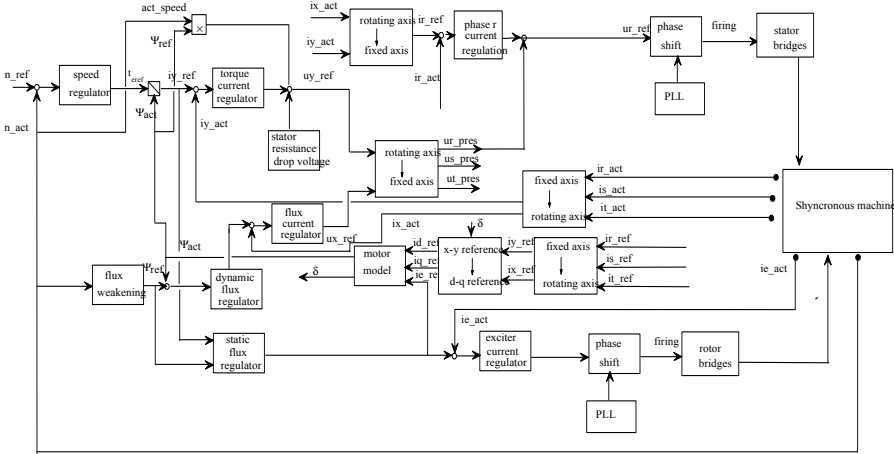


Figure 6: Control schema for AC synchronous machine

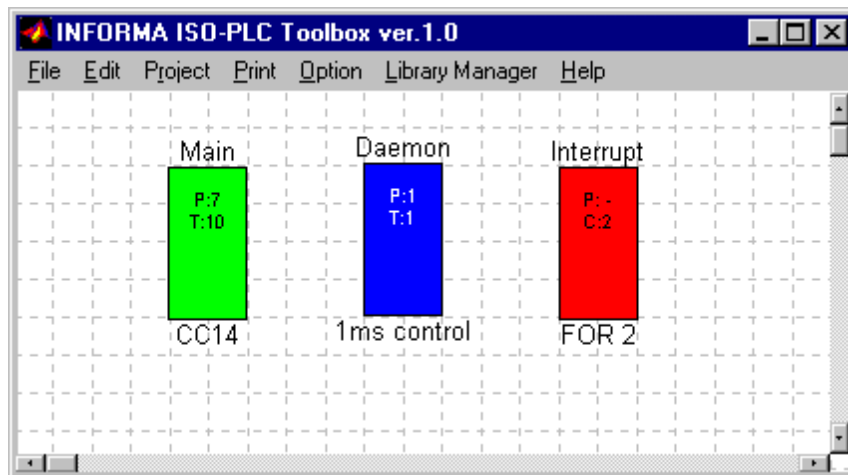
It should be noted that  $ix_{act}$  is the direct-axis stator current component in the special reference frame, and thus is collinear with the magnetizing flux-linkage space vector, while the  $iy_{act}$  is collinear with the electromagnetic torque of the machine.

By utilizing the inverse of the transformation matrix, the obtained two-axis stator current references ( $id_{act}, iq_{act}$ ) are transformed into the three-phase stator current references by the application of the three-phase transformation indicated by the block containing the name rotating axis  $\rightarrow$  fixed axis. The reference stator currents are compared with their respective monitored values, and their differences are fed into the respective stator current controllers, which are PI-controllers. The output signals from these current controllers are used to generate the firing pulses of the cycloconverter which supplies the salient-pole synchronous machine.

Similarly, the reference of the field current and the actual value of the field current are compared and their difference serves as input to the exciter current regulator. This supplies the necessary signal to a controlled three-phase bridge rectifier.

This control structure can be described in terms of a set of task where a single task is furthermore composed of a suitable collection of sessions.

In figure 7 the main window of the INFORMA ISO-PLC editor and the set of tasks constituting the control structure are visualized.

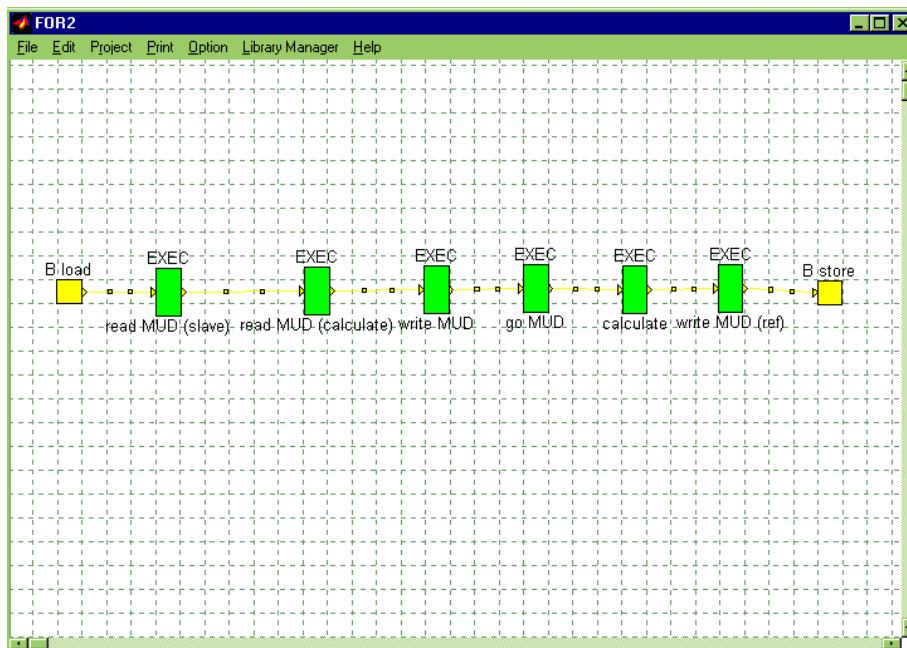


**Figure 7: The main specification window for the example**

Double-clicking on a task icon the related session is opened allowing to walk through the hierarchical structure of the specification. In this specification is possible to recognize:

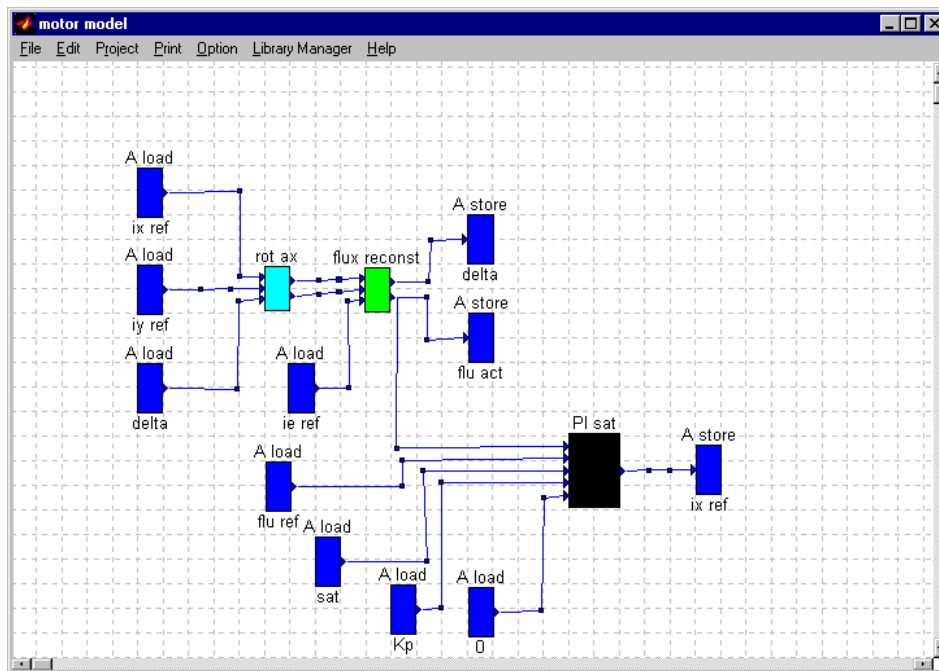
- the main task (*CC14*) with a period of 10 ms
- a periodic daemon task (*1 ms control*) with a cycle of 1 ms
- an interrupt routine (*FOR 2*) related to the interrupt source named 2

Let us consider the interrupt routine. The related session is illustrated in figure 8



**Figure 8: The session related to the interrupt routine FOR2**

This session is described in terms of a sequence of 6 sessions executed one after the other. Exploding the session *motor model* the diagram of figure 9 is obtained: in this session user and standard blocks are used together to define the control logic for flux reconstruction and control.



**Figure 9: The session "motor model"**

## Conclusions

In this paper the main characteristics of the INFORMA project have been presented together with their relation with the IEC1131-3 standard. The advantages of an integrated environment for editing, simulating, and verifying the integrated controller and the plant specification are illustrated through a simple example.

## Acknowledgement

The authors would like to thank all the partners involved in the INFORMA project and in particular Carla Penno (Ansaldo Ricerche) and Ezio Cosatto (Ansaldo Sistemi Industriali) for their support in the description of the example reported in the previous paragraph.

## References

- [1] L. Baresi. *Formal Customization of Graphical Notations*. PhD thesis, Dipartimento di Elettronica e Informazione – Politecnico di Milano, in Italian (1997)
- [2] L. Baresi, A. Orso, and M. Pezzè. Introducing Formal Methods in Industrial Practice. In *Proceedings of the 20th International Conference on Software engineering*, pp 56-66. ACM Press (1997)
- [3] E. H. Durr and N. Plat. VDM++ Language Reference Manual. Technical Report, IFAD – The Institute of Applied Computer Science (1995)
- [4] C. Ghezzi, D. Mandrioli, S. Morasca and M. Pezzè. A unified High-Level Petri Net Model for Time-Critical Systems. *IEEE Transaction on Software Engineering*, **17**(2): 160-172 (1991)
- [5] IEC. Part 3: Programming Languages, IEC 1131-3. Technical Report, International

- Electrotechnical Commission - Geneva (1993)
- [6] The MathWorks Inc. Matlab 5.2 URL: <http://www.mathworks.com/products/matlab/> (1998)
  - [7] T. De Marco. *Structured Analysis and System Specification*. Prentice Hall (1978)
  - [8] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, **77**(4): 541-580 (1989)
  - [9] O. Faergemand and A. Olsen. Introduction to SDL-92. *Computer Networks and ISDN Systems*, **26**:1143-1167 (1994)