MLAV. Object-Oriented Methodology for the Analysis and Modelling of the Control Logic of Discrete Event Systems

Víctor M. González, *Member, IEEE*, Felipe Mateos and Amos Ng

***Abstract*—This paper shows an introduction to the basic characteristics of MLAV *(Methodology of the Virtual Automation Laboratory)*, an object-oriented methodology for the analysis and modelling of discrete event systems applied to the development of the control logic based on the IEC 61131-3 standard.**

***Index Terms*— Control of Discrete Event Systems, IEC 61131-3, Object-Oriented Analysis and Modelling, PLC Control Logic Development.**

I. Introduction

THERE are two different actors that take part in the development of the control logic with an automation project: *the customer and the engineer.*

The customer indicates the way he wants the process to be controlled, and the engineer develops the necessary control logic in order to control the process the way it is intended to.

The problem is that these two actors talk different languages. The customer express himself in natural language referring to the components of the process and the way they are expected to behave. Whereas the engineer writes the control logic in the formal language of the static and dynamic diagrams that are difficult to understand by the customer as shown in Figure 1.

This fact leads to a number of misunderstandings [1] that if they are not corrected will appear as errors in the commissioning phase of the project where they are very expensive to correct [2].

On possible solution could be to define a common language that allows both actors easily understanding each other. It should be, on one hand, a formal language that allow the engineer to formally represent the control logic, but at the same time, it should be a very intuitive language so the customer is able to read and understand those diagrams in an easy way.

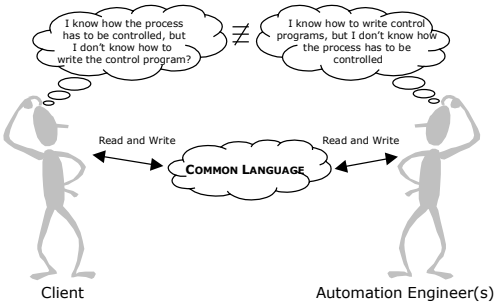

Figure 1: Communication Problems

Using this new language, both actors will be able to interact more thoroughly all along the development of the project, reducing the number of misunderstandings, and therefor the number of errors and the final cost of the project.

The proposed language is an object-oriented methodology called **MLAV** *(Methodology of the Virtual Automation Laboratory),* which has been defined under a PhD thesis ([3]) and whose main characteristics are shown in the following section.

Finally, the pros and cons are analysed from several points of view in a reasoned way, in the conclusion section.

II. MLAV

A. Introduction

MLAV is an object-oriented methodology to analysis and modelling the control logic of discrete event systems. MLAV is designed to help only with the phase of control logic development. This phase is one of the number of phases carried out within the control system design of an automation project, which includes, as it is stated by [4], several different phases such as: elaboration of offers, process components selection, exploitation and supervision components selection, commissioning, etc.

Following the definition of the term *"methodology"* introduced by David Harel [5], MLAV has been built over four main elements: *the concept, the notation language, the procedural method and the software tool.*

This work was supported in part by the Deparment of Education of the Principado de Asturias under Grant PC-CIS01-24, and by the University of Skövde (Swden) under research contract HF 5Kap, 1-7.

Víctor M. is with the Department of Electrical, Electronics, Computer and Systems Engineering, University of Oviedo, Campus de Viesques, 2.1.2, 33204 – Gijón, Spain (phone: +34 985 18 19 64; fax: +34 985 18 20 68; email: victor@isa.uniovi.es).

Felipe Mateos is with the Department of Electrical, Electronics, Computer and Systems Engineering, University of Oviedo, Campus de Viesques, 2.1.2, 33204 – Gijón, Spain (felipe@isa.uniovi.es).

Amos Ng is with the Department of Science Engineering, University of Skövde, PO Box 408, 541 24 Skövde, Sweden (amos@ite.his.se).

B. Basic Concept

The basic concept over which MLAV is built is the simulation of the process. It is demonstrated ([6]) that the simulation of the process can be the natural meeting point for the actors that take part in the control logic development to exchange information about the way the process has to behave (see Figure 2).

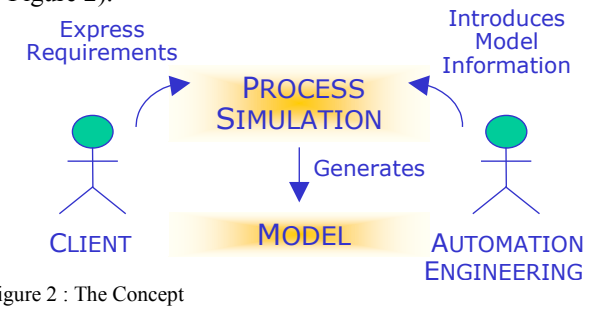

Figure 2 : The Concept

MLAV suggests that the engineer analyse the system from 4 different points of view:

1) Static

From the static point of view, the engineer should think about the **"Who"**. "*Who are the objects that will make up the model?"* is the basic question the engineer has to ask himself at this point. Further on, it will be stated how to get those objects.

2) Functional

From the functional point of view, the engineer should think about the **"What"**. "*What is the responsibility every object has to fulfill?"* and *"What is the functionality (attributes and services) every object will provide in order to meet this responsibility?"* are the two basic questions the engineer has to ask himself at this point.

3) Dynamic

From the dynamic point of view, the engineer should think about the **"When"**. "*When will every service of every object be activated?* This is equivalent to think about how every object behaves internally and how it interacts with other objects.

4) Structural

From the structural point of view, the engineer should think about the **"How"**. "*How to organise the final model in a coherent way?* MLAV suggests a hierarchical, recursive and scalable way to organise the model as shown in Figure 3.

C. Notation Language

The notation language proposed by MLAV is a combination of three different ones: UML ([7]), GEMMA ([4]) and GRAFCET ([8]).

On one hand, MLAV is defined as an object-oriented methodology. A subset of UML has been chosen to represent the object-oriented aspects of the model generated using MLAV, because UML is a broadly accepted standard that can be used to represent classes, objects and their relationships. Also the functioning requirements expressed by the customer can be represented in a very intuitive way.

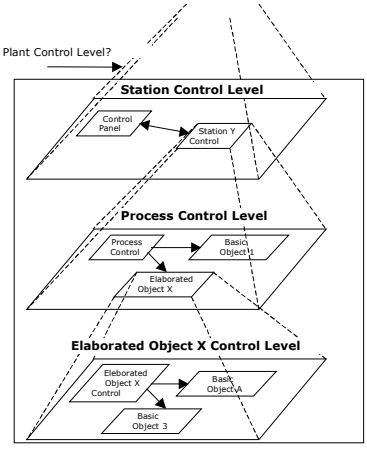

Figure 3: Model Structure

On the other hand, MLAV is designed to analysis and model discrete event sequential systems. The modelling languages that UML provides to represent such information are not very intuitive and are not broadly accepted by the community of automation engineers, potential users of MLAV. However, GRAFCET is a modelling language specially designed to model sequential systems broadly accepted amongst those engineers.

Finally, although GEMMA is not widely spread, it is a very powerful and intuitive tool to help defining the different possible states at which the process may eventually be during its life (see Figure 15).

D. The Method

MLAV proposes an iterative and incremental method divided into several phases as shown in Figure 4.

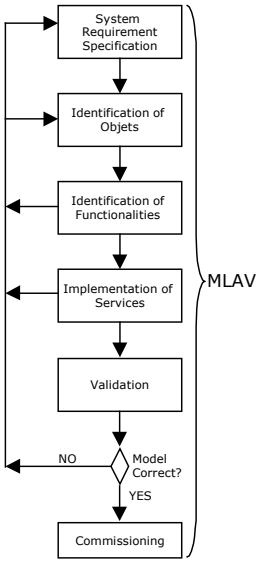

Figure 4 : Phases of the Method of MLAV

It is proposed to start the analysis and modelling with a subset of the system requirements expressed by the customer. After the first iteration is over, the subset can be enlarged with more requirements that will conduct to a new layer in the model. When all the requirements have been modelled, the model is done.

1) System Requirements Specification

During this phase the dialog between the customer and the engineer has to be very intense in order to completely allow this one understanding the way the process has to behave.

MLAV proposes that the model be represented in three formal languages:

a) GDMMA

GDMMA *(Descriptive Graphic of the Modes of Start and Stop)* is a graphical representation of the different states the process may eventually be in during its life. To build a GDMMA, MLAV suggests that the engineer take the GEMMA template and for each of the states depicted in it, ask the customer if it could be possible that the system be eventually in it. If this state is not viable, then it can be removed from the template, and all the lines that connect this state with the others. (See Figure 16).

b) Use Case Diagram

Additional information can be represented using the use case diagram language defined in UML. This diagram allows the engineer to complete the GDMMA with information about the actors and their interaction with the system.

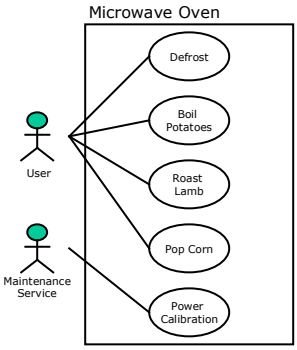

Figure 5: Use Case Diagram

c) Event Flows

For each use case in the use case diagram, MLAV proposes that the engineer represent how would be the sequence of events that will occur when the use case is executed.

This information can be represented in natural language as an algorithm or using the more formal notation language GRAFCET at level I.

2) Identification of Objects

Once the customer and the engineer have reached an agreement about the way the system has to be controlled, is the moment to identify the objects that will make up the model of the control logic.

MLAV suggests that the engineer apply Martin Fabian's theories [9] stated in the CHAMP reference architecture [10] developed at Chalmers University (Sweden).

As it is shown in Figure 6, for every object of the process, called *"External Object"*, there must be an object in the control logic, called *"Internal Object"*. All these objects will be coordinated and sequenced by a *"Controller"* object not present in the process, in order to fulfill the behavior requirements stated before.

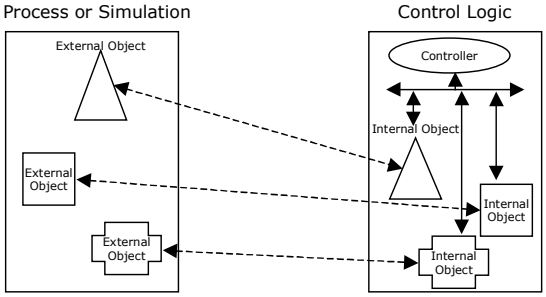

Figure 6: External/Internal Objects

MLAV establishes two kinds of objects:

- *Basic Objects* are not divided into more objects. They carry out primitive actions. Sensors and preacturators/actuators are examples of such objects.
- *Elaborated Objects* are made up of basic and/or elaborated objects. They carry out non-trivial actions.

This information has to be represented by means of two formal languages:

- *Class Diagrams*. Using this language taken from UML, the engineer can represent the different classes of objects and their relationships.
- *CRC Cards,* first introduced by Kent Beck and Ward Cunningham ([11]), is an intuitive technique to represent the responsibility of a class and its collaborations with other classes in order to fulfill it.

3) Identification of Functionalities

MLAV defines *"functionalities"* as the set of attributes that characterize a class and the services it provides to fulfill its responsibility.

MLAV suggests that the engineer represent this information by means of two new formal languages defined within MLAV:

- *CRC+F Cards* is an extension to the CRC language. It allows representing in a card the attributes and services that characterize a class.
- *Publication-Subscription Diagrams.* This language allows the engineer to represent the contractual relationships that exist between two classes. When a class X needs to use an attribute or service of another class Y, you can say that the class X is *subscribed* to those attributes and services of class Y. To do so, it is necessary that class Y has previously *published* those attributes and services. MLAV suggests that this relationship be represented using a special symbol as shown in Figure 7.

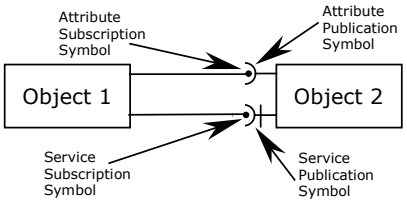

Figure 7: Publication-Subscription Symbols

4) Implementation of Services

MLAV pursues a model that is independent of the technology and the type of control equipment finally used to execute the model. Besides, MLAV is focused on the modeling of discrete event sequential systems.

Finally, GRAFCET is the more widely spread and accepted modeling language amongst control engineers. It is independent of the technology and the type of control equipment, and it has been specially designed to model sequential systems.

Because of all these reasons, MLAV suggests to use GRAFCET to implement the services in two phases: *specification of sequences and implementation of sequences*.

a) Specification of Sequences

MLAV indicates in this phase to roughly represent the skeleton of the sequences over the previously developed publication-subscription diagrams by means of a sequence of numbers.

This is not a very powerful language but allows the engineer softly transitioning from an intuitive language (easy to understand by the customer) to a more formal specification of sequences in GRAFCET (not so easy to understand by the customer).

b) Implementation of Sequences

In this phase, MLAV suggests to use GRACET at level II to fully represent the sequence that implements a service based on the skeleton previously depicted.

MLAV produces object-oriented models, whereas GRAFCET is a non object-oriented language. This imposes an added difficulty. To solve this drawback, MLAV suggests to extend GRAFCET with the operator "**.**". Doing so, it would be possible to access any attribute or invoke any service of any object from within an action or transition represented in GRAFCET as shown in Figure 8.

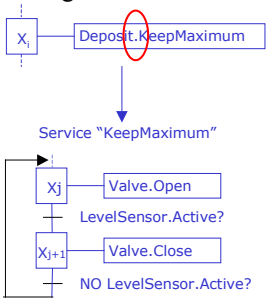

Figure 8: Operator "." in GRAFCET

5) Validation

Once the model is complete, that is, all services of all objects have been implemented it is necessary to verify if the control system behaves in the way expressed by the customer.

To do so, MLAV suggests carrying out a visual validation against the simulation of the process in three phases, before transitioning into commissioning:

- *Virtual Phase*. In the first phase, MLAV suggests to simulate the execution of the model of the control logic, in a real-time connection to the simulation of the process. Doing so, the engineer will be able to make quick modifications to the control logic until it works as stated by the customer.
- *Medium Phase*. Secondly, MLAV suggests replacing the simulation of the execution of the control logic, by a real programmable control equipment in a real-time connection to the process simulation. This will allow the engineer concentrating in adapting the control logic to the peculiarities of the control equipment, knowing that the control logic works fine.
- *Real Phase*. Finally, MLAV suggests substituting the process simulation by the real process. Doing so, will allow the engineer to pay attention to the tuning of the process, knowing that the control logic is correct.

Once these phases are over, you can say the control logic has been validated and the commissioning has been carried out.

6) Conclusion

MLAV defines a procedural method that allows the engineer to automatically generate a model of a control logic from the specification of the functioning requirements stated by the customer.

However, it is necessary to define a software tool that facilitates the systematical application of this method.

E. The Software Tool

MLAV gives the name *LAV (Virtual Automation Laboratory)* to the software tool that helps systematically applying the methodology.

MLAV defines the generic term *CACLE (Computer Aided Control Logic Engineering)* to identify those tools that help developing control logics to control processes.

This CACLE tool is analysed from two points of view in the following sections.

1) Functional Characteristics

From the functional point of view, LAV has to provide the engineer with a set of instruments to support the development of a model of a control logic. More precisely, it has to support the basic concepts, the notation and the method as shown in Figure 9.

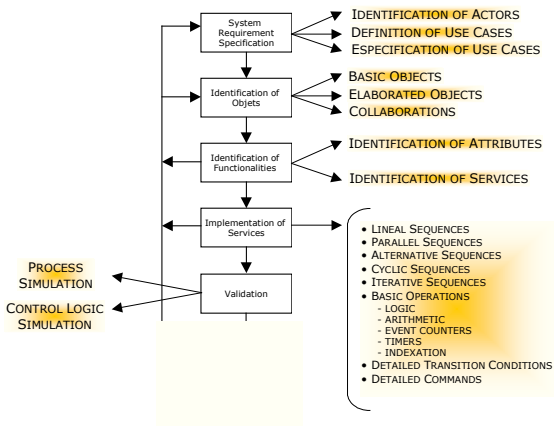

Figure 9: Functional Characteristics

This means that LAV should provide with some instrument to easily allow the engineer:

- identifying which actors will interact with the system, typically maintenance service operators;
- specifying what are the possible use cases of the system and their specification in natural language and GRAFCET level I;
- identifying what are the objects, basic objects, elaborated objects, and their collaborations;
- defining what attributes and services each object provide;
- implementing every service's sequence;
- simulating the process;
- simulating the execution of the control logic by a programmable controller.

This set of functionalities determines the architecture of the software tool.

2) Architecture

MLAV suggests that LAV's architecture be open and scalable based on a client-server structure as shown in Figure 10.

MLAV focuses on the control logic development phase of an automation project. Because of that, only the architecture of those components needed to help with this phase, i. e. the server, the process simulator and the control equipment simulator, is defined in the following sections. However, with such architecture, LAV could become a true *VEE (Virtual Engineering Environment)* as defined by Dr. Grübel ([12, 13]) by adding new clients to help with other phases of an automation project: such as supervisory control and exploitation, estimates, reports, documentation, cabling schemes, etc.

a) The Server

The LAV's server module should provide a set of services allowing to main objectives:

- The interaction of the different clients of the VAL, e. g. exchanging signals and events, etc.
- The automation project's information integrated and centralized management. This will foster the vision of an automation project as a *"whole"* made up of several different interconnected phases, and not as it is seen nowadays, as a set of independent tasks, whose results have to be integrated to fulfill the automation goal.

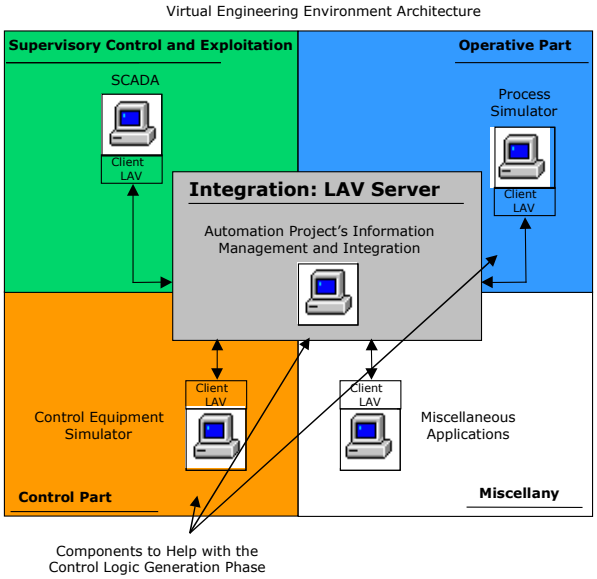

Figure 10: Architecture of LAV

b) Process Simulator

MLAV suggests that the process simulator be made up of three different modules as shown in Figure 11.

- The Edition module should provide the user with the necessary tools to allow the user specifying the actors, the use cases and the flow of events for each use case. Besides, it should also provide with a library of basic and elaborated objects that will finally make up the control logic and the plant to be simulated.
  This library should be extendable by means of defining new objects, that is, by defining the attributes and services that characterize them.
- The Control Logic Generation module should provide the user with the necessary tools to specify the collaborations or connections between objects, that is the way they will exchange information to fulfill their responsibility. Finally, it should also provide the user with a way to edit the sequences that will implement each of the object's services. (In [3] the Guidance method is proposed as a way to capture that information).
- The Execution module should allow the user simulating the process. This will allow validating the control logic against the process simulation by means of a real-time connection provided by the LAV server.

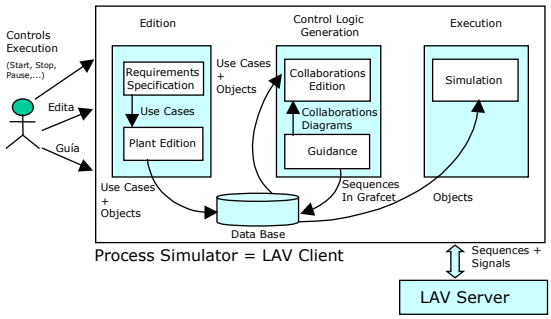

Figure 11: Process Simulator Architecture

c) Control Equipment Simulator

PLCs *(Programmable Logic Controllers)* are the most used programmable control devices in the process automation discipline. However, there has been a historical incompatibility at hardware and software level among these devices that has seriously fragmented the market. There has been several intents to standardize those devices ([14], [15] and [16]) from the hardware and software point of view. The standard IEC 61131 proposed by the IEC (International Electrotechnical Commission) is becoming a de-facto standard. It is made up of several parts. Part III is dedicated to the programming languages. It defines a set of common structures as shown in figure Figure 12: Common ElementsFigure 12, such as: global variables, several different program organization units, a way to execute those units, etc., and five programming languages that must be used to write the program organization units.

This part of the standard is very well explained by R. Lewis in his book [17].

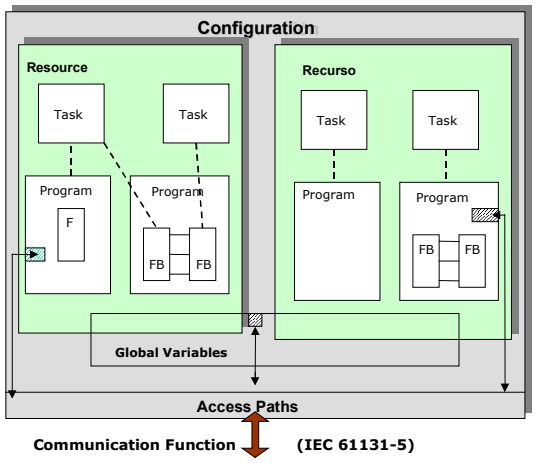

Figure 12: Common Elements

On one hand, MLAV suggests produces models that are independent of the technology and the type of control equipment finally used to control the process.

On the other hand, the PLCs are the most common control equipment used nowadays in the process automation discipline, and IEC 61131-3 defines an independent hardware and software PLC architecture.

So, MLAV suggests that the architecture of the control equipment simulator be based on the IEC 61131-3 model. To do so, MLAV suggests that this module have an architecture like the one shown in Figure 13.

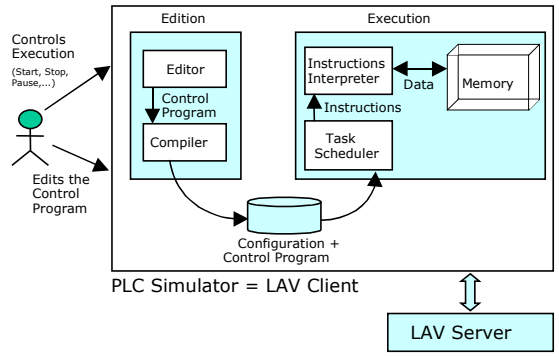

Figure 13: PLC Simulator Architecture

MLAV suggests that the PLC simulator be made up pof two different modules:

- The Edition module should provide the user with the standard mechanisms present in any common editor such as copy, past, save,... for the five languages of the standard IEC 61131-3. This module should also provide the user with a compiler that will allow him checking the code for errors and generating an intermediate code that finally will be executed by the execution module.
- The Execution module will interpret the instructions that make up the different task in the way the user has specified they should be executed. This will eventually produce an interaction with any external components that is a LAV's client.

  This architecture will allow validating the control logic against the simulation of the process by means of the real-time connection provided by the LAV's server.

d) Conclusion

The LAV provides a CACLE tool that helps systematically applying the MLAV methodology by providing a set of tools that supports all the phases of its method, its notation language and its concepts.

III. Conclusion

MLAV defines a methodology that allows the engineer to automatically generate a model of a control logic from the specification of the functioning requirements stated by the customer.

Using MLAV has some pros and cons that can be analyzed from several points of view:

1) Models Assimilation

Because MLAV produces object-oriented models organized in a hierarchical manner, this leads to a kind of standardization of the models what helps understanding them by third engineers that didn't participate in their development, as the number of models studied increases, as shown in Figure 14.

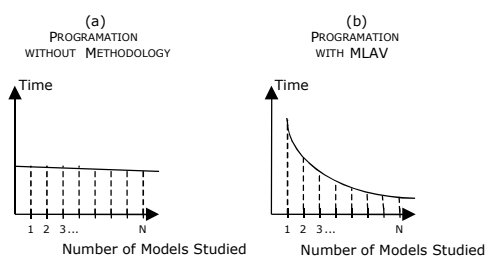

Figure 14: Models Assimilation

However, and because of the number of characteristics of MLAV, it could be more difficult to understand the models produced by MLAV than those obtained when no methodology is used.

2) Control Programs Generation

MLAV allows reducing the development costs by:

- Improving and facilitating the dialog between client and engineers, what leads to a reduction of misunderstandings and errors;
- Systematizing the development of control programs because of the procedural method and the CACLE tool.

However, one thing is true. The toll to pay is the generation of more documentation and the fact that a simulation of the process has to be built.

3) MLAV Main Drawbacks

Because MLAV is an object-oriented methodology users will find two problems:

- On one hand, programs will be bigger than if they were built without using any methodology. This is a problem when the final control equipment to be used is a PLC. These devices usually have few Kbytes of memory.
  On possible solution could be to use some other kind of programmable devices with more memory such as industrial PCs.
- On the other hand, programs will be object-oriented, but PLC programming languages usually are not object-oriented.
  This problem could be solved using the languages and data structures provided by the standard IEC 61131-3 as indicated in [3]. The price to pay is a more complicated code that could make it harder to understand.

REFERENCES

[1] Emerson, D., *{What does a Procedure Look Like?}*. 1999, World Batch Forum. p. 9.

[2] Bonfe, M. and C. Fantuzzi, *{Mechatronic Objects encapsulation in IEC 61131-3 Norm}*. IEEE International Conference on Control Applications, 2000: p. 598-603.

[3] González, V.M., *{"Object-Oriented Methodology for the Analysis and Modelling of Discrete Event Systems. Application to the Generation of the Control Logic based on IEC 61131-3"}*, in *Department of Electrical, electronics, Computers and Systems Engineering*. 2002, University of Oviedo: Gijón. p. 324.

[4] Moreno, S. and E. Peulot, *{Le GEMMA}*. 1997, París: Casteilla.

[5] Harel, D. and M. Politi, *{Modeling Reactive Systems with Statecharts: the Statemate Approach}*. 1998: McGraw-Hill. 258.

[6] Maclay, D., *{Simulation Gets into the Loop}*. IEE Review, 1997. **43**(3): p. 109-112.

[7] Powel Douglass, B., *{Real-time UML}*. 2º ed. Addison-Wesley Object Technology Series, ed. Addison-Wesley. 1998: Addison Wesley Longman Inc. 365.

[8] IEC, *{Preparation of Function Charts for Control Systems. Nº 848}*. 1988, International Electrotechnical Commission: Geneva.

[9] Fabian, M., *{On Object-Oriented Non-deterministic Supervisory Control}*, in *Control Engineering Laboratory*. 1995, Chalmers University of Technology: Göteborg.

[10] Adlemo, A., et al., *{Models for Specification and Control of Flexible Manufacturing Systems}*. 1997, Chalmers University of Technology: Göteborg. p. 179.

[11] Beck, K. and W. Cunningham, *{A laboratory for teaching object oriented thinking}*. ACM SIGPLAN Notices, 1989(10).

[12] Grübel, G. *{Perspectives of CACSD: Embedding the Control System Design Process Into a Virtual Engineering Environment}*. in *IEEE International Symposium on Computer Aided Control System Design*. 1999. Kohala Coast - Island of Hawai'i, USA.: IEEE.

[13] Grübel, G., *{Email conversation. (Conversación por correo electrónico).}*, V. González, Editor. 2002: Gijón.

[14] OSACA, *{Open System Architecture for Controls within Automation Systems. Osaca I AND II Final Report.}*. 1996, OSACA Association: Stuttgart. Germany. p. 96.

[15] Sousa, M.D., *{Linux-based PLC for Industrial Control}*. Embedded Linux Journal, 2001(3).

[16] IEC, *{Programmable Controllers - Part 3: Programming Languages}*. 1993, International Electrotechnical Commission: Geneva.

[17] Lewis, R., *{Programming Industrial Control Systems using IEC 1131-3}*. IEE Control Engineering Series 50, ed. P.P.J. Antsaklis, P.D.P. Atherton, and P.G.W. Irwin. 1995, Herts, U.K.: The Institution of Electrical Engineers, London. 293.

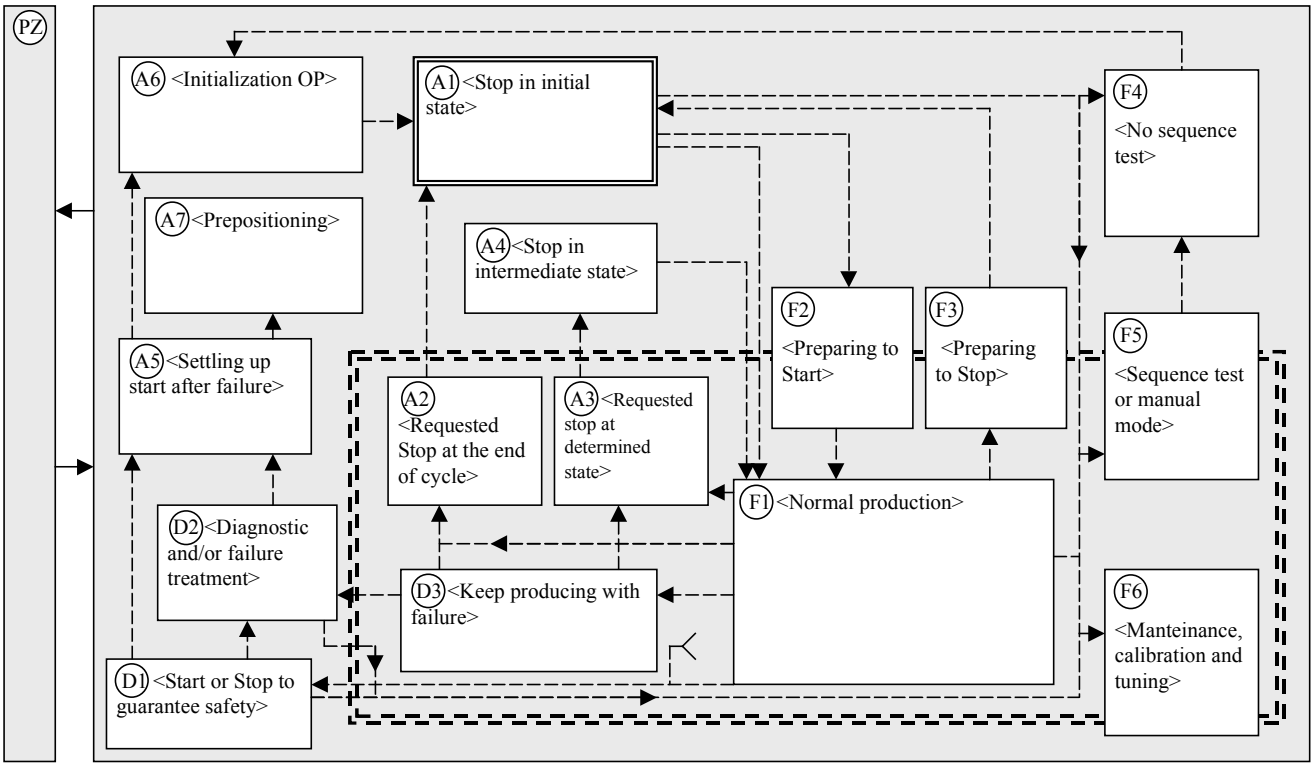

Figure 15: GEMMA Template

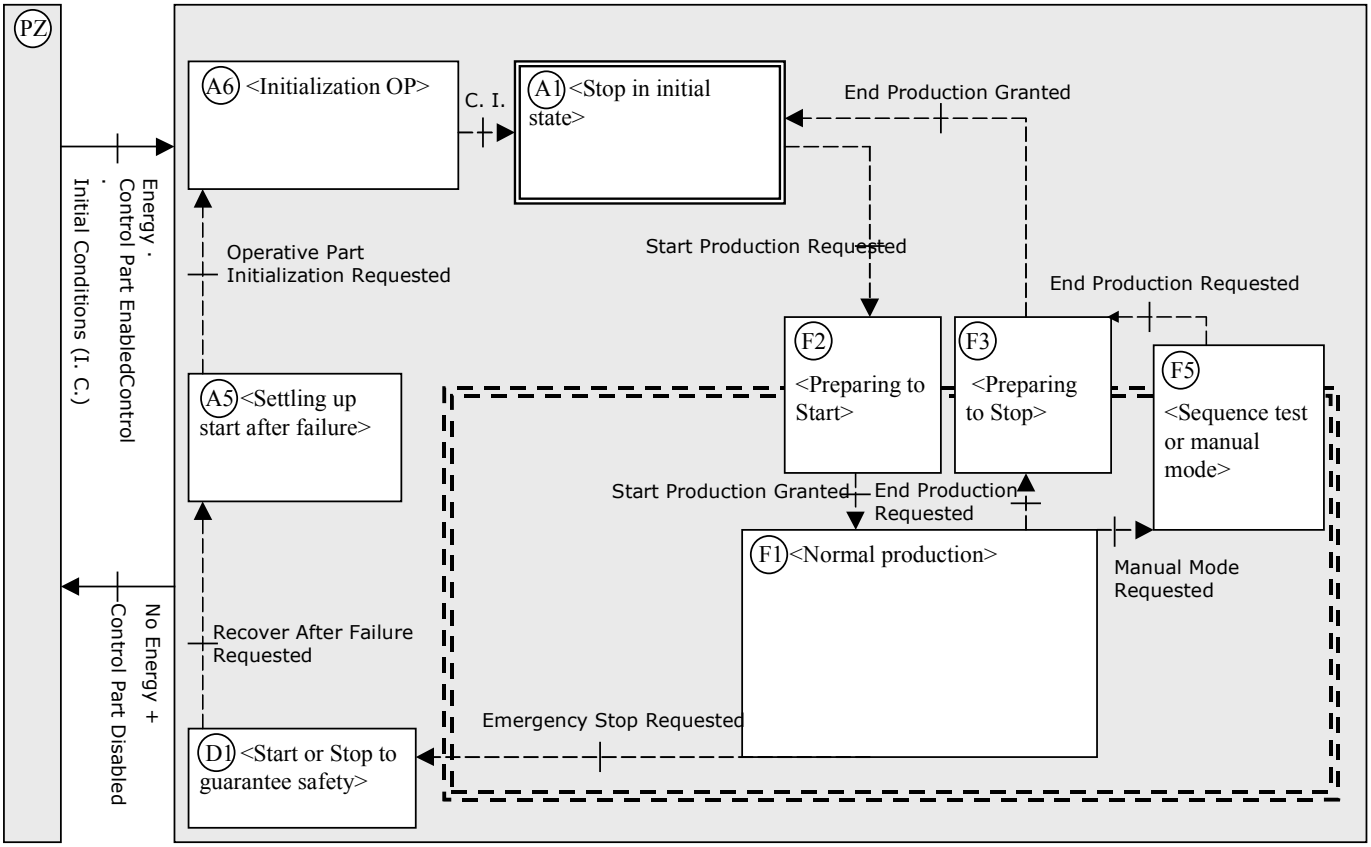

Figure 16: GDMMA Example