Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online
CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 48-10 (2015) 178–183

A PackML-based Design Pattern for Modular PLC Code

Giacomo Barbieri * Nicola Battilani * Cesare Fantuzzi *

* University of Modena and Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia (Italy) (e-mail: {giacomo.barbieri, nicola.battilani, cesare.fantuzzi}@unimore.it).

Abstract:
Software is exponentially growing in modern automatic machines. Consequently, the operations of writing, debug and maintenance are increasing the time necessary to be accomplished. In order to shorten this time, there is the need to define methodologies for software which enhance modularity, along with re-usability and standardization. This work collects the needs of several industries working in packaging domain for PLC code writing, proposing a solution which overcomes the limits of other well known approaches defined in literature: Object-Orientation and Model Driven Engineering. The proposed solution is a design pattern based on the PackML. This can be considered as an introductory work about this topic. We hope that the scientific community will increase the researches, in order to standardize and facilitate the software writing which dimensions are becoming an issue above all for Small and Medium Enterprises (SMEs).

Keywords: PackML, Design Pattern, PLC Code, Modularity, Reuse.

1. INTRODUCTION

Programmable Logic Controller (PLC) software has a fundamental importance in industrial automation systems, and there are numerous evidence that their complexity is growing. For example, the German Engineering Federation VDMA has presented the growing ratio of software development in the costs of machinery, Stetter (2011). The ratio of software has doubled in one decade from 20% to 40%. If this trend continues, the main activity of Original Equipment Manufacturers (OEMs) will be software engineering.

Among the world of software for automated machines, there is a plethora of different approaches for PLC programming. As Bonfatti et al. (2001) states, the market fragmentation in small teams with radicated development methods and tools, the dependancy of software quality on the skills of the programmers and the lack of documentation are the main traits that can describe the environment we are playing in. Moreover, automation systems are continuously being required increasing capability at reduced costs and within shorter delivery cycles. In order to reach these requirements, PLC software programming should be improved in the direction of modularity, applying a *Divide et impera* policy to everything that needs to be coded and validated. Furthermore, the developed modules need to be reusable in different projects: different arrangements of them may constitute the software of different PLC-based machines and plants, with the advantage of saving time in the development and validation phases. However, pieces of software coming from different development teams are rarely easy to integrate, because different conventions are used along with interfaces etc. Therefore, efforts are required to make them working together in the same application and to understand and maintain the code. Time and efforts increase with the number of modules and the different internal conventions. This is the reason why a widely supported and implemented standard would be an important step in the objectives listed above.

This paper proposes to adopt the Packaging Machine Language (PackML OMAC (2006)). This is a "common approach" for automated machines but not yet a standard. In fact, there is not an unambiguous and unique interpretation of its tools. The main reason why this approach is getting interest is the simplicity that comes for performance analysis (see Loughlin (2003)). Moreover, it makes different machines "speak the same language", facilitating the line controller programming and inter-machine communication. For these reasons, customers have started requiring PackML compliant machines.

According to our experience, PackML is usually implemented mapping a *Non-PackML* software – that is meaningful only for the team that developed the code – into the conventional set of states coming from PackML. This leads to several counterparts. First, PackML states are arbitrary interpreted without a common view. This creates issues when PackML machines from different OEMs are assembled in a line. Second, the mapping of the current state of a Non-PackML software into a PackML one is often computationally expensive and may introduce errors.

The objective of this paper is to propose an interpretation of the PackML tools and to introduce a design pattern which results in a software completely PackML based and compliant, in order to reach modularity, along with reusability, maintainability and understandability.

This work is organized as follows: related works are described in Sec. 2. Sec. 3 resumes PackML mission, while

Sec. 4 introduces a PackML library for the Rockwell platform. Sec. 5 illustrates the proposed design pattern for the Rockwell environment. Eventually, results and future works are reported in Sec. 6.

2. RELATED WORKS

IEC 61131-3, International Electrical Commission (2013), which was first published in 1992, defines a model and a set of programming languages for writing PLC control code. Since its introduction, many software design methodologies have been developed. Vyatkin (2013) illustrates the software engineering state of the art in the industrial automation sector that spans from manufacturing factory automation to process control systems and energy automation systems. Among the presented approaches, Model Driven Engineering (MDE), OMG (2003), has motivated researchers to exploit its benefits in IEC 61131-3 based systems. MDE is a software development methodology which exploits domain models rather than pure computing or algorithmic concepts. The three main goals of MDE are portability, modularity and re-usability.

PLCopen (2008) defines an open interface, which provides the ability to transfer Program Organization Units (POUs) between all different kinds of PLCs. AutomationML, Drath et al. (2008), is a neutral data format used for interconnecting information contained in different engineering tools and implements PLCopen interface for the software part. These standards act on portability, but currently this is not a fundamental requirement for most OEMs. In fact, they rely on fixed PLC vendors, and seldom change suppliers.

Regarding modularity and re-usability, MDE proposes a higher layer of abstraction by exploiting either UML (e.g. Katzke and Vogel-Heuser (2007), Ramos-Hernandez et al. (2005), Sacha (2008)) or SysML diagrams (e.g. Chiron and Kouiss (2007)). However, as claimed in Thramboulidis and Frey (2011), these works do not achieve the Object-Oriented (O-O) aspects of IEC 61131-3 and result into inefficient mappings of UML/SysML constructs to IEC 61131-3 constructs.

O-O is, in fact, perceived as a key factor to manage software complexity, Maffezzoni et al. (1999). Encapsulation features of objects and classes prevents from incorrect use or manipulation of software parts and data, while inheritance enables software reuse and so-called design by extension, Bonfé et al. (2006). However, in our experience with industries, O-O is scarcely utilized because requires capabilities which are currently not available in operators responsible for installation and debugging of the machine in the customer site.

The need of standardization, modularity and reuse pushed us to look for something which was able to embody the benefits of MDE and O-O, according to the simplicity required by operators. In our opinion, PackML is the solution to these needs. This approach was introduced in OMAC (2006), and then was included in the ISA-88 as an extension to the packaging domain of its well-known batch process control model, ISA (2008). However, as pointed in Bonfé et al. (2013), PackML is proposed as a template/approach, rather than a standard and design pattern. The objective of this paper is to show how PackML can be used to satisfy the requirements of modern OEMs.

3. THE PACKML MISSION

PackML mission is to establish a common presentation and high level software architecture to all machines making up a complete packaging line, especially if they are produced by different vendors.

Its objective is to standardize the following parts:

(1) high level state machine (Fig. 1),
(2) mode management procedure,
(3) interfaces to the plant supervisor (e.g. MESs, ERPs, OEE analyser etc.) and among software modules.

PackML assumes that software is modular and modules are identified on the basis of the performed functionalities. The recommended modular decomposition of a production line, according to the *Machine Template* introduced in ISA (2010), is:

- *Control Module* (CM): element which, from a functional viewpoint, cannot be decomposed into granular finer parts;
- *Equipment Module* (EM): performs one or more functions necessary for achieving the machine operation through the coordination of CMs;
- *Unit Machine* (UN): performs a specific operation through the coordination of EMs.

In this document, we generally define Mechatronic Object (MO) either UN, EM or CM.

4. P&G ROCKWELL IMPLEMENTATION

Procter & Gamble (P&G) implemented ladder-based PackML function blocks, called add-on instructions (AOIs), and user defined data types (UDT) oriented towards Rockwell control systems. P&G customized PackML approach through the assignment of a category field to the events, and through the generation of lists for reporting events to higher level MOs. The P&G implementation will be illustrated in the next paragraphs. In fact, these tools will be utilized in the proposed approach.

4.1 Add-On Instructions

Rockwell AOIs utilized for the definition of the design pattern are illustrated in Table 1.

AOI_PackML: configures and controls modes and states of the MOs. Configuration parameters allow defining the MO modes, the state machine of each mode (i.e. a mode might not have all the states of the PackML base state machine), and the states in which a transition from a mode to another is allowed. Then, this AOI checks whether a request for either changing mode or state is received and provides the MO current state and mode.

AOI_CM_Events: generates an event whether the trigger conditions are true. An instance of this AOI should be created for every event. Each event is associated with an ID, a value and a message as established from the

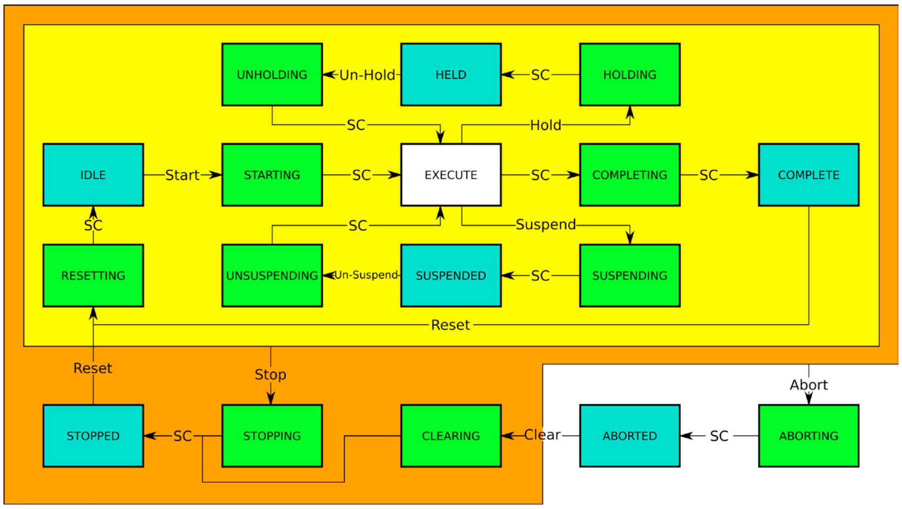


Fig. 1. Illustration of the PackML base state machine. The *wait states* in which the MO is waiting for a command are represented with the cyan color; the *acting states* in which the MO is executing the command are shown in green; in white, the *dual state* Execute that is both a *wait* and an *acting state*.

.

AOI	**Configuration Tags**	**Input Tags**	**Output Tags**
AOI_PackML	Modes definition, Modes state machine, Modes transition	Command change mode, Command change state	Current state, Current mode
AOI_CM_Events	Message, ID, Send to (Event Manager), Value, Category	Trigger Conditions	EventActive, Event
AOI_Event_Manager		Active Events	MO_list
AOI_EventSummation		MO1_list, ..., MOn_list	MOs_list

Table 1. P&G Rockwell-based PackML AOIs used in the proposed design pattern.

PackML approach, but also a category which is introduced in the P&G implementation. The category enumerates the event from 0-9, for grouping or prioritizing events for alarm handling and reporting purposes. When the event is generated, it is automatically sent to the established event manager and the BOOL EventActive is set to true.

AOI_Event_Manager: collects event information from the different AOI_CM_Events and creates a unique list of events. List structure is explained next, in the UDT_EventStatus compartment. Only one instance of this AOI should be created (per MO) for each event type; for example one for alarms, one for warnings etc. The number of event categorizations which can be represented is consequently defined as the number of categories (0-9) multiplied for the number of AOI_Event_Manager (arbitrary decided by the user).

AOI_Event_Summation: summarizes the output data of the AOI_EventManager(s) and creates a unique list. As for the AOI_Event_Manager, only one instance of this AOI should be programmed for each event type. For example, three EMs have (each one) a Warning AOI_Event_Manager which respectively generates a list called EM1_WarningList, EM2_WarningList and EM3_WarningList. This AOI, utilized at the UN level, might merge the three lists by creating the list UN_WarningList.

4.2 User Defined Data Types

Input, output and configuration tags of the AOIs described in Sec. 4.1 need particular data types in order to contain all the necessary information. UDTs necessary for the illustrated AOIs are described in Table 2.

UDT_PackML: this typed tag is utilized for the communication with other hierarchy levels and as input of the AOI_PackML. It consists of three parts:

- Cmd_i-th: array of BOOL which contains all the commands of the PackML state machine (e.g. Cmd_Reset, Cmd_Start etc.);
- j-th_SC: array of BOOL which indicates that the condition of state complete has been reached in a certain state (e.g. Resetting_SC etc.);
- Cmd_Mode: DINT which commands to execute a transition to the indicated mode.

UDT_Event: data type which contains the information of a particular event: ID, Value, Message and Category.

UDT_EventStatus: structure used to contain the lists generated by the AOI_EventManager and AOI_ EventSummation. This tag indicates the first out event, the active events (maximum 100 events), the active categories of events and a BOOL for resetting the list.

4.3 Conclusion

In the previous sections, the PackML mission was shown (Sec. 3), along with available tools for its application (Sec. 4). In Sec. 5, a design pattern based on these concepts is introduced. The approach has been defined for the Rockwell environment, but may also be applied into other PLC platforms. In that case, one should either use the

UDT	Fields
UDT_PackML	Cmd_i-th (BOOLs), j-th_SC (BOOLs), CMD_Mode (DINT)
UDT_Event	Message (STRING), ID (DINT), Value (DINT), Category (DINT)
UDT_EventStatus	First Out Event (UDT_Event), Active Events (UDT_Event[100]), Category_k-th_Active (BOOLs), Reset List (BOOL)

Table 2. Rockwell-based PackML UDTs. BOOLs indicate an array of BOOL.

tools provided from the chosen PLC vendor or create a library which implements the PackML approach.

5. THE PACKML-BASED DESIGN PATTERN

Before adopting the proposed design pattern, a deep conceptual study must be performed in order to:

- *Subdivide the machine in functional EMs*: the defined modules will constitute software POUs implemented in PackML. The company must establish criteria for an efficient decomposition (e.g. in terms of re-usability, modular testing etc.).
- *Identify all the possible failures and their handling procedures*: every machine has hardware and software parts for detecting internal and external anomalies. The consequences of anomalies are different and these different behaviours should be addressed into the PackML base state machine.
- *Define MO operative modes and necessary procedures for changing mode*: automatic machines may present different behaviours which are defined as modes, ISA (2008); for example automatic, manual, cleaning, maintenance etc. A mode management procedure must be identified which establishes how, and in which states, a MO may change mode. Specification on transitions between control modes is left to the user, but typical transition points are at wait states.

Next, the design pattern is explained.

5.1 The centralized control

The centralized control proposed from Fantuzzi et al. (2011) is utilized. In this, every object supervises and synchronizes its child objects. For this reason, communication is allowed among different hierarchy levels, but not among objects of the same level. A PLC-program is created (per MO) for defining the behaviour of the UN and the EMs. The CMs consist of actuators and specific functions (e.g. a routine which performs calculations), so there is not the need to consider them as modular and independent objects.

5.2 PackML state machine interpretation

PackML base state machine (Fig. 1) is characterized from states in which actions that bring to certain conditions are performed (i.e. the ones ending for "-ing"), and others which are reached once certain conditions are verified (i.e. through the State Complete tag, SC). For example, a Suspend command is received. The MO goes in the Suspending state in which a controlled stop is performed. When the MO is stopped, it generates a SC command and reaches the Suspended state.

PackML state machine must formalize typical behaviours and phases of automatic machines. Next, our interpretation is provided.

Nominal cycle: during a working cycle, every MO presents the following phases:

- Initialization phase: the MO starts its cycle in the Stopped state and when is turned on the initialization operations are performed in the Resetting state (e.g. motors powering and homing etc.). Then, the Idle state maintains the conditions which were achieved during the Resetting state, until a Start command is received.
- Production phase: the steps needed to start the MO are performed in the Starting state. Then, the MO remains in the Execute state in which the production operations are performed;
- Mission complete: the MO has completed its mission and is stopped through the Completing state. Then, it remains in the Complete state until a Reset command is received and a new working cycle is started.

Alarms handling: in automation domain, alarms are generally classified according to safety regulations into conditions for Normal Stop, Fast Stop and Safety Stop.

- Normal stop: is the consequence of faults considered without particular gravity which occurs during the Execute state (i.e. production phase). The stop is controlled and actuators do not lose synchronism. In this way, MO can re-start from the Execute state once handled the responsible fault. It can be requested either through the Hold or Suspend command. When an internal equipment fault is detected or after an operator request, an Hold command should be generated. Whereas, faults which come from outside the MO provoke the Suspend command generation; for example, starvation of upstream material in-feeds (i.e. container feed, product feed etc.) or a downstream out-feed blockage. The Un-holding command is generated from an operator when the fault has been handled, while the Un-suspending command from the MO responsible of the fault.
- Fast stop: reached through the Stop command. MO actuators are stopped as quickly as possible but remain powered. Actuators may lose synchronism and the resetting operations should be performed again, once the fault is no more detected. If no kinematic information (e.g. axis position etc.) was lost, no resetting operation is actually performed and the MO will move to Idle state in one PLC cycle.
- Safety stop: reached through the Abort command. Here, MO actuators are stopped as quickly as possible and power is cut. It requires the clear of the responsible faults (from an operator) and the resetting operations.

5.3 MOs communication through PackTags

PackTags are an uniform set of naming conventions, variables and data types for all the data elements necessary within the PackML approach. PackTags are broken out into three groups:

Command Tags: for commanding the MO. They also include the recipe parameters for the MO operations.
Status Tags: for describing the MO internal state. State is intended in its large meaning (i.e. the current MO situation) and not just the state of the base state machine; for example type of processed items etc.
Admin Tags: for quantifying the performed operations (e.g. number of processed items etc.) and the alarm information.

Many PLC vendors have implemented PackTags, or part of these. However, they may also be implemented *ex novo*. In our approach, we used a subset of them which type is described in Sec. 4.2. Moreover, we implemented the communication among MOs though the Rockwell alias mechanism. Objects have a modular behaviour through the utilization of local tags. Their external interface is implemented setting the local variables alias of global ones. For example, locally EM1 has the tag My_AlarmList which is alias of the global EM1_AlarmList. This latter will be utilized by the UN for read/write operations.

5.4 Type and scheduling of routines

For every MO, the structure defined in Figure 2 is proposed and all the routines are written in the IEC 61131-3 structured text. However, also other languages of the standard might be adopted.

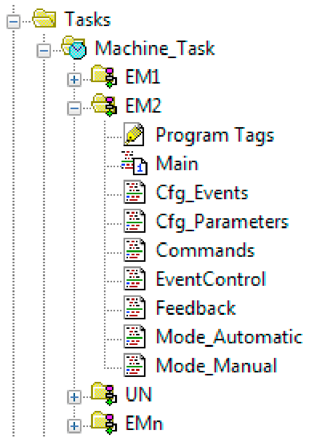

```
Tasks
  Machine_Task
    EM1
    EM2
      Program Tags
      Main
      Cfg_Events
      Cfg_Parameters
      Commands
      EventControl
      Feedback
      Mode_Automatic
      Mode_Manual
    UN
    EMn
```

Fig. 2. The proposed software architecture.

Every PLC-cycle, the Main routine is first executed. This is the main sequencer which calls the illustrated routines in the following order:

- Operation 1: active just during the first scan. The parameters are set: definition of the configuration parameters for the AOI_PackML, of the MO parameters and of the configuration parameters for the AOI_CM_Events. These operations are performed respectively through the call of the Cfg_Parameters and Cfg_Events routines.
- Operation 2: trigger conditions are controlled and events and lists are generated (EventControl routine).
- Operation 3: inputs are read from the hardware and from the other hierarchy levels (Command routine).
- Operation 4: AOI_PackML calculates the actual mode and state.
- Operation 5: actions of the active mode and state are performed (Mode_k-th routine).
- Operation 6: outputs are written in the hardware memory location and in the MO tags for the communication with other hierarchy levels (Feedback routine).

Next, the most significant routines are illustrated. An extract from the EventControl routine of an EM is shown in Figure 3. It defines an alarm generated due to a certain trigger condition. The Cgf_Event[0] tag assigns a certain ID, value, string and category to the event, which is automatically sent to My_AlarmList. The list is then filled through the AOI_EventManager. The UN EventControl routine contains events generated at machine level (e.g. Door Open alarm) and merges the event lists of the EMs through the AOI_EventSummation.

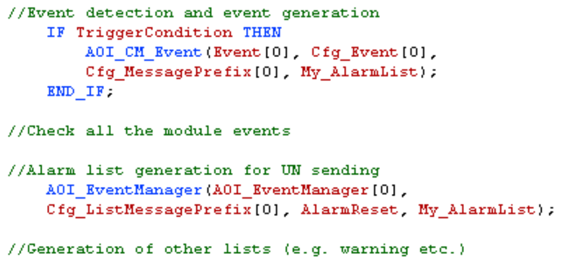

```
//Event detection and event generation
    IF TriggerCondition THEN
        AOI_CM_Event(Event[0], Cfg_Event[0],
        Cfg_MessagePrefix[0], My_AlarmList);
    END_IF;

//Check all the module events

//Alarm list generation for UN sending
    AOI_EventManager(AOI_EventManager[0],
    Cfg_ListMessagePrefix[0], AlarmReset, My_AlarmList);

//Generation of other lists (e.g. warning etc.)
```

Fig. 3. Extract from the EM EventControl routine.

Figure 4 illustrates part of the UN Command routine. As one can notice, the sending of commands goes from the higher hierarchy levels to the lowers, whereas the state complete and alarm management follows the opposite direction. The first block shows the management of an alarm of category 0 through the UN sending of the command abort to itself and to the EMs. In the second block, it is shown how the operator can interact with the machine by writing the global tag UN_PackML, alias of the local tag My_PackML of the UN. Eventually, the last block illustrates how the state complete propagates from the EMs to the UN.

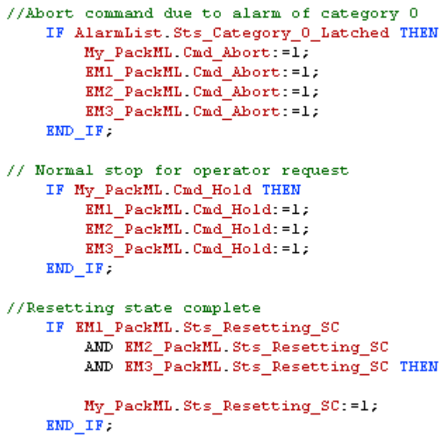
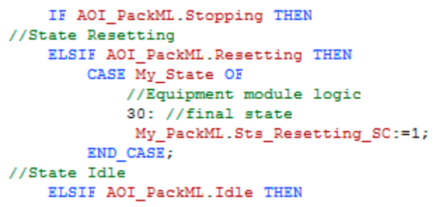

```
//Abort command due to alarm of category 0
    IF AlarmList.Sts_Category_0_Latched THEN
        My_PackML.Cmd_Abort:=1;
        EM1_PackML.Cmd_Abort:=1;
        EM2_PackML.Cmd_Abort:=1;
        EM3_PackML.Cmd_Abort:=1;
    END_IF;

// Normal stop for operator request
    IF My_PackML.Cmd_Hold THEN
        EM1_PackML.Cmd_Hold:=1;
        EM2_PackML.Cmd_Hold:=1;
        EM3_PackML.Cmd_Hold:=1;
    END_IF;

//Resetting state complete
    IF EM1_PackML.Sts_Resetting_SC
        AND EM2_PackML.Sts_Resetting_SC
        AND EM3_PackML.Sts_Resetting_SC THEN

        My_PackML.Sts_Resetting_SC:=1;
    END_IF;
```

Fig. 4. Extract from the UN Command routine.

Figure 5 shows how the AOI_PackML outputs are utilized for selecting the proper Mode_k-th routine and for executing the actions of the active PackML state. The example illustrates how hierarchical state machines can be built, refining the PackML states with a state machine. Moreover, the state complete is set once all the actions of a certain state have been performed. As one can notice, the software designer must fill the control logic of the PackML states without considering the PackML modes and states transitions which are automatically calculated by the AOI_PackML.



```
    IF AOI_PackML.Stopping THEN
//State Resetting
    ELSIF AOI_PackML.Resetting THEN
        CASE My_State OF
            //Equipment module logic
            30: //final state
             My_PackML.Sts_Resetting_SC:=1;
        END_CASE;
//State Idle
    ELSIF AOI_PackML.Idle THEN
```

Fig. 5. Example of the Mode_k-th routine.

6. RESULTS AND FUTURE WORKS

This paper has proposed an interpretation of the tools provided from the PackML approach and defined a Rockwell-based design pattern for writing modular PLC code based on the P&G implementation.

The proposed approach will bring the following benefits:

- Re-usability enhancement through the creation of modular and parametric MOs.
- Common "look and feel" for all the objects in terms of: modes and state machines, interfaces and communication mechanisms, and alarm management. The standardization of these aspects facilitates the software comprehension and let the designers concentrate just in the control logics.
- Companies are forced to rationalize the machine before writing software. This is something usually neglected from SMEs.

Future works will consist in:

- include in the approach redundant MOs which determine asynchronous states. Consider the situation in which two redundant EMs are inserted. One may be in a fault situation (e.g. Stopped state), while its function is being performed from the redundant EM. In this way, an EM will be in Stopped state, and the other EM and the UN in the Execution state. In our approach, we only considered synchronous states.
- application to industrial case studies in order to refine and validate the approach.

We hope that this work will enhance the PackML discussion for transforming it from an approach to a standard. We do not expect our interpretation to be the best and completely unambiguous option, however we wish that OEMs and End-Users collaborate with OMAC for reaching a common interpretation.

REFERENCES

Bonfatti, F., Gadda, G., and Monari, P. (2001). PLC software modularity and co-operative development. In *Advanced Intelligent Mechatronics*.

Bonfé, M., Fantuzzi, C., and Secchi, C. (2006). Behavioural inheritance in object-oriented models for mechatronic systems. *Internation Journal of Manufacturing Research*.

Bonfé, M., Fantuzzi, C., and Secchi, C. (2013). Design patterns for model-based automation software design and implementation. *Control Engineering Practice*.

Chiron, F. and Kouiss, K. (2007). Design of IEC 61131-3 function blocks using SysML. In *Mediterranean Conference*.

Drath, R., Luder, A., Peschke, J., and Hundt, L. (2008). AutomationML - the glue for seamless automation engineering. In *Emerging Technologies and Factory Automation (ETFA)*. Hamburg, Germany.

Fantuzzi, C., Bonfé, M., Fanfoni, F., and Secchi, C. (2011). A Design Pattern for translating UML software models into IEC 61131-3 Programming Languages. In *18th World Congress of International Federation of Automation Control (IFAC)*.

International Electrical Commission (2013). IEC 61131 Programmable Controllers Part 3 : Programming Languages. Technical report.

ISA (2008). Machine and unit states: An implementation example of ISA-88. Technical report. `http://www.isa.org`.

ISA (2010). Batch control part 1: Models and terminology. Technical report. ANSI/ISA88.00.012010.

Katzke, U. and Vogel-Heuser, B. (2007). Combining UML with IEC 61131-3 languages to preserve the usability of graphical notations in the software development of complex automation systems. In *Analysis, Design, and Evaluation of Human-Machine Systems*.

Loughlin, S. (2003). A holistic approach to overall equipment effectiveness (oee). *Computing Control Engineering Journal*, 14(6), 37–42.

Maffezzoni, C., Ferrarini, L., and Carpanzano, E. (1999). Object-oriented models for advanced automation engineering. *Control Engineering Practice*.

OMAC (2006). Guidelines for packaging machinery automation V3.1. Technical report. `http://www.omac.org`.

OMG (2003). MDA Guide Version 1.0.1. Technical report. `http://www.omg.org/mda/`.

PLCopen (2008). XML Schemes and documentation Version 2.01 released . Technical report. `http://www.plcopen.org`.

Ramos-Hernandez, D., Fleming, P., and Bass, J. (2005). A novel object-oriented environment for distributed process control systems. *Control Engineering Practice*.

Sacha, K. (2008). Verification and Implementation of Dependable Controllers. In *Dependability of Computer Systems*.

Stetter, R. (2011). Software im Maschinenbau lästiges Anhängsel oder Chance zur Marktführerschaft? Technical report, VDMA.

Thramboulidis, K. and Frey, G. (2011). Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation. *Journal of Software Engineering and Applications*.

Vyatkin, V. (2013). Software Engineering in Factory and Energy Automation: State of the Art Review. *IEEE Transactions on Industrial Informatics*.