Preprints of the 18th IFAC World Congress
Milano (Italy) August 28 - September 2, 2011

Specification and implementation of a Hierarchical and Modular meta-model for manufacturing system Control

Luca Ferrarini*, Alessio Dedè, Andrea Gianazza*****

**Politecnico di Milano - Dpto. di Elettronica e Informazione, Milan, ITALY (Tel: +39-02-23993672; e-mail: ferrarin@elet.polimi.it).*
***Politecnico di Milano - Dpto. di Elettronica e Informazione, Milan, ITALY (Tel: +39-02-23994023; e-mail: dede@elet.polimi.it).*
****Politecnico di Milano - Dpto. di Elettronica e Informazione, Milan, ITALY (e-mail: andrea.gianazza@mail.polimi.it).*

Abstract: The paper presents a research work proposing the definition of a new approach to design and test the control logics for manufacturing systems. Starting from the Object-Oriented paradigm, a modular and hierarchical meta-model for the control specification and design is proposed. The core of the new approach is the *control module*, a basic standard reusable functional entity which permits to design and implement a portion of larger control logics focusing only on the behaviour of single component. The interface standardization and the hidden information concepts allow facilitating the overall control architecture design, the testing of the developed logics, a high level of reuse of the pre-designed *control modules* and finally an automatic control code generation. The paper describes an application of the proposed methodology for a real application, the machine SPI, providing an overview on the GUI editor and on the control code generator for a commercial soft PLC.

Keywords: Control system design, Manufacturing systems, Hierarchical Control, Logic Design, Control oriented models

1. INTRODUCTION

One of the most important goals addressed in the last decade by modern production facilities is the reduction of time and costs in the production itself. This is particularly true for the manufacturing sector where the enlarged world market has introduced a world-wide high level of concurrency. In this scenario a lot of research projects have been carried out. Most of them are focused on the increase of the flexibility of the production plants. In this case the aim is the reduction of time and costs with special reference to frequent setups of the plants caused by the changes of the production mix or by possible faults or malfunctions. The EU projects PABADIS and PABADIS PROMISE (Ferrarini et. al (2006)), for example, have defined a new control architecture based on the agent concept. The proposed architecture is realized following the agent paradigm and it is a complex and flexible supervisory control structure which allows a fast re-configuration of the production systems. Similar approaches are obtained in SOCRADES (2009), and DISC (2010).

The flexibility and quality above mentioned bring the plant to a high level of complexity. For this reason the time and costs reduction question becomes a problem for plants producers. In this direction the necessity to improve the modelization and, more in general, the design of production plants is becoming an investigated topic (Endsley (2004) and Bunse et. al (2007)). In order to control the complexity and the dimension of the systems, it is mandatory to start from a suitable model, definitely including efficient decomposition and modularization criteria. This idea is also supported by the increase of the processing power distribution and the diffused Object-Oriented paradigms which are driving the evolution of the control design phase and, consequently, of the entire plant. In this way new approaches has been proposed defining the concept of component as single element able to interface itself with others components to create a net which composes the overall controlled system. Each component can model a control part, a mechanical/physical one or both. A distributed control architecture has been defined in the TORERO project (Schwab et. al (2005)) and in (Sunder et. al (2006)) where, exploiting a set of interconnected devices/modules, the control logics have been divided in a net of control units able to exchange information in order to achieve the desired behaviour of the system. The MEDEIA project (MEDEIA Consortium (2008)) has improved this concept defining the idea of Automation Component as element useful to model the control, the mechanical parts, the hardware and the diagnostics of a generic automation system. The overall plant is modelled as a set of connected Automation Components and the implementations of the control, diagnostics or simulation code are obtained starting from such a model-based approach through a set of code generators specific for each considered hardware platform.

The paper presents an innovative model-based approach for the design of the control logics and diagnostics for manufacturing systems starting from the results discussed in (Castelnuovo and Ferrarini (2005)). The proposed model can be considered as a particular specification of the MEDEIA Automation Component. As a matter of fact, the Control

Module (CM), described in next sections, is similar to an Automation Component but it is specific for manufacturing plants and in particular it is used only to model the control logics and the diagnostics of each sub-system.

The paper is organized in six sections. After this short introduction on the discussed work, a general overview of the proposed model is provided in section 2. Instead, section 3 describes more in detail the control design and the application of the modelization approach. Section 4 is focused on the implementation of the model editor and code generator for the softPLC Orchestra (Sintesi SCpA (2009)), which are used in the work presented in section 5 to model and the control the SPI Machine. Section 6 discusses the obtained results and the next steps and goals that will be achieved in future.

2. CONTROL-MODULE

2.1 Overview for the Proposed Control Architecture

The proposed control architecture is based on a tree structure, in order to grant the characteristics of modularity and hierarchical organization to the whole system. Each node can communicate with its children and with the father node while the communications between nodes of the same level are forbidden as those between nodes belonging to non adjacent levels. There is no limit to the number of levels of the tree structure as there is no limit to the number of children that a node can have. The generic node of the tree structure is implemented by a control module that is equipped with a set of functions that make it able to manage its internal state and the communication with other control modules. In this way, the overall architecture is composed by a set of standard modules with standard interfaces which permit the multi-level aggregation and enhance reusability. An example about the implementation of the control architecture with three levels is presented in (Ferrarini and Dedè (2010)) and shown in Fig.1. Each node of the tree structure is the aggregation of its children and the provided functions contain calls to the sub-level functions. To realize this structure a preliminary multi-level functional decomposition is needed in order to identify the main components of the system. The next steps are a dual formalization of the control functions provided and then the organization of the components in a modular and hierarchical control architecture.

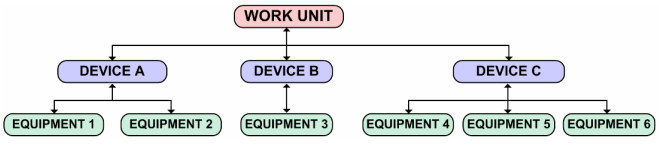

Fig. 1: Example of the modular and hierarchical architecture for the control model

2.2 The Control Module

The Control Module (CM) meta-model provides a standard structure with a standard communication interface. This allows that any possible change to the implementation of the internal structure of a single CM does not affect other CMs and therefore the requirements of modularity and independence are maintained. As shown in Fig. 2, the internal body of the CM is composed of three different entities: the State&Operation Manager (SOM), the Operation Body (OB) and the Diagnostic Manager (DM). These entities allow the CM to manage the external communication interface, using a parametric set of events to exchange information with other modules, and to control and update its internal state, which is characterized by the available operations and "in-execution" ones. The behaviour of the control module can be summarized as follows: the CM awaits requests from the upper level; then the received requests are accepted or denied according to its internal state. If accepted, the request activates an operation. During the execution of an operation, the CM can receive other requests from the upper level and can make requests to the lower level. The CM can also manage the parallel execution of operations. When the CM terminates the execution of an operation, it communicates to the upper level if the operation has been successfully executed or not. Finally the CM is able to handle anomalies and errors and to provide diagnostic functions.

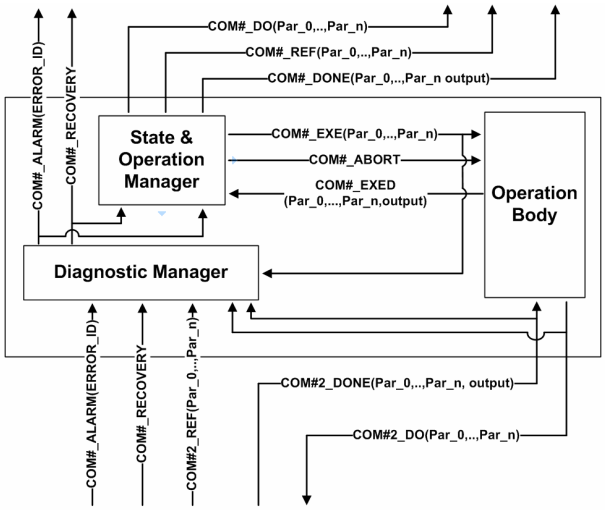

Fig. 2: Control Module Architecture

3. CONTROL DESIGN

This section presents the State&Operation Manager and the Operation Body, which manage the control functions of the module in most cases. The third module composing the body of the CM, the Diagnostic Manager, primarily provides diagnostic functions (ability to detect and identify faults and communicate the results to upper levels). The inner behaviour of such a sub-module is not discussed here since the paper mainly focuses on the control design.

3.1 State&Operation Manager

This module manages the state of the CM keeping up-to-date the available functions and managing the function calls of the upper level. It is modelled as a trigger\response FSM. Each state represents a particular state of the control module and the transitions are marked with a set of alarm, recovery and execution events that are used to model the occurrences of

particular requests or signals which change the state of the control module. When the SOM receives a request from the upper level to execute a particular function, it, according with the CM actual state, asks to the Operation Body to execute it or, if the function is not available, it answers with an execution denial event to the caller.

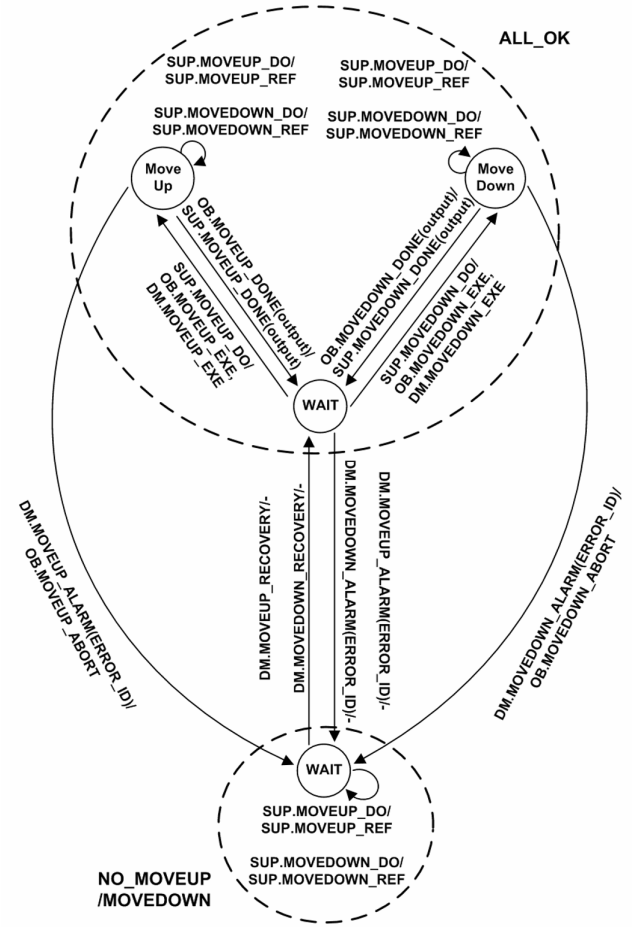

Fig. 3: Example of the SOM automaton

The SOM states are organized in sub-sets, each of which represents the macro-state of the device which is characterized by the available functions, so that the transitions between two different macro-states are marked with recovery or alarm events. The availability of a CM function mainly depends on the availability of the functions of the lower level CMs. So, in the construction of the SOM, mutual-exclusion and connection rules between the CM functions must be given. If two functions are in mutual-exclusion and one is in execution, the other one is not available. The connection rules define the connected availability so that if two functions are connected and one is unavailable the other one is also not available. Fig. 3 shows an example for a simple SOM automaton with two functions, called MoveUp and MoveDown, that are connected and in mutual exclusion. So in the automaton there are two sets of states: in the first the two operations are available, but only one can be executed at a time since they are in mutual exclusion, instead in the second none of the operations can be executed, since they are connected. The name of each state of the automaton is the composition of two parts: a prefix indicating the sub-set to which it belongs, for example ALL_OK, and a suffix indicating a unique name inside the sub-set, for example WAIT. To make the picture (Fig. 3) clearer only the suffix of each state has been listed inside each sub-set. Then the two WAIT states in figure are different: the first, whose full name is ALL_OK_WAIT, indicates a waiting state where all operations are available, while the second, whose full name is NO_MOVEUP/MOVEDOWN_WAIT, indicates a waiting state where no operation is available. In this case the automaton is quite simple, having only four states subdivided in two macro-states, but in real cases the number of the states and in general the complexity of the overall control do not allow to synthesize the SOM manually. In order to avoid the direct design of the SOM automaton, an algorithm composed by a set of rules has been defined and implemented to generate the SOM. This generator needs information about the functions of the device (names and parameters) and about the mutual exclusions and connections rules. To simplify the design phase, a standard form to specify this information has been defined.

3.2 Operation Body

The Operation Body (OB) block models a particular function, called operation, provided by the CM, so the CM will have as many OBs as functions. Each operation is characterized by a univocal name, a parameter list (may be empty), an OkCondition, a list of commands to be executed in case of abort and a sequence of actions that must be performed in order to achieve the correct expected result. In the lower levels, an action can be thought as a set of commands followed by a control on a set of signals provided by the plant sensors. Increasing the abstraction level of the model, the commands can be modeled as writing variables and the sensors signals as reading variables. Fig. 4 shows a class diagram which presents the elements composing the OB. Also in this case, a labelled automaton is used to model the working of the OB when the particular operation is called. The OB starts working when it receives a request from the SOM. In this case, before starting the execution, the OB verifies the OkCondition which represents the state of the plant after the operation execution, and if this condition is verified, the OB notifies to the SOM that the operation has been successfully executed, otherwise it starts the operation execution.

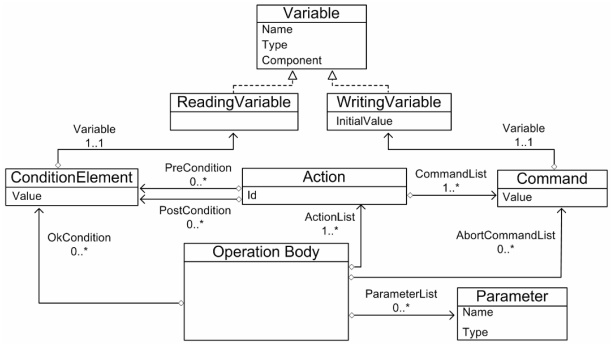

Fig. 4: Class Diagram describing the structure of the OB

In the latter case, the OB starts carrying out a sequence of actions, each of which is characterized by the verification of a precondition, an execution of a set of commands and finally a verification of a postcondition. The accomplishment of an operation execution is notified to the SOM, which reports to the caller if the operation has been terminated correctly or not. As just described before for the SOM automaton, also the OB one is automatically generated to facilitate model (re)use. In this case, the control designer must specify the information about the operation and an automaton generator creates automatically the OB automaton.

4. IMPLEMENTATION AND CODE GENERATION

This section presents the implementation adopted for the meta-model previously described and shows the steps to generate the control code for PLC. Moreover, the tools that the designer can use to specify the information about the Control Module and the benefits provided by this approach, compared to the traditional design methods used to realize control code, will be shown.

4.1 Description of the Control Module using XML

As mentioned earlier, a standard form to specify the information regarding CM features has been defined to simplify the design phase. More specifically, a XML meta-model, i.e. a XSD (XML Schema Definition), has been defined to describe the features of the control functions of the device, as shown in Fig. 5.

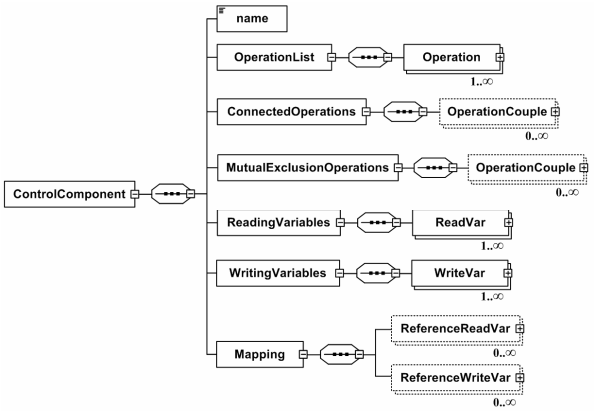

Fig. 5: XML Schema Definition (XSD) for the CM

To realize the description of the single CM, the designer has to specify a univocal name for the ControlComponent, declare the list of reading and writing variables used in the control functions and define the operation list. In particular, each operation is characterized by a univocal name, a parameter list, where each parameter is identified by a name and a type, an OkCondition, i.e. a condition describing the state of the plant after the operation execution, a list of commands to be executed in case of abort and a sequence of actions that must be performed. Each action is characterized by an Id, a precondition, which must be verified to execute the action, a list of commands that must be executed and finally a postcondition, which identifies the action end. The last element shown in the XML Schema is the Mapping list: for the devices in the lower levels of the tree structure, this list contains a link between each variable used by the CM and a field variable, therefore a reading variable will be linked to a sensor output and a writing variable to an actuator input (e.g. an electric motor or a solenoid valve). Instead for the devices in the upper levels, the variables used by the CM are linked to the events that enable function calls between devices of different levels. The XSD can be subdivided in two sections: the first one contains the general description of the behaviour of a generic device, which is not bound to a particular implementation linked to a specific plant, while the second one, identified by the mapping list, defines the physical implementation of the device. This particular structure of the XSD allows an easy reuse of XML descriptions of devices previously developed through simple copying\pasting operations or through the definition of libraries, on condition that the mapping list has to be edited. As an example, when the description of the general behaviour of an electric door has been defined with an XML, this XML description can be copied and pasted in all the projects that contain electric doors, leaving the designer to implement the mapping list, so he only has to specify the links between field variables and the ones used by the electric door CM. To further simplify the editing of XML descriptions, a graphic editor has been implemented using Eclipse Modeling Framework (EMF) (Steinberg et. al (2008)), an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model. This editor directs the designer during the implementation of XML description, indicating which tags can be added in specific points of the document, reducing the chance of errors and providing functions of validation on the document structure. Fig. 6 shows how XML descriptions realized with the EMF editor can be reused in more projects.

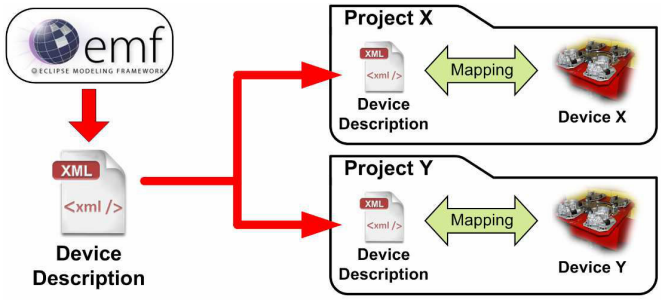

Fig 6: Example of reusability of the XML descriptions

4.2 Automatic Code Generation

The previous paragraph shows how to describe in a standard and formal way the general information regarding the Control Module. The next step is to translate this information in the automatons presented in section III. To do this, a set of translation rules has been defined to obtain the SOM and OB automatons starting from the XML description. In particular, for each device described in the XML, an algorithm allows to obtain a single SOM automaton and an OB automaton for each available operation. So a Java implementation of these rules has been realized. In this way the designer can automatically obtain the whole CM implementation starting from the XML descriptions. This CM implementation models

are conceived for a particular platform, in order to execute and connect control HW/SW with the physical system. In this work we focus on the widely used PLC architectures and on the standard IEC 61131-3, a normative which describes the programmable languages used to implement the software logics. Only Ladder Diagram (LD) and Sequential Flow Chart (SFC) have been used to implement the CM, considering as target platform the XML defined by the PLCOpen organization (PLCOpen (2009)).

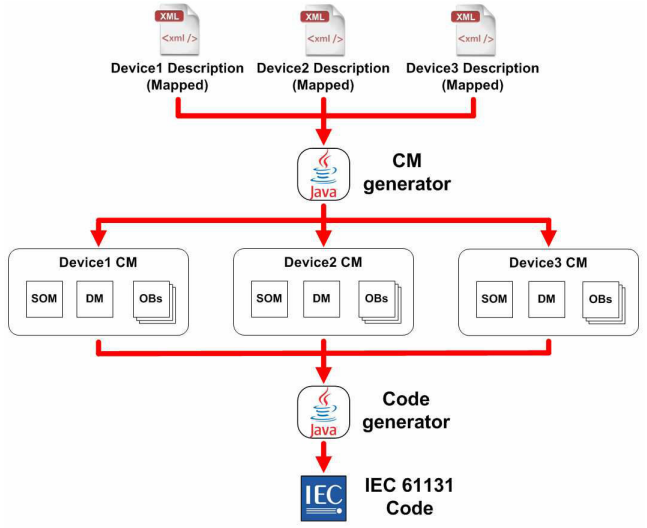

Fig. 7: control code generation starting from XML files

An automatic code generator algorithm has been defined and implemented. Starting from the CM models, it generates a set of Program Organization Units (POUs) which implement the logics of the SOM and of the OBs. In particular, the generator translates the SOM automaton states and transitions in a sequence of rungs inside a LD program, while the OB automaton is translated in a SFC. The choice to use LD for the implementation of the SOM automaton is due to the fact that this language is particularly convenient to handle the parallel execution of different operations. Instead, the presence of preconditions, commands and postconditions shows a lot of affinities with the SFC structure, so this is why this language has been chosen for the implementation of the OB automaton. Since in the IEC 61131 is not defined the concept of event, the communication between different CMs is realized through a set of shared variables. Fig. 7 shows an example of control code generation. Starting from the XML descriptions realized with the EMF editor, a CM generator implements the CM automatons while another generator, using the previously implemented automatons, generates IEC 61131-3 control code, which can be performed on a PLC. A graphic interface has been implemented to help the designer to choose the XML descriptions from which generate the control code. It is clear that it is possible to define and implement different code generators for different target platforms: the IEC 61131 generation proposed in this paper is just an example of code generation. In this way, the proposed meta-model facilitates the reuse of the designed models and control logics which can be implemented for different control systems. In conclusion, it is important to underline that the designer has only to make a high level description of the plant, through the XMLs, and then a software application will realize for him the control code in the language he prefers.

5. CASE OF STUDY: THE MACHINE SPI

The aim of the presented framework is to design control logics for manufacturing systems and in particular for machining centers, which exhibit a medium-high level of complexity with respect to classic industrial automation applications. In order to test the presented approach, it has been applied to design control logics for a particular manufacturing scenario. The considered test case is an automatic machining center, called SPI Machine, with five axis used to produce carters, which are mechanical parts used to assemble the flywheel and the clutch plate of motorcycle engines. The machine is composed of an automatic system to load/unload pieces, a local tool-store with thirty positions, a cartesian spindle (three axes) and a tilting table (one axis) with two plates and fixtures used to lock the pieces during the machining phase (one axis). This test case has been controlled using a simulated version of the SPI Machine. Fig. 8 shows a comparison between the real machining center and the simulated one.

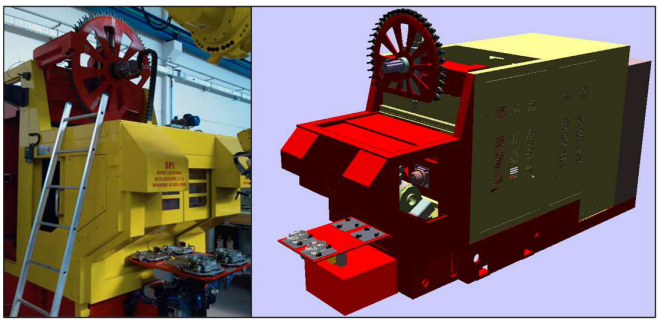

Fig. 8: Real SPI Machine (left) and the simulated one (right)

5.1 Design phase

In order to design a modular control system using the proposed technique, a physical decomposition has been applied to the SPI Machine with the aim to obtain a three level decomposition where each level contains a set of control modules (associated to physical machine portions). Fig. 9 presents a part of the control architecture designed for the SPI Machine. Each block is a Control Module implementing a set of functions provided to the upper level blocks. A XML description for each block of the tree structure has been realized with the EMF editor. This test case presents an example of reusability of the XML already implemented: as a matter of fact one electric door XML description has been realized and it has been reused for the four electric doors of the SPI Machine, editing only the mapping section of each door.

5.2 Code generation and simulation

The XML models, created with the EMF editor, have been used as input of a Java application which generates an IEC 61131-3 control code project. In particular, the control logics have been tested using Orchestra Control Engine, a soft-PLC

based on Linux RTAI (Real-Time Application Interface) which uses the PLCopen XML standard as storage system. To test the generated Orchestra project the soft PLC has been applied to a graphic and kinematics simulator of the considered machine in a closed feedback loop. This particular simulator configuration, called Hardware-In-the-Loop simulation (HILS), has permitted to test the code generation with a real PLC without the availability of the real machine. This testing architecture did not comprise the DM and the diagnostic system in general. This part of the CM has been generated separately using an external Java application able to create a set of C applications for the specific PLC Orchestra. Such a generated code implements the diagnostic models used to isolate possible faults occurred in the simulation environment (Ferrarini et. al (2010)).

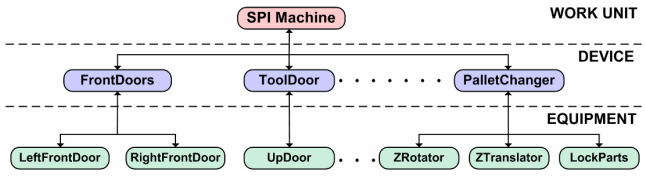

Fig. 9: Machine modular and hierarchical control architecture

6. CONCLUSIONS

The paper presents a new modular and hierarchical approach to design the control logics for manufacturing plants. The new concept of Control Module architecture has been applied to a real automatic system to prove the applicability of the proposed models. The Model-Based design has permitted to reuse a subset of the defined CMs reducing the required time to develop and test the logics for the considered machining centre. This is true in most of the manufacturing cases where single components are usually used in more scenarios. In this way, the reuse of the CMs allows the creation of components libraries. The type/instances concept, which are proper of the Object-Oriented programming languages, has been applied in the control design methodology in order to reduce the costs, increasing at the same time the safety and the accuracy of the final controlled systems. In particular, the application of the library with an automatic code generator permits to obtain a good quality of the generated logics.

A set of code generators and the mapping part of the CM guarantees the portability of the developed logics allowing the reuse of the modelled behaviours in more than one hardware controller. The idea is to use the CM model as a type which can be instantiated for each particular case.

As said, the whole process, from a high-level human-prone specification down to testing with HILS approaches and code generation has been investigated, tested on real machining centres, and automatized with automatic model transformation techniques and tools. The results presented in this paper must be considered as a starting point for future developments for real use in practice, with the final aim to obtain a high level modelization approach useful to develop control logics for complex manufacturing systems. In particular, the diagnostics should be better integrated and a set of graphic editors must be implemented in order to facilitate the creation of new CMs and the connections between them. With the goal to disseminate the usage of this model-based approach among practitioners, a set of code generators will be implemented, expanding its applicability to more hardware platforms.

REFERENCES

Bunse, C., Gross, H., and Peper, C. (2007). Applying a Model-Based Approach for Embedded System Development. *33rd EUROMICRO Conference on Software Engineering and Advanced Applications*. Lubeck, Germany, August 28-31.

Castelnuovo, A., and Ferrarini, L. (2005). A Modular Formal Model for Pallet Transportation System in Machining Centres Automation. *Decision and Control, European Control Conference*.

DISC Consortium (2010). Project website: http://www.disc-project.eu/index.html

Endsley, E. W. (2004). Modular Finite State Machines for Logic Control: Theory, Verification and Applications to Reconfigurable Manufacturing Systems. Ph.D. thesis, University of Michigan, Ann Arbor, MI

Ferrarini, L., Allevi, M., and Dedè, A. (2010) Design and implementation of an automatic on-line diagnosis with TiDiaM and TiDE. *ETFA10 IEEE International Conference on Emerging Technology and Factory Automation*. Bilbao, Spain, September 13-16.

Ferrarini, L., Dedè, A. (2010). A Modular and Hierarchical Meta-Model for the Control Design of Manufacturing Systems. *6th annual IEEE Conference on Automation Science and Engineering, CASE 2010*. Toronto (Canada), August 21-24

Ferrarini, L., Veber, C., Lüder, A., Peschke, J., Kalogeras, A., Gialelis, J., Rode, J., Wünsch, W., and Chapurlat, V. (2006). Control Architecture for Reconfigurable Manufacturing Systems: the PABADIS'PROMISE approach. *11th IEEE International Conference on Emerging Technologies and Factory Automation*. Prague, Czech Republ, September 20-22.

MEDEIA Consortium (2008). Project website: www.medeia.eu.

PLCOpen (2009). Organization website: http://www.plcopen.org.

Schwab, C., Tangermann, M., and Ferrarini, L. (2005). Web based Methodology for Engineering and Maintenance of Distributed Control Systems: The TORERO Approach. *3rd IEEE International Conference on Industrial Informatics*. Perth, Australia, August 10-12.

Sintesi SCpA (2009). Orchestra Control Engine, website: http://www.orchestracontrol.com.

SOCRADES Consortium (2009), Project website: http://www.socrades.eu/Home/default.html.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2008). Eclipse Modeling Framework, 2nd Edition. Erich Gamma, Lee Nackman and John Wiegand Eds.

Sunder, C., Zoitl, A., Rainbauer, M., Favre-Bulle, B. (2006). Hierarchical Control Modelling Architecture for Modular Distributed Automation Systems. *2006 IEEE International Conference on Industrial Informatics*. Singapore, August 16-18.