Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

A Modular Formal Model for Pallet Transportation System in Machining Centres Automation

Adamo Castelnuovo and Luca Ferrarini, *Senior Member, IEEE*

Abstract— **The paper deals with modelling and control problem of machining centres. In particular, here only the pallet transportation system is analysed. The aim is to provide formal design models for the control engineer, so as to help the reconfigurability and diagnosis of specialised, software-intensive automation systems, and to exploit the potentials of agent-based control software development. The formalism of Modular Finite State Machines (MFSM's), well-known from the scientific literature in the manufacturing control field, is here adopted to represent the formal behaviour of control modules of the pallet transportation system in machining centres.**

I. Introduction

AMONG others, one of the most complex and challenging component which can be found in modern discrete manufacturing systems is represented by machining centres. They are at the basis of a number of interesting control problems, ranging from low-level control, supervision, formal verification, fault detection, diagnostic, just to mention a few. For most of such problems, from a practical perspective there is a strong push towards modularization and autonomy, which implies the distribution of "intelligence" (that is control laws, hardware and software control components) into the controlled plant. In particular, to enhance the overall system performance and to integrate automation level with management level, there is a natural trend to include methods and tools of the Information and Communication Technologies down into the automation systems [8,9].

On the other hand, control components of manufacturing systems have not experienced the same trend to modularization and standardization as the other plant components, most notably the mechanical and the electrical ones. Such a delay has impressive, though sometimes hidden, costs in the design, implementation, testing, installation and maintenance of the overall control system, and reduces the possibilities to gain real flexibility, real reconfiguration and reuse of control solutions [3].

Clearly, this requires new modelling paradigms able to capture the overall system description and help the translation of the more and more strict control specification into control design and implementation. Traditional design models and approaches used by control engineers are based on low-level modelling paradigms (often programming languages), centralised approaches, proprietary tools and solutions.

A viable solution from the methodological point of view seems the one based on two different, conceptual and practical models of the controlled plant [7,8]: the plant model, describing the plant components, data sheets, and I/O points, and structured in a modular hierarchical model, and the control model, obtained starting from the plant model, and following its basic structure, but describing the control commands and rules. Among the advantages, worth mentioning are the separation of functions from code, the semi-automatic generation of the control code, the clear correspondence between design model and implementation model, the soundness of the overall architecture, the availability of formal models.

Some notable proposals are emerging both in the scientific literature [1,2,7] and in the standardization field [10,11], encouraging the adoption of more formal design methods and object-orientation concepts [8,9]. In the manufacturing field, agent-based techniques [2,3,4,9] are even more promising for the design and implementation of the automation code. Agents are software components created as a natural extension of objects characterised by having autonomous control of their own execution, showing both reactive behaviour and proactive behaviour, and cooperating among one another to reach a common global aim.

Although there are successful experimentations of agents in different industrial applications, systematic formal models describing their behaviour are not extensively described in the literature. In this context worth mentioning is the work developed in [3] which addresses Adacor, a holonic control architecture using high-level Petri net models for each type of holon class representing the dynamic behaviour of the manufacturing components. Worth mentioning is even the work [2], where the holonic characteristics of the IEC 61499 [10] model for distributed intelligent control are exploited to describe a general approach for dynamic and intelligent reconfiguration. Unfortunately, the literature in this field often describes the behaviour of an agent either directly

A. Castelnuovo is with the Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza L. Da Vinci 32, 20133 Milano Italy (e-mail: castelnu@elet.polimi.it)

L. Ferrarini is with the Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza L. Da Vinci 32, 20133 Milano Italy (phone: +390223993672; fax: +390223993415; e-mail: ferrarin@elet.polimi.it)

0-7803-9568-9/05/$20.00 ©2005 IEEE

using a high-level programming language like Java. Alternatively, the formal model refers to a very concise description of the system under control (for example only the description of a portion of agent behaviour is provided without system-level description) or *viceversa* the agent system is described without sufficient insight into the single agent formal specification.

The present paper describes the overall control framework adopted for the control of machining centres. Such a framework is based on the modularity of control models and componentization of software implementation through agent-based paradigm. The aim is to maximise the reconfigurability and self-diagnosis capabilities of specialised, software-intensive automation systems. The formalism of Modular Finite State Machines (MFSM's), well-known from the scientific literature in the manufacturing control field, is here adopted to represent the formal behaviour of control modules of the pallet transportation system of machining centres. The control modules designed will then be implemented as agents in a straightforward way. Thus, such models can be interpreted as formal models of agents and agent systems in an agent-based control software implementation.

The paper is organized as follows. Section II introduces the main subsystems considered as illustrating examples (pallet units) in this paper while Sec. III shows the proposed control framework. Section IV exhibits the formal specification for the control modules and section V describes the application of the control model to the chosen examples. The results are summarised in Section VI.

II. The Considered Pallet Units

In the present section, the main topologies of pallet transportation unit investigated are described.

Shuttle with pallet buffers

The shuttle typically has a linear movement, and moves along suitable tracks. Buffers can be on one or both sides of the track, and can have a single or multiple floors. The shuttle is endowed with suitable clamping devices to unload a pallet to the buffer or viceversa to retrieve and load a pallet from the buffer.

Single or Double fork with circular buffer

This unit is composed of a fork device with a single or double clamp (each clamp can host one pallet at most). The buffer is circular (usually a semi-circumference), with a single o multiple floors. The fork device can rotate to the desired buffer position, lift/lower to pick up/release a pallet, and move forward and backwards to approach or leave a pallet position or the working position. In Fig. 2, an example of circular buffer with a single floor of five position and a pallet unit with double fork is shown.

Rotating Multi-pallet station (carousel)

The rotating multi-pallet station is in principle similar to the circular buffer, but in this case the buffer itself can rotate, lift and lower: a suitable clamping device will then be used to pick up and release a pallet.

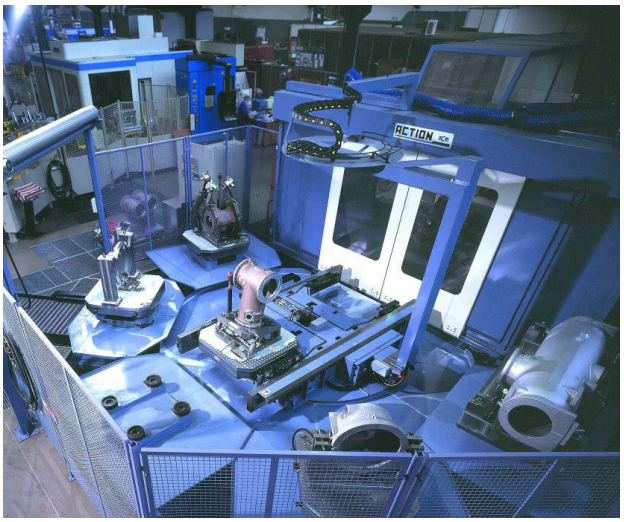

Fig. 2 – Circular buffer pallet unit with double fork

III. Overall Design Methodology

The overall proposed design methodology lays on few concepts described in [7,8], where the object-oriented approach is used not only for modelling and design of the control functions but also to implement the control code.

As it is well-known, among the various advantages of object-oriented approaches there is the "reuse", that can be both reuse of "code" and reuse of "model". As for the latter case, the great improvement than can be obtained from the reusability is that the reuse is not restricted to a single control project, but to a family of related projects, having similar patterns. So, in similar control applications, there are not only similar *objects* (that can be stored in a library for example), but also similar *object relations* and it is interesting to maintain this information through the "family design" life-cycle. This is the idea of object-oriented framework, a set of abstract classes and their relations, as illustrated in [9]. Basically, the reuse comes from the use of the framework and provides also other advantages: reducing development costs and design errors across multiple similar applications, increasing understandability and maintainability of the system, and facilitating system evolution.

In the presented approach these ideas lead to the definition of two frameworks, one for the plant and one for the control system: namely, the *plant* model and the *control* model. The first one is defined through the specialization and aggregation relations; the second one is defined through the specialization and "use" relations. In Fig. 3, the draft structure of the two adopted frameworks is sketched.

Another important improvement towards reusability of models (objects, object relations and code) in the manufacturing field is the adoption of the agent paradigm [4].

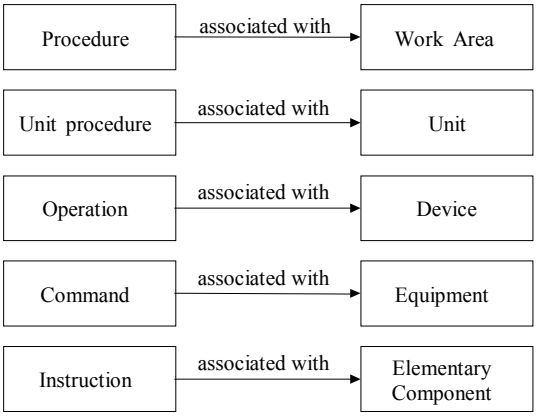

Fig. 3 –Control and plant model relationship

Basically, agents are software components (see also the FIPA standard: Foundation for Intelligent Physical Agents) characterized by autonomy, procreativity and sociality [4].

Unfortunately, only a small effort is paid in the technical literature to the adoption of formal models to describe the desired behaviour of agents. Worth noting is the effort in [3], where a compact PN model is proposed for the agents defined, although no formal modelling of the agent interaction is introduced.

In the present section, each object at any level of the control model of Fig. 3 will be formalized in models called "control modules", which can be implemented directly as agents. The structure of the generic control module (agent) is sketched in Fig. 4 (a single model for any module).

The meaning of the variables in that figure is explained in Table 1, in which O_i is the generic i-th operation within the control module. Basically, the control module can be thought of as realized by the suitable aggregation of different parallel sub-modules: *operation manager*, *state manager*, *operation body* and *alarm generator*. Operations of control modules correspond to methods of objects in the procedural model. Operations are represented by two sub-modules: one, generic, for the management of the request of the operation (operation manager), and the second one for the specification of the operation body. Clearly there is one operation manager and one operation body for each operation that the control module can perform. The state manager, on the contrary, encapsulates the internal state of the agent. Here the agent's state is conceived as the set of constraints to the execution of operations, i.e. the rules according to which an agent can answer to a service request. For example, an agent for shuttle can not perform two release operations one immediately after the other, since a pick-up operation must be executed in between. The meaning of "state" is of paramount importance. Actually all agents, at any hierarchical level, have their own state, which need not to be coherent, since they express just operation execution constraints. Eventually, the alarm generator compares the logical state of an agent (represented by the state manager) with respect to the actual one measured through sensors. For example, the shuttle agent can be in a state that represents the state where the shuttle is fixed in front of a buffer position: should a sensor show a shuttle movement, an alarm would be immediately triggered.

All sub-modules are modelled by means of a formal methodology shown in [1,6], the Modular Finite State Machines, which add to the classical concept of Mealy automaton a set of innovations thus improving its potentialities. Worth mentioning are the ideas of modularity, Trigger/Response FSM and socket for communication among modules [5].

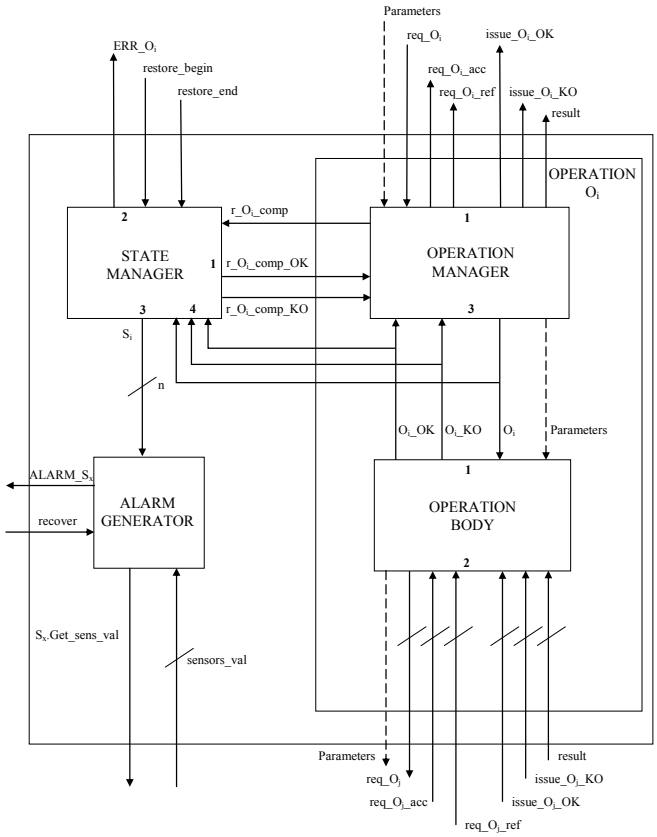

Fig. 4 – Schematic model of a control module (agent)

req_O_i	operation O_i requested by a highier level module
r_O_i _comp	compatibility request by the operation manager to the state manager
r_O_i_comp_OK	the state manager transmits to the operation manager that the request is compatible w.r.t. the state system
r_O_i_comp_KO	the state manager transmits to the operation manager that the request is not compatible w.r.t. the state system
req_O_i_acc	the operation manager forwards the compatibility of the request to the higher

	level
req_O_i_ref	the operation manager forwards the non-compatibility of the request to the higher level
O_i	operation body execution is requested by the operation manager
O_i _OK	operation O_i execution has been successfully completed
O_i _KO	operation O_i execution has not been completed or it has failed
issue_O_i _OK	completion of operation O_i is forwarded from the operation manager to the higher level
issue_O_i _KO	failure of operation O_i is forwarded from the operation manager to the higher level
ERR_O_i	the state manager collects an error when O_i fails and forwards it to the supervisor level
restore_begin	the nominal system working has been recovered, going back to the initial state of the operation in progress
restore_end	the nominal system working has been recovered, going to the final state of the operation in progress
ALARM_S_x	the alarm generator issues an alarm for a non-matching of the sensors' values in state S_x
recover	a supervisor level reports that the alarm has been recovered

Tab. 1 – The events in the MFSM control module

IV. Formal Specification of the Control Module

A. Operation manager

The *operation manager* in Fig. 5 is the same for each operation in the control module in Fig. 3, with the exception of the buffer, which is an information module requiring a simplified design.

B. State manager

The *state manager* embeds constraints on operation execution. This aim is pursued by keeping information about the physical conditions of the associated physical element. Modular Finite State Machines are used to model its behaviour; in general it is necessary to use different MFSM's for the different control modules.

However, it is possible to distinguish among three different patterns in which all the cases can be included (as usual the buffer is an exception): *single waiting state MFSM* (Fig. 6), *two waiting state MFSM* (Fig. 7) and *finite number waiting state MFSM* (Fig. 6). Notice that the representation of a finite number waiting state MFSM is the same as the single waiting state MFSM: the reason is that all the different states can be collapsed in just one symbolic representation of a single state.

In the proposed model it is possible to recognize three different kinds of states whose meanings follow:

- *Waiting state* (empty circle): the module is waiting for an operation request by the operation manager; it can be associated to a plant configuration in which the physical component is motionless.
- *Motion state* (grey-shaded circle): operation is in progress and the module is waiting for its conclusion; it can be associated to a plant configuration in which the physical component is moving.
- *Error state* (dash-shaded circle): state associated to an operation failure which is transmitted by the operation body itself and forwarded to a supervisor level and to the state manager.

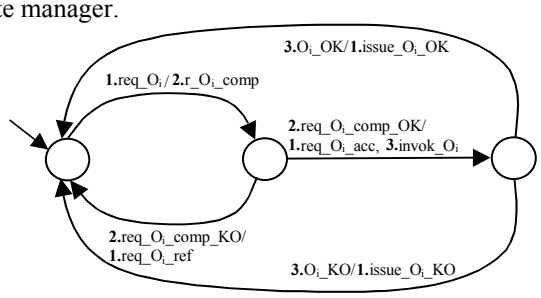

Fig. 5 – The MFSM description of an operation manager

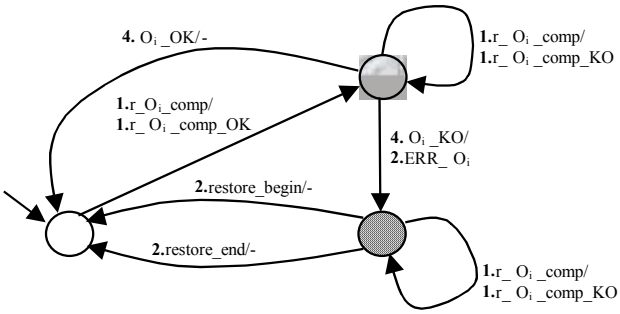

Fig. 6 – State Manager - single waiting state MFSM and finite number waiting state MFSM

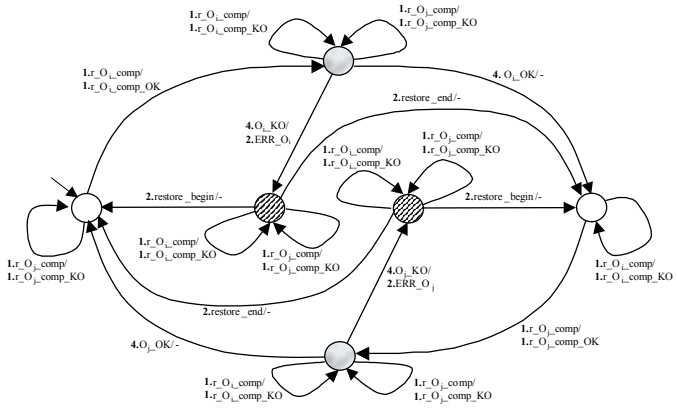

Fig. 7 – State Manager - two waiting state MFSM

C. Alarm generator

The *alarm generator* has the purpose to continually get information from both the state manager and the sensors, to compare them and, in case, to issue an alarm. In particular for each state in the state manager, an expected set of sensors' values is embedded in the alarm generator, and comparison is done between these values and the actual

ones. Whenever they do not match, an alarm event *ALARM_S$_x$* is issued until a *recover* event is triggered by a supervisor level. Different design solutions by MFSM's are currently investigated.

D. Operation body

For the sake of homogeneity, the body of a generic operation has been designed by MFSM's, in order to obtain an overall control methodology characterized by immediate communication among the different modules. Obviously there is a different MFSM for every operation. Worth mentioning is the *operation request pattern*, shown in Fig. 8, which is present in an operation body whenever it requests an operation to a (lower level) module.

In particular, the transition labeled with the trigger event *expired* enables the transmission of an operation failure whenever no result signal appears within a prefixed time ΔT. Such an event can be issued by a timer, that has not been represented in the overall model, whose input events are *Start_timer* and *Stop_timer*, and whose unique output event is *expired*.

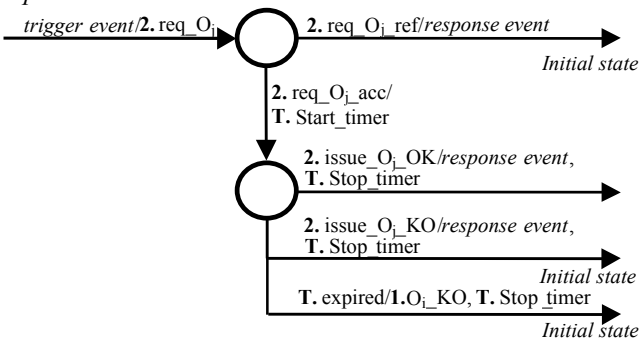

Fig. 8 – Operation request pattern

Alternatively, the operation body could be described in SFC formalism, easier for the control engineer. Automatic translation from SFC to MFSM's is actually under study.

V. THE CONTROL FRAMEWORK OF THE PALLET UNIT

The analysis of the functional characteristics for the previously described pallet unit topologies has given rise to the proposed framework shown in Fig. 9. A set of fundamental elements playing a significant role into the dynamic of pallet transportation has been outlined together with the endowed operations and the mutual relationships. These all have been represented as objects at the level of the control framework (procedural model) proposed in Fig. 3.

At the higher level, for each of the considered cases, the *pallet unit* shows a unique operation named *Move_pallet(X,Y)*, whose significance is "move pallet form position X to position Y". A position is a symbolic representation of set points that must be given to the actuators in order to reach it. It can be a place into the buffer or the load/unload station or the machine charge position.

The pallet unit can be always thought of as composed of a *buffer* and a *pallet transportation system*. The former is basically an information module; it exhibits a set of operations that can be called on by the pallet unit to retrieve information about the current buffer configuration. For example *Get_position_state(X)* can be requested in order to know whether X is free, occupied by a raw-piece pallet or occupied by a worked-piece pallet.

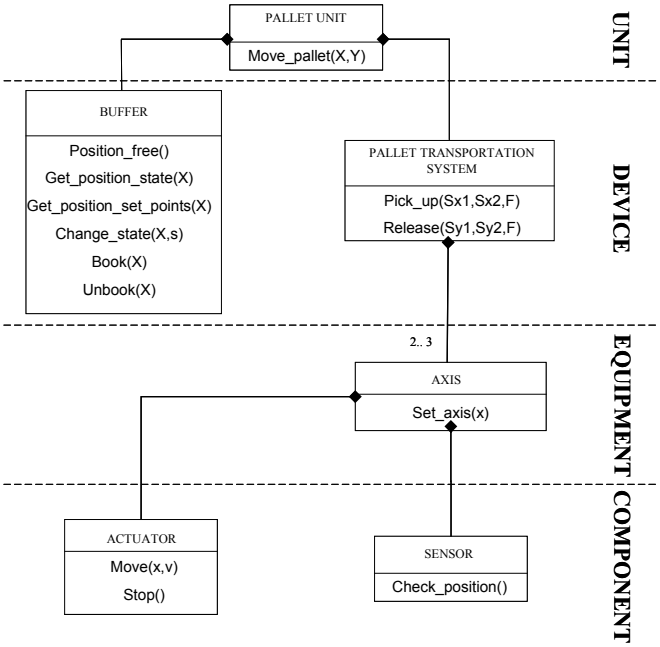

Fig. 9 – The portion of procedural framework for the pallet unit

The *pallet transportation system* shows two different operations that can be requested by the implementation of Move_pallet(X,Y): *Pick_up(Sx1,Sx2,F)* and *Release(Sy1,Sy2,F)*. Any operation at any level can be performed by composition of operations in the immediate lower level. In particular the moving pallet operation can be thought of as composed of two lower level operations: taking pallet and depositing it.

In both cases parameters are Sx1, Sx2 (or Sy1, Sy2, whose meaning is the same) and F. Sx1 and Sx2 are the set points associated to position X. The conversion of the position into the data has to be realized into Move_pallet(X,Y) requiring a buffer operation: *Get_position_set_points(X)*. In the case the buffer has a planar layout the second parameter is not significant. F has the purpose to distinguish which fork has to be used to perform the operation, merely in the topologies with double fork.

The pallet transportation system is composed only of axes, from the minimum of two to the maximum of three. The former case corresponds to a rotating multi-pallet unit, while the latter case includes all the other topologies. For example in a circular buffer pallet unit with double fork three are the possible movements: rotation of the

transportation system, advancing of a fork (and consequently retraction of the other one, since they are integral) and up-down movement.

The generic axis can perform only one operation named *Set_axis(x)*, in which the given set point x can be either an analogue value or a discrete one, on the basis of the motion control that have been chosen.

The framework exhibits the axis composed of *actuator* and *sensor* (the latter in a generic number). The characteristics of the chosen actuators and sensors, namely an actuator endowed of a logic or a modulating control and sensors returning real or logic values, define the axis typology.

The actuator provides two different operations: *Move(x,v)* and *Stop()*, with the obvious meaning; the parameter v enables the opportunity to modify the predefined speed profile. The sensor exhibits only one operation: *Check_position()*, which is requested by axis in order to obtain the actual value at every time.

Usually more sensors than the strictly necessary for nominal working are inserted, with the purpose to make the error detection easier during the system operation. They are usually set on the actuator or along the driving chain, always on a fixed part for wiring issues.

Operations of control modules correspond to methods of objects in the procedural model. Since each object in Fig. 9 can be thought of as realized by co-ordination of the described sub-modules, each object have been designed as specified in Sect. III and Sect. IV. Namely, as stated in Sect. III, each object in the portion of procedural framework for the pallet unit has been modelled by a control module. It has been endowed of the operation manager (the same for all the objects) and of an operation body for each operation, with the proper exception for the buffer; then a unique state manager and a unique alarm generator have been added.

The different state manager patterns shown in Sect. IV can be used in this way: the pallet unit is modelled by a single waiting state MFSM; the axis is modelled by a finite number waiting state MFSM; the pallet transportation system and the actuator are modelled by a two waiting state MFSM. The two waiting states in the case of pallet transportation system model the wait for a pick up operation and the wait for a release operation while the two waiting states in the case of an actuator model the wait for a move operation and the wait for a stop operation.

It is worth mentioning, by a general overview, that within the obtained framework each element can interact only with one higher level element. As a consequence, the model exhibits a hierarchical structure, which is particularly useful in a simulation perspective. The lack of cross-references among the different elements makes it possible to conceive control system design in a structured fashion. Lower level elements, in fact, might be inserted at first only *via* virtual models so that partial verification of the control code correctness can be performed.

VI. Concluding remarks

In the present paper, the first modelling results obtained for the control of machining centres have been shown. A fundamental control module structure has been proposed, that is described formally with the Modular Finite State Machine formalism and that corresponds to the behaviour of a control agent. Prototypical implementation of the proposed model in JADE (Java-based Agent Development Environment), an execution platform for agents, has been positively tested. Future works include the full application of the model to whole plants and an automatic translation of MFSM models into agent-based implementation. Also, automatic translation from SFC to MFSM's and self-diagnosis will be further investigated.

Acknowledgment

The authors are thankful to MCM spa, Italy, for the pictures and precious information provided, and in particular to Dr. Giuseppe Fogliazza for his constant support to the illustrated activities.

References

[1] E. Almeida, D. Tilbury, "Automatic Logic Generation for Reconfigurable Cell-Based Manufacturing Systems," *WODES'04*, Reims, France, September 22-24, 2004.

[2] R.W. Brennan, M. Fletcher and D.H. Norrie, "An Agent-Based Approach to Reconfiguration of Real-Time Distributed Control Systems," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 444-451, 2002.

[3] A.W. Colombo, R. Schoop, P. Leitão and F. Restivo, "A Collaborative Automation Approach to Distributed Production Systems," 2nd *IEEE Int. Conf. on Industrial Informatics*, pp. 27-32, 2004.

[4] S.M. Deen, *Agent-based manufacturing*. Springer Verlag, Berlin, 2003.

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

[6] E. W. Endsley and D. M. Tilbury, "Modular Finite State Machines for Logic Control," *in: Proc. of the IFAC Workshop on Discrete Event System*, 2004..

[7] L. Ferrarini and G. Fogliazza, "Advanced Control System Design for Machining Centres," *IEEE/ASME (AIM 2001)*, 8–11 July 2001, Como, Italia, pp. 671-676, vol. I, 2001.

[8] L. Ferrarini, C. Veber and G. Fogliazza, "Modelling, Design and Implementation of Machining Centres Control Functions with Object-Oriented Techniques," *IEEE/ASME (AIM 2003)*, 20-24 July, 2003, Kobe, Japan, p. 1037-1042, 2003.

[9] B. S. Heck, L. M. Willis and G. J. Vachtsevanos, "Software Technology for Implementing Reusable Distributed Control Systems," *IEEE Control Systems Magazine*, pp. 21-35, Feb. 2003.

[10] Standard IEC 61499, *Function Blocks for Industrial-Process Measurements and Control System*, IEC TC65/WG6, Draft, 18/9/96.

[11] Torero Project, IST-2001-37573, Total life cycle web-integrated control, funded by the European Community under the Information Society Technology Programme (1998-2002). Available: http://www.torero-project.com