Available online at www.sciencedirect.com
ScienceDirect
Robotics and Computer-Integrated Manufacturing 23 (2007) 94–106
Robotics and Computer-Integrated Manufacturing
www.elsevier.com/locate/rcim

Reconfigurable control of robotized manufacturing cells

Manfredi Bruccoleri*

[a]Department of Manufacturing, Production and Management Engineering, Università di Palermo, Viale delle Scienze, 90128, Palermo, Italy

Received 11 December 2003; received in revised form 19 July 2005; accepted 6 August 2005

Abstract

This paper investigates the field of manufacturing system control. The addressed subject is indeed very fascinating, due to the importance that it has reached in the last decades both at research and industrial level. On the other hand, it seems to the author that most of the complexity intrinsic to the subject itself relies on the different meanings or levels of abstraction that both the terms "manufacturing system" and "control" may symbolize. The presented research aims to face the topic in a concrete fashion, i.e., by developing a control software system for a specific, although easy to be generalized, robotized manufacturing cell. Two different development methodologies, from the conceptual design to the actual implementation, of a cell control system are presented and compared. The former, based on ladder logic diagrams, for a PLC controlled manufacturing cell; the latter, based on object-oriented modeling and programming techniques, for a PC controlled manufacturing cell. The analysis has been conducted considering the internal and external requirements of the manufacturing system, mostly driven by the contemporary industrial need of reconfigurable control systems, largely acknowledged as the critical key to succeed in the new era of mass customization.

Keywords: Object-oriented modeling; Reconfigurable control; Interlocking logic; PLC-based control; PC-based control

1. Introduction

Monitoring and controlling computer integrated manufacturing (CIM) systems is a complex but crucial task. Modern computerized manufacturing systems have lagged far behind what could be achieved with existing technology. The cost of such systems is often too high and it should be justified by a return on investment, which can be gained only if the required system flexibility is assured. Although flexibility is mainly gained by the utilization of shop floor control software modules for planning, scheduling, dispatching, coordination, and data monitoring-analysis activities, from a low-level control perspective the utilization of programmable devices that control machines, robots, and material handling systems is also required. At this level, the control system has to perform mainly low-level coordination activities (task planning activities) such as synchronization of shop-floor devices like actuators and sensors. Considering a robotized manufacturing cell, the control issues concern the coordination of the cell hardware components in order to achieve a given task plan. In most cases, the planner is a sequence controller that controls I/O variables by using simple algorithms or programs. These sequence and synchronization controllers are usually programmed in the PLC language or in a common general purpose programming language (if the task plan is run by a PC). Ladder diagrams (LD) are the most utilized modeling and programming language for PLC-based controller. They are very easy-to-use and relatively familiar to the shop floor personnel. Also, Petri nets (PN), state-charts, and finite state machine diagrams have been extensively applied for preliminary phases of the control system development such as specification, design, verification, and performance evaluation of discrete event control systems, despite they employ a process-based model, which does not fully correspond to the control programming behavior [1,2]. As far as the main requirement is to develop reconfigurable cell-level control software that is reconfigurable in its basic components, object oriented (OO) modeling techniques are widely proposed in the scientific literature for the conceptual

*Tel.: +39 091 6657036; fax: +39 091 6657039.
E-mail address: manbru@dtpm.unipa.it.

0736-5845/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rcim.2005.08.005

modeling phase of the control software development because of their well recognized features related to software modularity, rapid prototyping, and re-use. Two main concerns arise yet.

- The first is related to the significant gap which exists between the object oriented conceptual model or design of the control software and its actual implementation. While general purpose programming languages, for PC-based control software, encapsulating object oriented feature are surely available (e.g. C++), concerning a PLC based control system, ladder diagrams, for instance, do not include object oriented features.
- The second concern regards the unconditional need of a simulation environment where to test the effective operation of the new or the reconfigured control software. Indeed, an incorrect synchronization between two cell components (a robot arm and a machine tool working table) could damage the cell unrecoverably. Thus, a simulation environment where to test the modeled and programmed control software is strictly required to avoid such unwanted effects.

The aim of the research presented in this paper is mainly focused on proposing an integrated methodology which adopts an object-oriented approach for modeling, designing, simulating and, finally programming the control software of a robotized manufacturing cell in both PLC and PC-based embedded control systems. Specific attention has been given to the two concerns above mentioned.

The author proposes the unified modeling language (UML) as tool for the design and modeling of the control system itself, while LD and C++ for its implementation, respectively, in PLC and PC-based embedded control systems. Also, it is shown how the use of UML and its activity diagrams makes easier the trade off between the design and the programming phases both in a PLC and PC environments. Furthermore, the paper shows the development of a discrete event simulation platform for testing reconfigurable control software.

The paper is structured as follows: Section 2 gives an overview of discrete control of robotized manufacturing cells, focusing specifically on the most used design and implementation tools. Section 3 presents the problem statement while the proposed control system design methodology is illustrated in Section 4. Section 5 presents the development and implementation methodologies for both the PLC-oriented LD-based control system and the PC-oriented C++ based control system. Conclusions and further remarks are drawn in the last section.

2. Discrete control of robotized manufacturing cell

The term "manufacturing cell," also referred to as "group technology cell," usually connotes a collection of manufacturing, material handling, control, and auxiliary equipments that are needed to manufacture a specific part type or, more generally, a part family. A robotized manufacturing cell is usually a typical manufacturing system of this type and, depending on the manufacturing goal, on the required flexibility, automation, and productivity, it can consist of different numbers and different kinds of manufacturing equipments (machining stations), auxiliary fixtures, material handling systems (robots and transportation systems), and control systems (relays, PLCs, PCs, etc.). In particular, the cell control system architecture mainly depends on the structure and configuration of the cell itself. As an example, if the manufacturing cell consists of many and different kinds of equipments, its control system probably needs to include several hardware devises like PLCs or PCs. Also, depending on the functionality requirements of the cell governance policy, the control software could consist of various kinds of modules. For instance, if the only requirement is the synchronization of actuators and sensors for running a given task plan, then a simple program can be loaded into the PLC, which controls both the input signals originating from the cell sensors and the output signals which trigger the cell actuators. On the other hand, if the control system is required to automatically download the NC part processing program from a database according to the information originated by an automatic vision system which detects the part type entering the system, or some error recovery functionality is required, then it is obvious that the control system needs to include different hardware devices and different software modules. Usually, the control system software architecture complexity is related to its hardware configuration even if this is not always true.

Given such considerations, the scientific literature does not lack of studies and proposals of assorted control architectures (both hardware and software) whose challenge relies on the proposal of generic control systems for generic manufacturing cells. Essentially, two main classes of approaches proposed for cell control system development can be recognized into literature.

- The former deals with the problem of modeling and analysis of the control system software and hardware architectures. Usually, starting from a requirement analysis, authors propose control system architectures based on modeling tools and standards. Object oriented techniques [3,4], IDEF methods, pattern languages [5], and Petri nets [6] are doubtless the most utilized modeling standards for centralized control system architectures, while multi agent systems [7] and IEC standards, like IEC614999 [8], are widely used for decentralized control architectures.
- The latter class of approaches to manufacturing cell control development concerns its low level programming. Even in this case, the programming approaches are numerous and mainly depend on the control system hardware configuration as illustrated in what follows.

2.1. PLC-based embedded control systems

A very conventional way to control a robotized manufacturing cell is by a PLC device, which interacts with every other element by exchanging electrical signals. As an example, it can trigger or stop discrete event sequences on a machine tool according to the ladder logic it is executing. In other words, the PLC not only controls basic actuators but also the synchronization of production processes among many manufacturing and auxiliary elements, which can also be programmable. The most common programming language for PLC is ladder logic. Ladder logic differs from the most common general-purpose languages such as C++ or Visual Basic, being a *device-oriented* programming language. In its essential form, the ladder logic is a graphical representation of Boolean switching functions based on an analogy to physical relay systems.

As far as PLC-based embedded control systems are concerned, the topic of the limited and inflexible capability of their programming languages is largely treated in literature. Following these directions authors propose Petri net models [1,9], object oriented models [10], state-chart diagrams [11,12] that, afterward, need to be, automatically or manually, converted in ladder logic. This pre-programming phase is needed in order to have a higher level model of the control program, which is easier to understand and easier to design, due to its process oriented nature which reproduces the discrete event functioning of the manufacturing cell. Lee et al. [13] by using object oriented models and state charts developed a virtual prototyping environment to reduce the risks involved in PLC-based control programs, such as deadlock problems. Also, expert systems modules [14] and layered Petri nets [15] are used when additional features are required to the control program of the cell task plan, such as error handling capabilities.

2.2. PC-based embedded control systems

On the other hand, in the very recent years researchers are facing the problem of cell control programming directly bypassing the limited capabilities of the PLC with a PC-based control approach. PC-based control environments bring indisputable advantages respect to PLC-environments especially concerning their networking predisposition, real-time monitoring and visualization capabilities, and flexibility related to the great number of programming languages that could implement the real control software. However, in these environments the main problem is to deal with the software and hardware incompatibility arising from various vendors. The Open Modular Architecture Controller (OMAC), instigated by GM and Ford, the Open Systems Environment for Control (OSEC), developed within the machine tool Industry, and the Open System Architecture for Controls in Automation systems (OSACA), developed within the ESPRIT III European project, are signals of strong interest in these fields.

Following these directions, Apte and Zeid propose a Java-based control model by using PC standard parallel ports for asynchronous I/O such as IEEE 1284-A, or peripheral driver integrated circuits, such as Intel's 8255-A, which allows the design of a multiplexed I/O bus [16]. Hong et al., propose a UML modeling tool, C++ programming language, and TCP/IP communication protocol for PC-programming of a robot within a robotized cell according to the OSACA reference platform [17]. Martinez and Garcia developed a visual tool based on sequential function charts, called SFC++, to bridge the gap between object-oriented model and implementation programming for a PC-based distributed industrial control system [2].

3. Problem statement

In order to approach the development of a cell control system, given a desired level of abstraction from the conceptual design, to the modeling, simulation, testing, until its real implementation, four main requirements need to be well defined. This is what here is called problem statement.

The first requirement concerns the identification of the boundaries of the problem, i.e., the specification of the manufacturing system to be controlled, in this case the specific robotized cell.

The second requirement is related to the functional requisites that the control system should provide, such as simple task plan running, or monitoring, or error handling, or supervising, or scheduling, or part quality visioning, or, also, all of these.

The third main requirement concerns the specification of the basic properties that the control system should have. This last requirement depends, of course, on the second requirement and involves decisions on the system hardware configuration, such as centralized/distributed or hierarchical/heterarchical architecture, and on its software features, such as device/process/object orientation or synchronous/asynchronous procedural implementation.

The fourth main requirement regards the possibility to test the control system before its real implementation. Indeed, depending on the first two requirements, it could be strictly necessary to test the correct functioning of the control software in order to avoid critical damages in the manufacturing system.

Not specifying these requirements before developing a control system could lead to the formulation of one of the many generic methodologies which the scientific literature does not certainly lack of. Thus, in what follows, these requirements are specified and described. In such a way, the proposed approach for reconfigurable control software development could reveal more practical and useful for actual circumstances. Also, such specific requirements can be easily generalized besides being quite typical in the shop floor reality.

3.1. The system to be controlled

The manufacturing system test-bed considered in this paper is a robotized cell, which consists of the following components:

- parts to be processed,
- machines to perform processing,
- input and output stations,
- material handling devices and transporters for transferring parts into and out of the robotized cell,
- one control device, such as PLC or PC, to perform the control activities,

In particular the considered manufacturing cell, pictured in Fig. 1, is a STAUDINGER physical toy model, which is installed at the Department of Manufacturing, Production, and Management Engineering of the University of Palermo. The toy mainly consists of three machines, three conveyors (one for each machine), one 3 axis-portal robot, one input station and one output station, and a number of electrical and mechanical sensors and actuators. In particular, the three machines are configured in a line layout and the first is a vertical milling machine (VMM), the second is a multi-spindle milling machine (MSMM), and the third is a horizontal milling machine (HMM).

Basically, a mechanical pusher pushes the work-piece placed into the input station to the processing line, as soon as it receives a start signal from the control system. Three sensors detect the piece position in front of the three processing machines and depending on the piece work-plan the conveyor stops its run exactly when the piece is in front of the right machine where it needs to be processed. A number of sensors and actuators are associated to every machine. For instance the multi-spindle milling machine includes three actuators. The first rotates the tool changer in order to set the requested mill up. The second rotates the working axis in order to execute the real processing operation. The third moves the mill up and down in order to position the mill in contact with the piece to be processed. In the same way, such a machine includes a number of sensors necessary to make the process correctly work. Two sensors detect the vertical position of the mill and one sensor detects the correct direction of the spindle after the tool changer has rotated.

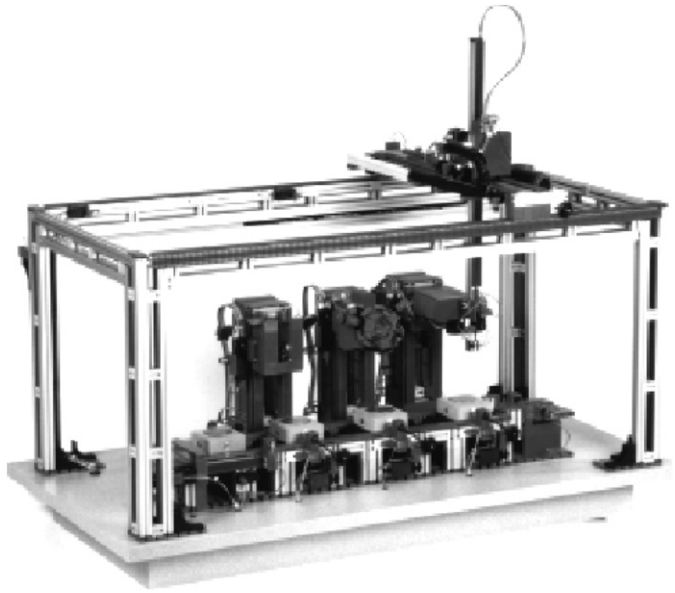

Fig. 1. The robotized manufacturing cell test-bed.

When the work-piece finishes to be processed by the last machine, it is sent to the output station. Because of the mono-directional run of the conveyors, if a work-piece needs to visit two processing machines in a different order then that of the layout configuration, the 3-axis portal robot grips the part in the output station and re-loads it in the input station for a second run. Let us assume that a work-piece should visit first the third machine (i.e., HMM) and then the first one (i.e., VMM). In this case, after having been processed by the HMM, the piece must go to the output station and wait for the robot to load and transfer it again to the input station. In this second run the conveyor will stop the piece in front of the VMM where it will be processed.

3.2. Control functional requirements

The aim of the control system, in this case, is simply to run procedural programs for ordinary task plan executions of a couple of part types of the same group technology family. No integration with other supervising control system, no error handling, no data collection functions are required. The abstraction level of the required automation is very low, as far as the operation of each cell component is driven by a number of actuators while a number of sensors identify its status. Both actuators and sensors exchange digital signals (ON/OFF) with the control system by means of electromechanical relays. Only the robot exchanges with the control system analogical signals by means of an analogical encoder needed in order to correctly define the 3D robot positioning. The coordination of every component for executing the task plan is driven by an asynchronous control mechanism, i.e. an interlocking based control. An interlock is a mechanism for coordinating the activities of two or more devices in order to ensure that the actions of one device are completed before the next device begins its activities. Interlock-based control, in contrast with time-based control, works by regulating the flow of control signals back and forth between the controller and the controlled devices.

For these purposes, two different approaches has been developed and compared. The former is oriented to a PLC-based control, the latter to a PC-based control. In other words, the abstraction level of the required control system, coincides with a stand-alone PLC/PC-based control system for the above mentioned robotized cell toy.

3.3. Control properties: reconfigurable control

Despite the sequence co-ordination capability is the primary required control property, the control system should also be reconfigurable. Indeed, using hardware and software rigid configurations could be an easy solution to perform a specific manufacturing application. However, the redefinition of such applications which includes removing/adding one or more hardware subsystem (such as processing machines) or software subsystem (such as error handling modules) would involve significant efforts in reengineering the system. The control system, even if it performs very low-level control activities such as equipments coordination, must be able to re-define the presence of the hardware devices it controls and the governance functions it performs in an easy way by being hierarchically structured and modularly designed. OO technologies are the paradigm mainly proposed and used for software development in general but also in the specific field of manufacturing system control applications. The characteristics of such a paradigm are perfectly suited for software applications that require reusability, scalability, and reconfigurability of the software components.

However, it is a matter of fact that OO methodologies are widely used during the phase of modeling the control system architecture but quite rarely adopted when it comes to the control software implementation. Indeed, as far as a complex control system is concerned, such as a large control system for a CIM system, then the OO approach allows the definition of its hierarchical architecture and the relationships among its many software modules (such as dependencies, aggregations, and so on) [18]. On the other hand, concerning the low-level control software implementation it is easy to comprehend that as far as most manufacturing equipments are controlled by PLC devices and PLC can be programmed with specific programming languages (such as ladder logic or sequential function chart or instruction list) OO programming is not practical and easy to implement. That is the reason why in this paper two different control systems are proposed for the governance of the robotized cell. Besides the PLC-based one, which cannot include an object-oriented programming phase, the PC-based control system has been developed in OO fashion from the modeling to the real implementation performed by C++ program that is, of course, PC compatible. Nevertheless, concerning the PLC environment, the paper proposes a methodology for easily translating the OO software design into the LD program.

3.4. Testing the control system

As already mentioned in the introduction of this paper, during the control system development there is an unconditional need of a simulation environment where to test the effective operation of the new or the reconfigured control software. This is even more important for low-level control systems, such as task planners and sequencers, than for high level control modules that control part sequencing or resource allocation to jobs. It is not reasonable, indeed, to imagine testing a new task plan for a specific part processing sequence by directly loading the control program in the PLC or the PC and running it into the robotized manufacturing cell. An incorrect synchronization between two cell components (a robot arm and a machine tool working table) could damage the cell unrecoverably. Thus, a simulation environment where to test the modeled and programmed control software is strictly required to avoid such unwanted effects. Several commercial discrete event software platforms, even specific for shop floor simulation such as Arena® by Rockwell Software, Inc., and eM-Plant® by Tecnomatix, Inc., offer high level simulation languages and environment to model manufacturing systems operation and to test manufacturing control policies. Nevertheless, their simulation languages or graphical notations are obviously different from those used for the real control and then, it is not easy to model without a given level of imprecision, the control software that has to be tested.

On the contrary, the paper shows how, starting from the conceptual design of the control software, a simulation environment can be developed in order to test exactly the control software that should be then programmed into the PLC or PC for the actual control of the robotized cell.

4. Control system design

In this section the OO control architecture of the robotized cell is presented. While the modeling principle guidelines can be applied in a variety of scenarios, the actual design and development are driven by the

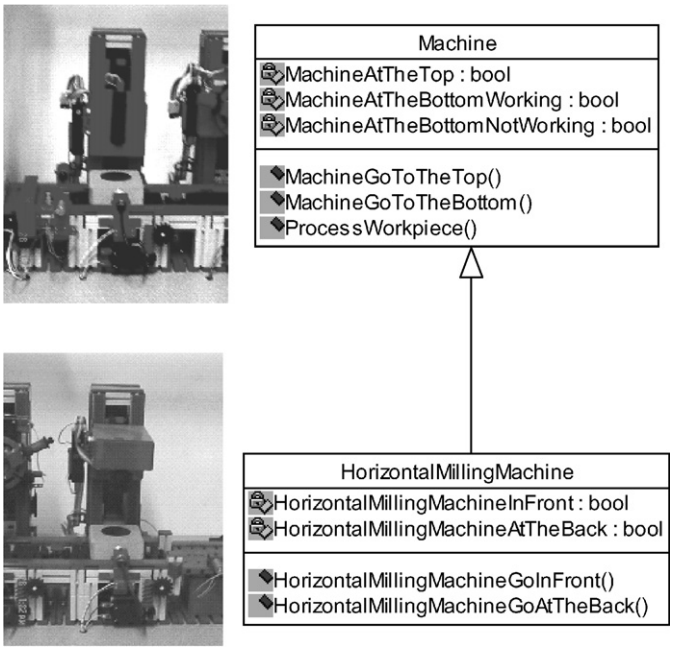

Fig. 2. The real objects and their software classes "machine" and "HMM".

requirements of specific control system, as described in the previous section. Thus, the proposed architecture should then be put into practice by developing the real control software as it is shown in the next section. For the software system design, the UML (standard OO modeling tool for software development) has been used. The UML encloses the advantages of a high level modeling tool (such as PN) and the capability of easy translation from modeling to implementation (such as LD or C++). Furthermore, due to its object-oriented nature, modularity, reusability, and reconfigurability are assured in the software design phase and in the real-time control phase [19]. In the design phase the property of inheritance and instantiation allow easy reuse of software objects. In the real-time control the modularity of the software allows to easy reconfigure the control software itself in order to manage, at a low level control, hardware reconfigurations which are necessary to gain a given level of reactiveness [20].

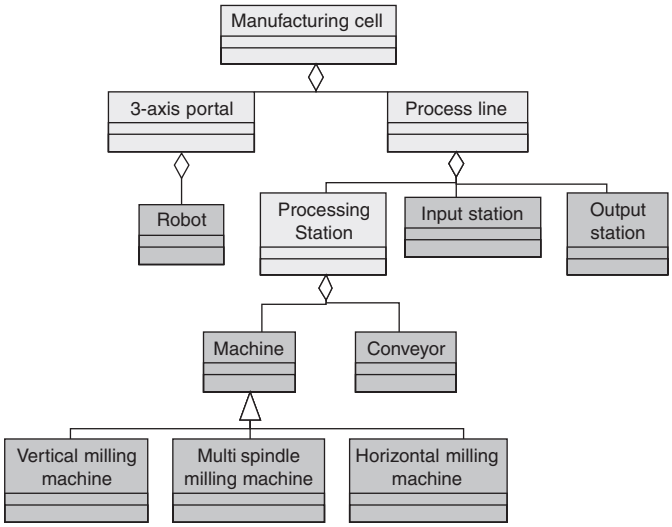

Fig. 3. The control system class diagram.

According to the OO paradigm, in the UML notation, a class of objects is characterized by a set of attributes, whose values represent its status, and a set of operations, which represent the tasks the class is able to perform. Fig. 2 depict the real objects "machine" and "HMM" and their related software class. Notice that, as the HMM is a specific case of the class "machine," it inherits all of the attributes and operations of the upper class "machine". Thus the UML representation of the HMM class shows only its additional attributes and operations plus the generalization/specialization relationship between it and its upper class "machine". So, for instance, the HMM can "go in front" and "go at the back" in addition to the standard "machine" operations "go to the top," "go to the bottom," and "process workpiece".

After having modeled all of the classes participating the system, the next step is to build up the network of classes by representing their hierarchical dependencies and their associations. Such a network of classes and their relationships, called class diagram, embodies the system architecture. As far as the robotized cell test-bed is concerned, its control architecture is represented in the class diagram reported in Fig. 3. In darker color the classes of objects

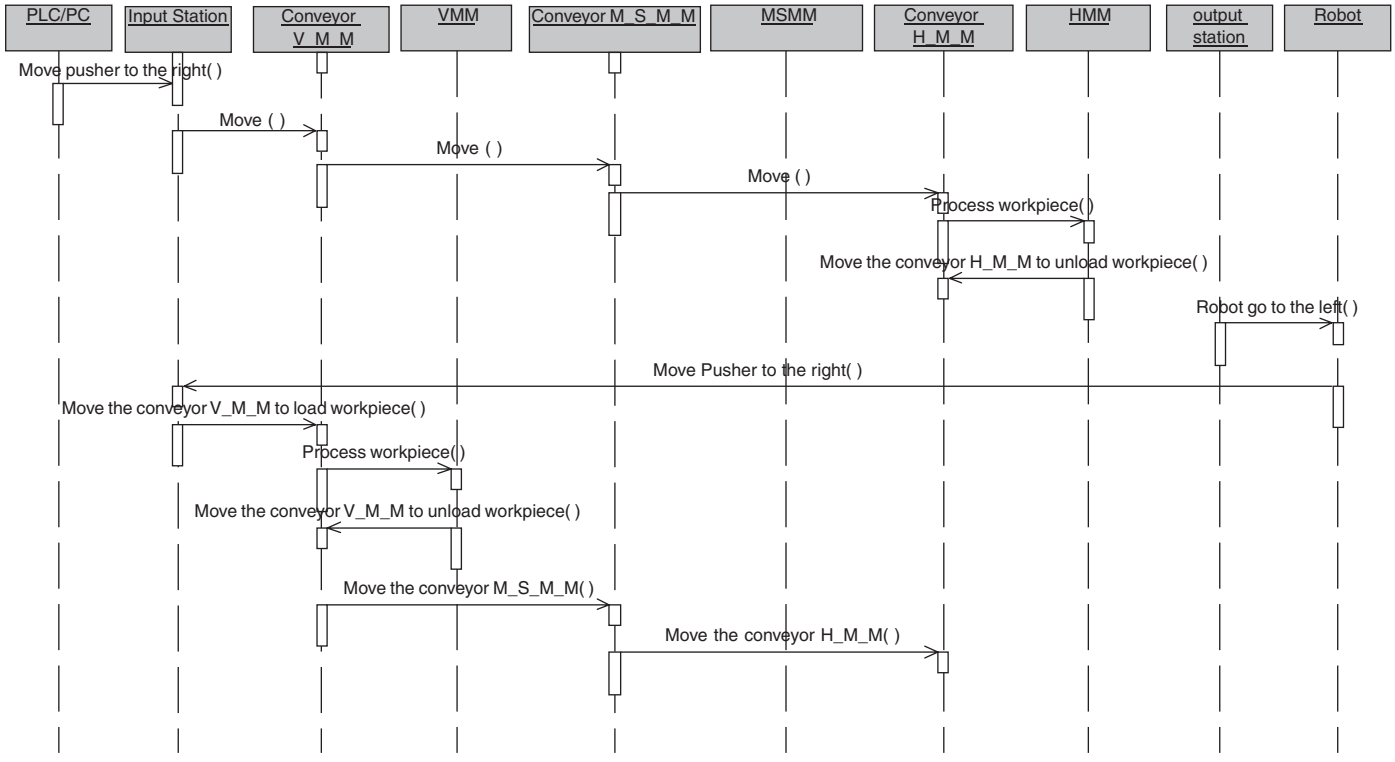

Fig. 4. Sequence diagram for task planning execution.

which actively participate in whatever processing task plan, are reported.

In such an environment, each object is responsible for a specific set of tasks and when an object requires the execution of a task, which is not within its responsibilities, it sends a request, in the form of a message, to another object designed to accomplish that specific task.

Consider the task plan for processing a work-piece whose process plan requires the piece to be processed first in the HMM and then in the VMM, as described in the last paragraph of Section 3.1. In Fig. 4, the flow of messages that are necessary for such specific task plan execution is reported in a UML sequence diagram. The diagram shows the message exchange that is necessary to make the manufacturing cell automatically process that specific part type. Note that every message sent corresponds to the execution of a specific operation (i.e., C++ method or LD action) of the recipient object. Although the sequence diagram shows clearly the sequences of operations needed for the process accomplishment, the interlocking logic of the discrete control system could not be implemented without the design of the correct synchronization of events, activities, and states that characterize such a process. In other words, what still needs to be designed is the sequence of activities triggered by certain events and changing objects states.

Fig. 5 reports the UML activity diagram, which represents the interlocking synchronization among the activities (and states) performed by the input station, the conveyor VMM, and the VMM classes when a work-piece enters the system and needs to be processed in the VMM itself. Note that every state corresponds to a specific configuration of the class attributes, i.e., a configuration of ON/OFF conditions of the sensors belonging to the physical object; on the other hand, every activity matches with a specific class operation, i.e., with the activation of a specific actuator belonging again to the physical object.

A specific state of an object triggers the execution of a specific activity whose fulfillment brings the object to a new state, which will stop the execution of that activity and trigger another one. Alternatively, a timer could stop the activity itself as it happens for the activity depicted in dark color (see Fig. 4). In fact, the VMM keeps processing the workpiece for a specified duration without involving any sensor for detecting its stop. Such an activity is time-based and interlock-free.

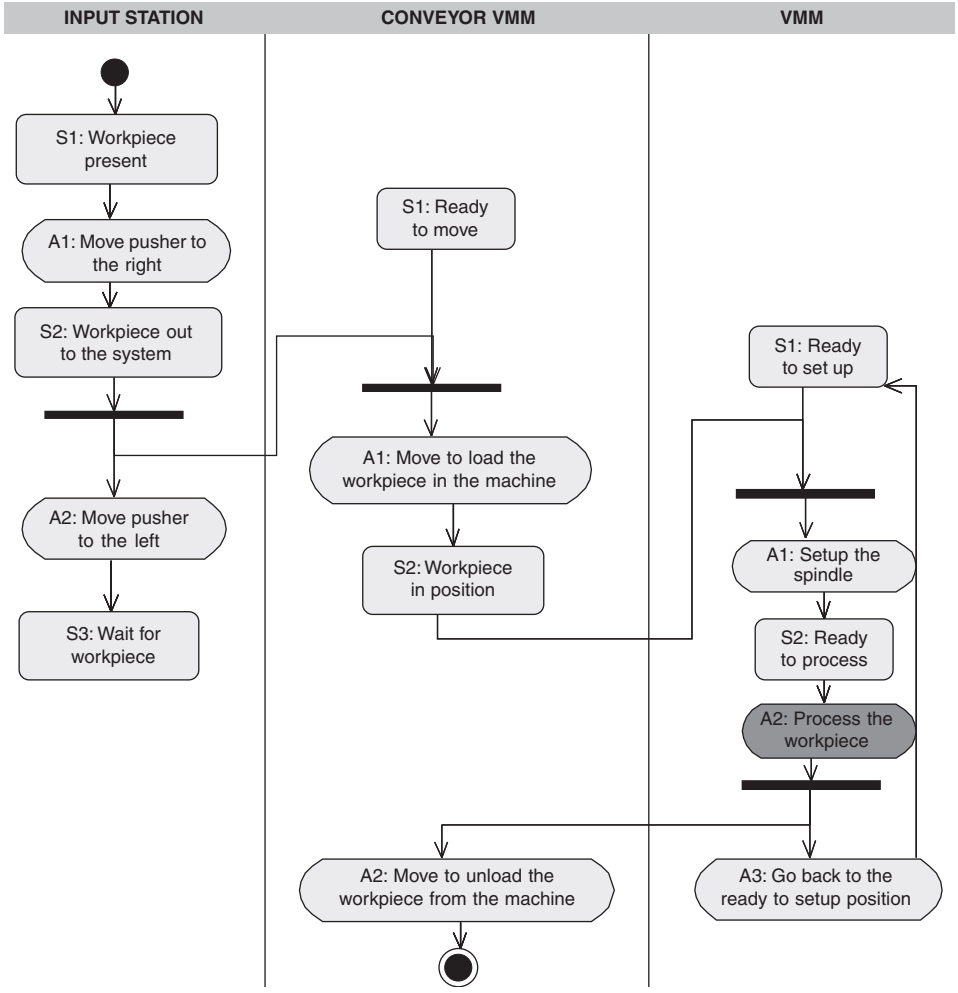

Fig. 5. Activity diagram representing the interlocking logic.

5. Control system implementation

As already mentioned in the previous sections, besides the manufacturing system itself, the main drivers of control software development are the control functional requirements, its properties, and the control system hardware configuration. Depending on this last factor, very different control software could be developed. Because of the very simple function the control system must perform in the application presented here, a single PLC or PC can be used for controlling the manufacturing system. The PLC, or PC, is connected to every sensor and actuator of the manufacturing system (through electro-mechanical relays) and makes the task plan running by a sequence of signal ON/OFF exchanged with these components. In what follows it is illustrated how the control software system has been implemented for both the hardware configurations.

5.1. PLC-based control system implementation

The PLC is usually programmed with specific purpose languages or notations. The PLC used in this application is a SIEMENS S7200 and its programming language Step 7 is based on the classical ladder logic diagrams. The control logic that governs the execution of a given task plan, should hence be translated from the UML activity diagram into the Step 7 notation. Fig. 6 reports a partial view of the Step 7 program that has been written for executing the task plan of Fig. 5 UML activity diagram.

The design of the interlocking logic by using the UML activity diagram surely supports the PLC programmer in understanding and writing the LD code. Indeed, by associating:

- every object to hardware components (machines, robots, etc.),
- every condition related to a specific state of that object (note that, on the UML side, a state corresponds to a specific configuration of the object attributes) to the input conditions connected to that real object (in the LD logic, the input conditions, also referred to as *contacts*, represent ON/OFF devices such as switches or relays), and
- every object activity (note that, in UML an activity is associated with the execution of one specific or a group of object operations) to the output actions (in LD logic, the output actions, also referred to as *loads*, represent motors, lamps, alarm, etc.),

the shop-floor control operator can program the PLC, by simply interpreting the UML activity diagram into the ladder logic diagram.

In particular, Fig. 5 shows that the activity A1 "move pusher to the right" is performed by the object "input station" only when the state S1 "workpiece present" of this object is activated. This is represented in the network 1 of the Step 7 program sketch (Fig. 6) where the *contact* "S1_Input_Station" represents the condition for the *load* "A1_Input_Station". Of course the *contact* "S1_Input_

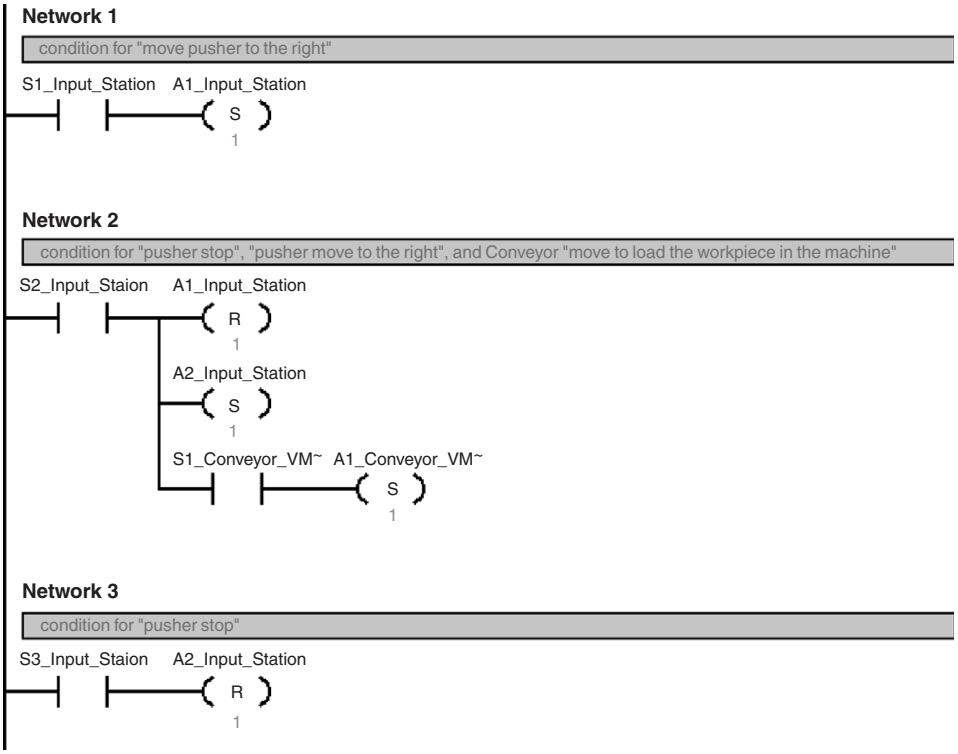

Fig. 6. Partial view of the LD-based Step 7 program.

Station'' is physically connected with the sensor which detects the presence of the workpiece into the input station, and the *load* ''A1_Input_Station'' is physically connected with the electric motor of the pusher.

Despite this simple procedure represents an easy way to translate UML activity diagrams into LD (Step 7, in this case) programs, it cannot be said that during this translation the object-oriented features of the control system design keeps being in the implemented code. This is the author's opinion, the biggest drawback of such kind of control software implementation and, then, of the PLC-based control system hardware configuration. Nevertheless, PLC-based embedded control systems still remain the most used solution for shop floor control, probably because LD are very familiar to shop personnel who must construct, test, maintain, and repair the discrete control system [21]. This is why, it is very important to keep the research efforts exploring this field, although from many aspects a PC-based solution could reveal to be much more promising.

5.2. PC-based control system implementation

While the implementation of the control code within the PLC involves dealing with LD, when it comes to the implementation of the control code within a PC (equipped with I/O card and related hardware to provide the necessary devices to connect to the manufacturing cell equipments), a general-purpose programming language can be selected depending on the preferences (in this application C++ has been chosen). In the simple case in which the controlled variables, i.e., input conditions and output actions, are digital (ON/OFF) variables, then the control program should simply synchronize and interlock binary variables according to activity diagrams such as the one of Fig. 5. Also, the programming phase of the control software is largely simplified by using Rose C++ Analyzer (Rose Enterprise package by Rational), a software tool that allows the designer to obtain software code generation from the UML model.

The final control software consists of two C++ files, a header file and a body file. The former includes the library of all the created classes, their attributes, operations, and relationships. The latter contains all the classes' instances and the exchanged messages that implement the programmed task plans. The C++ generator produces the appropriate C++ class for each class in the UML model. For standard and user-defined operation the generator produces skeletal member functions that must be filled up with member function bodies. Fig. 7 shows the skeleton code generated for the class ''machine''.

As far as the reconfigurable control requirement (Section 3.3) is concerned, of course, a C++ implementation of the control software surely maintains the OO features of its UML design, making preferable a PC-based control system respect to PLC-based one from the reconfiguration requirement perspective. The property of inheritance and

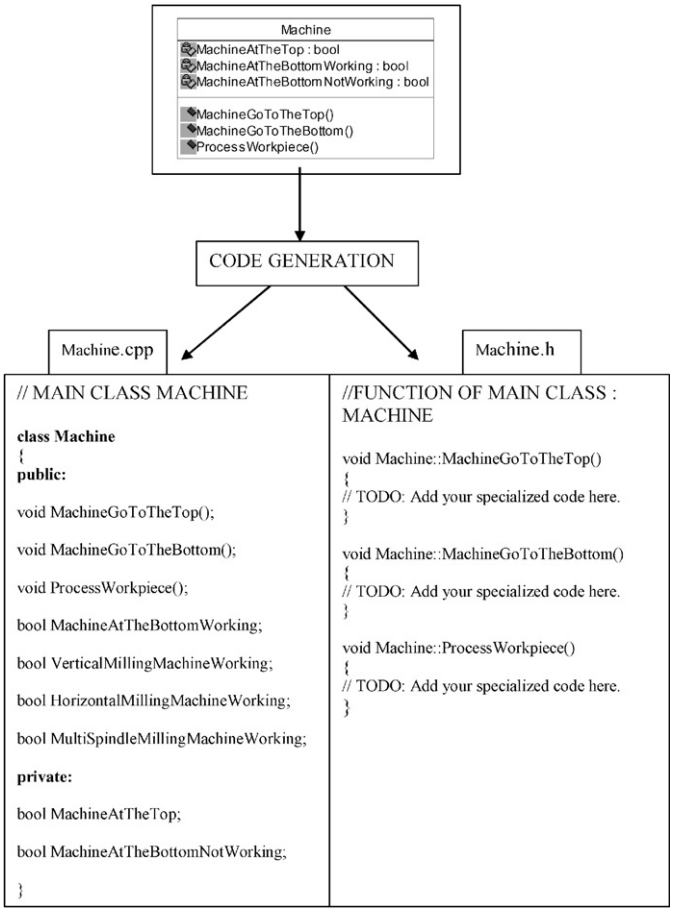

Fig. 7. Skeleton code for the class ''machine''.

the modularity of the control program (the actual code) allow easy re-definition of the hardware configuration of the cell (e.g. adding or removing workstations). Also, given a control procedure for running a specific task plan, let's suppose that a new part requiring a new task plan needs to be processed into the robotized cell. In the control system design, the new task plan will imply only the definition of a new activity diagram which uses classes' attributes and operations already defined. On the contrary, from the programming perspective, if the program is run in the PLC using the LD language the entire program must be re-written, while if the program is written in C++ language and is run by a PC only the C++ version of the activity diagram must be re-written. All of the classes' declarations and definitions can be re-used.

However, the real difference between the two presented approaches (PLC and PC-based) concerns the different functional requirements that the two approaches are able to accomplish. Indeed, besides the coordination of binary variables which make possible the simple task planning execution, the PC-based implementation could easily include some other functional features typical of this kind of control system approach. In particular, they are the real-time visualization/monitoring capability, some exception handling features, and the possibility to test the control system by simulating it before real implementation.

5.2.1. Real-time visualization/monitoring capabilities

Using the commercial Borland, Inc. tool C++ Builder®, a typical windows application has been developed starting from the control software design. Specifically, standard graphical attributes and operations have been added to the software classes of the control system and have been associated to their real control attributes and operations. In this way, every object state or operation also activates some graphical visualization on the computer monitor. Fig. 8 shows the main window of the control system.

The window visualizes, in a real-time fashion, the status of all of the objects of the system and their activities during the execution of a given task plan. For instance, Fig. 9 shows an instantaneous sketch of the vertical milling machine (VMM) status and activities. In particular, the VMM status is represented by the values of its attributes (''Machine at the top'' and ''Machine at the bottom'') and the VMM activities are represented by the operation which is temporarily executing. In the specific instant when the sketch has been taken, its status was ''Machine at the bottom'' and the operation that it was performing was ''Machine is working''.

5.2.2. Networking and exception handling capabilities

Another important functional requirement of the PC-based control system here presented is its ability to react against unexpected events such as out of orders. It is well known that, in control system hardware and software hierarchical architectures, the exception handling phases, such as error detection, diagnosis, and recovery, are usually performed by software modules of the high levels of the control architecture. Also, generally, the error handling strategies are related to the possibility to change the part process routings in order to avoid block states due to the out-of-order manufacturing resource and its temporary unavailability. Such strategies, which are put into operation by control software modules usually called reactive scheduling systems, require the manufacturing system to allow a certain routing flexibility or reconfiguration flexibility [22]. Once the reactive scheduling system finds

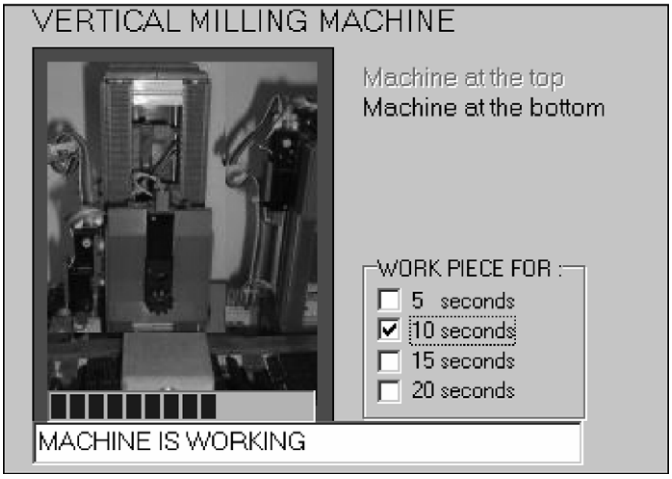

Fig. 9. Monitoring the vertical milling machine.

Fig. 8. Main window of the control program.

a temporary solution against the occurred out-of-order, then a new task plan should be implemented for executing the new part process plan. Real time networking and communication with high level control modules and real time loading of the new task plans proposed by the reactive scheduling system are the two exception handling concerns that involve the low-level control modules responsible of the task plan execution as those considered in this paper.

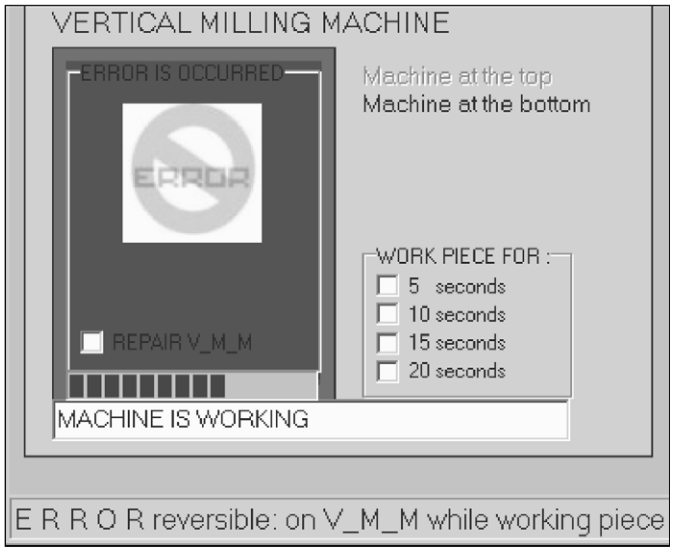

Fig. 10. Error occurrence during VMM operation.

Of course, such functional requirements could not be performed in a PLC-based control environment while can be implemented in a PC based environment as in the one presented in this paper. As an example, Fig. 10 shows a monitor warning due to the occurrence of an error during VMM operation.

The manufacturing cell will stop is run until the operator do not check on the "REPAIR VMM" checkbox which will make the system recover his operation. Meanwhile, if an error handling procedure is ordered by a high-level control module of the control system (such a change in the current part process plan) then the low level control system should adapt to this new status by changing the current task plan. Fig. 11 depicts the control software form which allows the operator selecting the new task plan by simply checking the appropriate checkboxes.

5.2.3. *Control system testing*

Given the requirements already mentioned in Section 3.4, concerning the simulation/testing phase of the control system development, a mixed interlock-based and time-based control program has been realized. In few words, given the interlock-based control software written in C++ described so far, the input and output variables which were connected to the physical relays of the system has been now connected with software classes simulating the physical sensors and motors and governed by C++ *timer* classes which implement the time advancement mechanism. In this

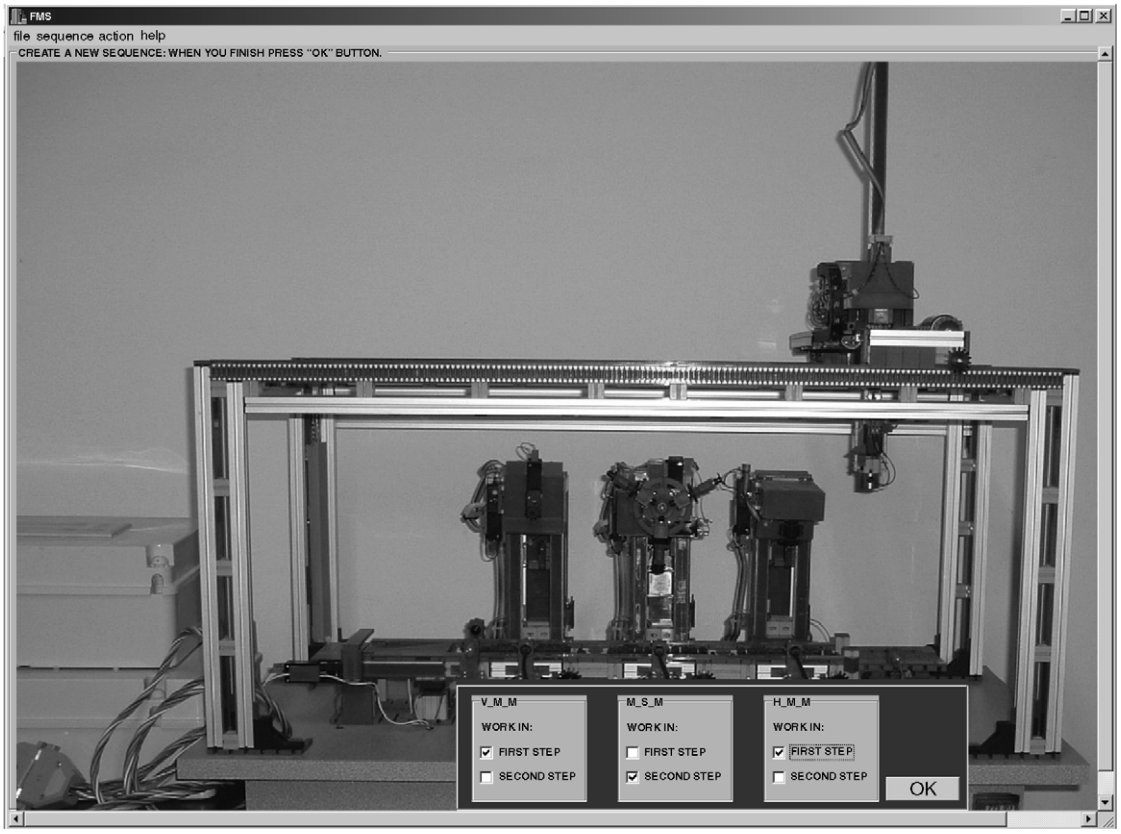

Fig. 11. The control software form "create new sequence".

way by running such a virtual control system, its correct operation can be visualized and tested in the graphical interface already described.

6. Conclusions

This paper describes a study that has been conducted for the development of a control system for a robotized manufacturing cell toy installed at the Department of Manufacturing, Production, and Management Engineering at the University of Palermo. After a preliminary requirement analysis, the study involved the hardware and the software design of the control system and then its implementation aspects. Two different solutions have been proposed and implemented, one PLC-based and the other PC-based.

Regarding the PLC-based solution, although the study showed that the object-oriented design surely facilitates the LD programming phase (typically a shop floor activity), unluckily it takes a considerable effort and time and thus it is not really recommended unless the manufacturing cell is required to process different part type. In this case, its control system, fixed in its hardware configuration, is called for running different task plans and writing many activity diagrams and then translating them into LD is easier than writing directly all the LD programs.

On the other hand, when it comes to the PC-based control system, the author believes that the object-oriented design phase is somewhat crucial because of the possibility to also implement the control software in an object-oriented fashion by utilizing a whatever OO programming language. This feature brings a considerable advantage respect to PLC-based solution. Indeed the control software OO features, both at the design and the programming level, allow, as variously demonstrated in the literature, an easy reconfiguration of the control software [23,24]. This reconfiguration, which can be thought of as the possibility to add, subtract, and reuse software objects, becomes crucial for two main issues. The first concerns the reutilization of software components during the phase of design of the control software; the second regards the management, at low-level control, of hardware reconfigurations, which are necessary to gain a given level of reactiveness. Besides being a crucial enabler for reconfigurable control, the paper shows as the PC-based approach in controlling robotized manufacturing cells is suitable for providing other important functional requirements such as real-time visualization/monitoring, networking integration, exception handling capabilities, and easy testing.

Acknowledgments

The author would like to thank Carlo D'Onofrio, Mario Vitrano and Massimiliano Portella for their contribution during the control software programming phases. The purchase of the robotized cell toy test-bed used in this work was supported by a grant of the Italian Ministry of University and Research.

References

[1] Peng S, Zhou M. Sensor-based stage Petri net modelling of PLC logic programs for discrete-event control design. Int J Prod Res 2003;41(3): 629–64.
[2] Martinez XCP, Garcia RF. SC++: a tool for developing distributed real-time control software. Microprocess Microsyst 1999;23:75–84.
[3] Zhang J, Gu J, Li P, Duan Z. Object-oriented modeling of control system for agile manufacturing cells. Int J Prod Econ 1999;62: 145–53.
[4] Hopkins JM, King RE, Culbreth CT. An object-oriented control architecture for flexible manufacturing cells. In: Computer control of flexible manufacturing systems. New York, NY: Chapman & Hall; 1994 [chapter 16].
[5] Elia G, Menga G. Object-oriented design of flexible manufacturing systems. In: Computer control of flexible manufacturing systems. New York, NY: Chapman &Hall; 1994 [chapter 12].
[6] Yan P, Zhou M, Hu B, Feng Z. Modeling and control of a workstation level information flow in FMS using modified Petri nets. J Intell Manuf 1999;10:557–68.
[7] Odrey NG, Mejía G. Are-configurable multi-agent system architecture for error recovery in production systems. Robot Comput Integr Manuf 2003;19(1–2):35–43.
[8] Vyatkin V, Hanish HM. Verification of distributed control systems in intelligent manufacturing. J Intell Manuf 2003;14:123–36.
[9] Jang J, Koo PH, Nof SY. Application of design and control tools in a multirobot cell. Comput Ind Eng 1997;32(1):89–100.
[10] Pires JN, Sa' Da Costa JMG. Object oriented and distributed approach for programming robotic manufacturing cells. Robot Comput Integr Manuf 2000;16:29–42.
[11] Borchelt RD, Thorson J. Toward reusable hierarchical cell control software. Int J Prod Res 1997;35(2):577–94.
[12] Klein P, Jonsson C. Backstrom, Automatic synthesis of control programs in polynomial time for an assembly line. Proc IEEE Conf Decision Control 1996;2:1749–54.
[13] Lee JI, Chun SW, Kang SJ. Virtual prototyping of PLC-based embedded system using object model of target and behavior model by converting RLL-to-statechart directly. J Syst Arch 2002;48: 17–35.
[14] Hu W, Schroeder M, Starr AG. A Knowledge-based real-time diagnostic system for PLC controlled manufacturing systems. In: Proceedings of the IEEE international conference on systems, man and cybernetics, vol. 4, IEEE, USA, 1999. p. 499–504.
[15] Hasegawa M, Takata M, Temmyo T, Matsuka H. Modeling of exception handling in manufacturing cell control and its application to PLC programming. In: Proceedings of IEEE international conference robotics automation, Published by IEEE, Computer Society, Los Alamitos, CA, USA, 1990. p. 514–9.
[16] Apte N, Zeid I. Evolution of transparent manufacturing: An architecture for a Java-based controller of a CIM cell. J Intell Manuf 2002;13:89–100.
[17] Hong KS, Choi KH, Kim JG, Lee S. A PC-based open robot control system: PC-ORC. Robot Comput Integr Manuf 2001;17:355–65.
[18] Ou-Yang C, Guan TY, Lin JS. Developing a computer shop floor control model for a CIM system—using object modeling technique. Comput Ind 2000;41:213–38.
[19] Bruccoleri M, Perrone G, Noto La Diega S. An object oriented approach for flexible manufacturing control systems analysis and design using the unified modeling language. Int J Flexible Manuf Systems 2003;15(3):195–216.
[20] Koren Y, Heisel U, Jovane F, Moriwaki T, Pritschow G, Ulsoy G, Van Brussel H. Reconfigurable Manufacturing Systems. Annals of the CIRP, vol. 48/2, 1999. Keynote Paper.

[21] Groover MP. Automation, production systems, and computer-integrated manufacturing. 2nd ed. Upper Saddle River, NJ: Prentice-Hall, p. 257–81.
[22] Bruccoleri M, Amico M, Perrone G. Distributed intelligent control of exceptions in reconfigurable manufacturing systems. Int J Prod Res 2003;41(7):1393–412.
[23] Kovács GL, Kopácsi S, Nacsa J, Haidegger G, Groumpos P. Application of software reuse and object-oriented methodologies for the modelling and control of manufacturing systems. Comput Ind 1999;39:177–89.
[24] Kopacek P, Kronreif G, Probst R. A modular control system for flexible robotized manufacturing cells. Robotica 1999;17:23–32.