UML design and AWL programming for reconfigurable control software development of a robotic manipulator

M. Bruccoleri
Dipartimento di Tecnologia Meccanica, Produzione e Ingegneria Gestionale
Università degli Studi di Palermo
Viale delle Scienze - 90128
manbru@dtpm.unipa.it

C. D'Onofrio
Dipartimento di Tecnologia Meccanica, Produzione e Ingegneria Gestionale
Università degli Studi di Palermo
Viale delle Scienze - 90128
cdonofrio@dtpm.unipa.it

U. La Commare
Dipartimento di Tecnologia Meccanica, Produzione e Ingegneria Gestionale
Università degli Studi di Palermo
Viale delle Scienze - 90128
ulacomma@dtpm.unipa.it

F.M. Raimondi
Dipartimento di Ingegneria dell'Automazione e dei Sistemi
Università degli Studi di Palermo
Viale delle Scienze - 90128
raimat@dias.unipa.it

Abstract

The goal of the presented research is to face the topic of reconfigurable control software development in a concrete fashion, i.e., by presenting a control software system development approach which has been used for a specific, although easy to be generalized, robotized manufacturing cell component. In particular, a methodology for the control software development of a planar robot (2-degrees of freedom) is presented, from the conceptual design to the actual implementation. The methodology suggests UML and object-oriented modeling and programming techniques for the design phase, while AWL programming language run by a PLC for the implementation phase. The analysis has been conducted considering the internal and external requirements of the manufacturing system which comprises the robot, mostly driven by the contemporary industrial need of reconfigurable control systems, critical key to succeed in the new era of mass customization.

1. Introduction

A control system needs to confer to the manufacturing system capabilities for easy upgradability and integrability with new components. The manufacturing system should be able to be easily re-configured to process different part types or at different production rates. Flexibility and re-configurability are primarily based on easy usability, modularity, reusability, and scalability of the control system itself. While for high level control activities (SCADA, scheduling, planning, etc.) many approaches for developing flexible and reconfigurable control software can be found in literature, the low level control system still presents some difficulties.

From a low-level control perspective, the manufacturing 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), functional block diagram FBD), structured text (ST), and instruction list (IL) 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. 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 modeling phase of the control software development because of their well recognized features related to software modularity, rapid prototyping, and re-use. Indeed, these features represent crucial enablers for control system reconfiguration and flexibility.

However, one main concern arises. It is related to the significant gap which exists between the object oriented conceptual model or design of the control software and its actual implementation. Indeed, 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, AWL instruction list programming language, for instance, do not include object oriented features.

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 and designing the control system encapsulating reconfiguration capabilities and the AWL instruction list programming language for its implementation. Specific attention has been given to the concern above mentioned related to the integration of the two development phases.

The authors propose the unified modeling language (UML) as tool for the design and modeling of the control system itself, while AWL (the standard Siemens® for IL representation) for its implementation in PLC-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.

The paper is structured as follows. Section 2 overviews the context of the research presented in this paper. The control system requirements and the description of the test case which has been used in this research are presented in section 3, while section 4 shows how the proposed methodology has been applied for a PLC-oriented AWL-based control system. Conclusions and further remarks are drawn in the last section.

2. Research context

A robotized manufacturing cell consists of 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.

Depending on the manufacturing goal, on the required flexibility, automation, and productivity, a robotized manufacturing cell 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 depends mainly 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 the control system needs to include different hardware devices and different software modules.

A very conventional way to control a robotized manufacturing cell is by PLC devices, which interact with every other element by exchanging electrical signals. As an example, a PLC 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 languages for PLC are the standards IEC 1131-3, like LD, ST, FBD, and IL. All of them differ from the most common general-purpose languages such as C++ or Visual Basic, being devise-oriented programming languages. For instance, 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], [2], object oriented models [3], state-chart diagrams [4], [5] that, afterward, need to be, automatically or manually, converted in the PLC programming language. 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., 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 [6]. Also, expert systems modules [7] and layered Petri Nets [8] are used when additional features are required to the control program of the cell task plan, such as error handling capabilities.

3. Control system requirements

The research context described in the previous section highlight that, in order to approach the development of a manufacturing control system, three main requirements need to be well defined.

- 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 manufacturing cell component, i.e. the planar robot.
- 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.

In what follows, these requirements are described in detail.

3.1. The system to be controlled

The system to be controlled is a robotic manipulator with 2 degrees of freedom represented by two arms coupled by a brushless engine (see figure 1).

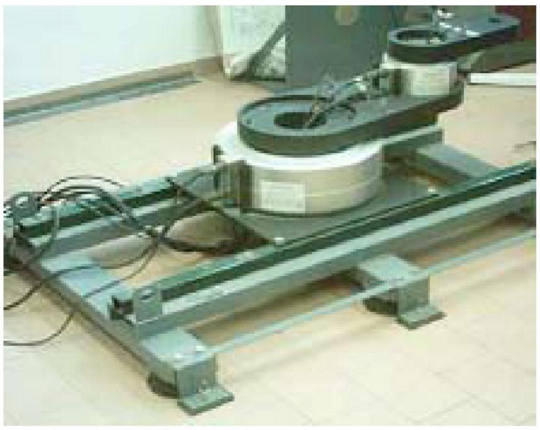


Fig. 1: The robotic manipulator

The bottom arm of the robot is fixed against the basement through another brushless engine, while the upper arm holds a grip for workpieces manipulation (end-effector). Also, two resolvers are used to measure the angular position of the robot.

The robotic manipulator includes two magnetic stroke-ends to limit the arms' rotation under 240°. The arms' rotation is performed by using two drivers, depicted in figure 2, coupled to the brushless engines.

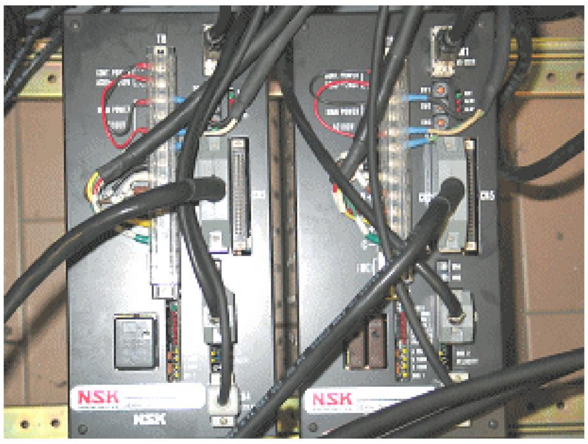


Fig. 2: Drivers coupled to the brushless engines

Each driver communicates to the central controller (in this case the PLC) the desired position of the robot through a serial port RS232C. For instance, by using the command string AD "data" the engine rotates the arm of the angle specified in the field "data" from the specified home position and in a specific rotation versus.

Through the RS232C port the shape of the velocity function is also communicated. This could be trapezoidal (as showed in figure 3) or simply sinusoidal.

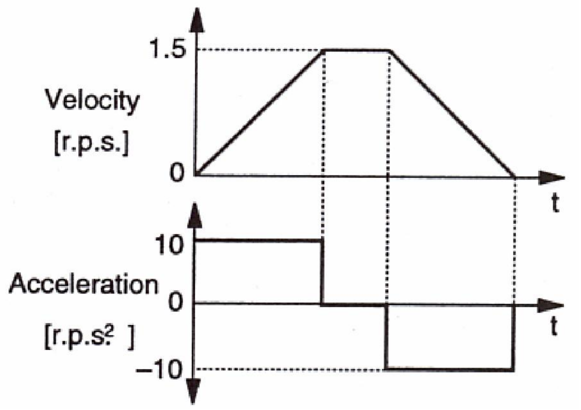


Fig. 3: Example of velocity and acceleration time laws

3.2. Control functional requirements

The aim of the control system, which has been developed, is simply to run procedural programs for transferring by means of the robot arms worpieces from a working station to another working station of the manufacturing cell. 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 operations of the robot are driven by 2 brushless engines while a number of sensors identify its status. Both engines and sensors exchange digital signals (ON/OFF) with the control system by means of electromechanical relays and analogical signals by means of the resolvers needed in order to correctly define the 2D 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.

Summing up, the abstraction level of the required control system, coincides with a stand-alone PLC-based control system for the above mentioned robotic manipulator.

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 working station) 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. Object-oriented (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) [9].

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 ease to implement. That's the reason why, the paper proposes a methodology for easily translating the OO software design into the AWL program.

4. Control system development

4.1. Control system design

As already mentioned, the control system which needs to be designed correspond to a task program which coordinates the sequence of actions of the specific robotic manipulator for workpiece transfer among working stations.

In the first step of the control program development, by examining the real objects which need to be controlled and coordinated, the designer needs to identify all classes involved in the system. Figure 4 provides the main class diagram (according to the UML graphical notation), where classes, their relationships, and hierarchy are displayed. All of the represented classes constitute the control system structure.

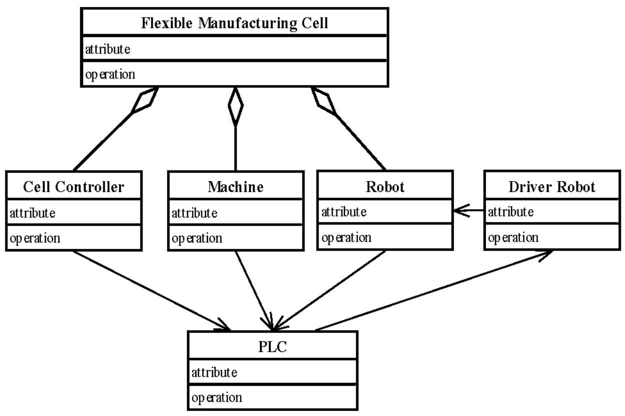


Fig. 4: The control system class diagram

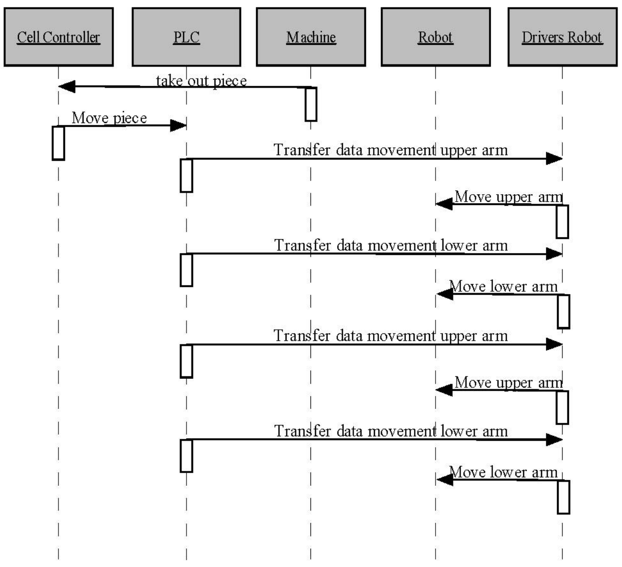
So far, only classes' relationships and their hierarchy have been defined. Yet, to define classes' structures, their attributes, and operations, it is necessary to analyze the dynamic behavior of the system and of its components during the realization of all of the system use-cases. In order to describe the dynamic behaviors and the process workflows in terms of information flows among objects, UML sequence and activity diagrams should be used.

Once the main class diagram has been designed, i.e. the classes of objects (and their relationships) which need to be controlled and coordinated have been identified, the messages that such object should send to each other in order to obtain the specific task program need to be discovered. Such exchanged messages are shown in the sequence diagram of figure 5. Specifically, the sequence diagram reports the sequence of messages that need to be exchanged when the transfer of the workpiece from a working machine to another requires the following movements:

- 110° rotation of the upper arm
- 320° rotation of the bottom arm
- 320° rotation of the upper arm
- 120° rotation of the bottom arm

Initially, the class "Machine" asks for being idled from the processed workpiece and sends to the class "Cell Controller" the message *take out piece*. For this purpose, the class Cell Controller sends the message *Move piece* to the PLC class and this activates the sequence of messages necessary for the robot movements.

For example, the first movement, associated with the message *Trasfer data movement upper arm*, is achieved by sending the string AD110CR to the Robot Driver class which is responsible for the robot real movements.

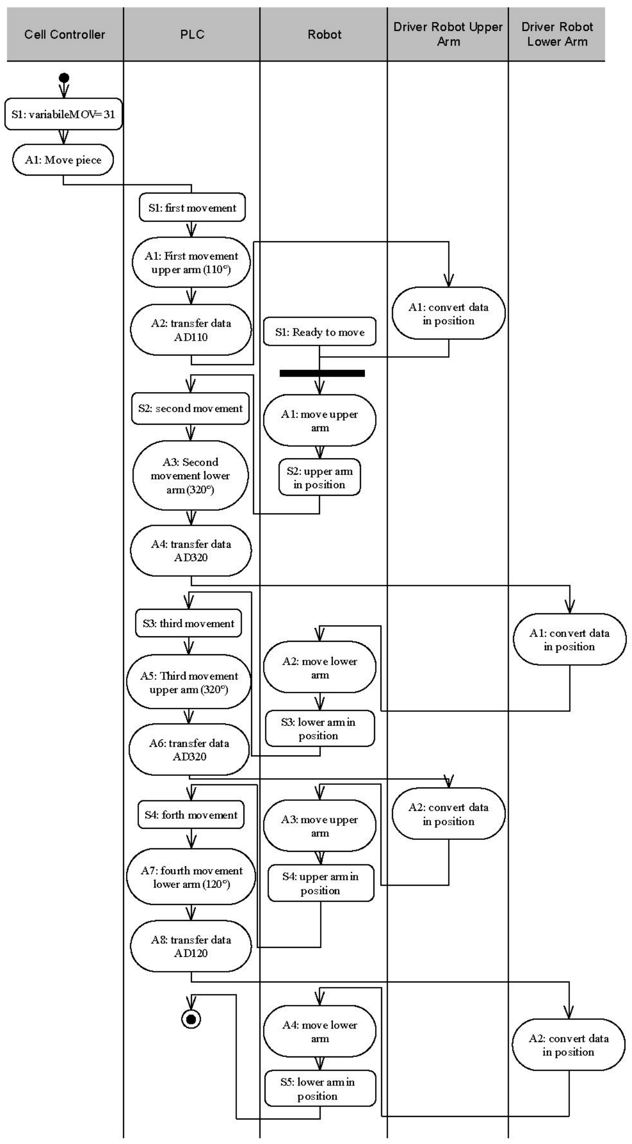


Cell Controller
PLC
Machine
Robot
Drivers Robot
take out piece
Move piece
Transfer data movement upper arm
Move upper arm
Transfer data movement lower arm
Move lower arm
Transfer data movement upper arm
Move upper arm
Transfer data movement lower arm
Move lower arm

Fig. 5: UML sequence diagram for the workpiece transfer operation

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. In order to describe the process dynamic in terms of workflow to be performed, the UML activity diagram has been chosen. In figure 6, the activity diagram for the control process is given.



Cell Controller
PLC
Robot
Driver Robot Upper Arm
Driver Robot Lower Arm
S1: variabileMOV=31
A1: Move piece
S1: first movement
A1: First movement upper arm (110°)
A2: transfer data AD110
A1: convert data in position
S1: Ready to move
S2: second movement
A1: move upper arm
A3: Second movement lower arm (320°)
S2: upper arm in position
A4: transfer data AD320
S3: third movement
A1: convert data in position
A2: move lower arm
A5: Third movement upper arm (320°)
S3: lower arm in position
A6: transfer data AD320
S4: forth movement
A2: convert data in position
A3: move upper arm
A7: fourth movement lower arm (120°)
S4: upper arm in position
A8: transfer data AD120
A4: move lower arm
A2: convert data in position
S5: lower arm in position

Fig. 6: UML activity diagram for the workpiece transfer operation

The activity diagram shows that the workflow is triggered as soon as the Cell Controller class is ready to start the movement sequence. Such state is represented by the value of attribute *variableMOV* equal to 31. At this point the activity *First movement upper arm (110°)*

is activated and the PLC will *transfer data AD110* to the Driver Robot upper arm class. This last class, then, will *convert data in position* and triggers the activity *Move upper arm* performed by the robot class until the state *upper arm in position* is reached.

Once all of the exchanged messages and the corresponding objects' activities and states have been identified, the design of the software classes constituting the system can be completed by assigning attributes and operation to every class as reported in figure 7.

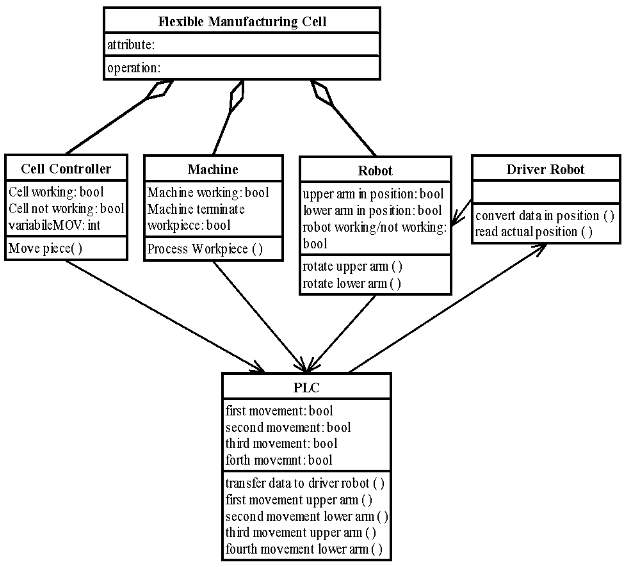

Flexible Manufacturing Cell
attribute:
operation:
Cell Controller
Cell working: bool
Cell not working: bool
variabileMOV: int
Move piece( )
Machine
Machine working: bool
Machine terminate workpiece: bool
Process Workpiece ( )
Robot
upper arm in position: bool
lower arm in position: bool
robot working/not working: bool
rotate upper arm ( )
rotate lower arm ( )
Driver Robot
convert data in position ( )
read actual position ( )
PLC
first movement: bool
second movement: bool
third movement: bool
forth movemnt: bool
transfer data to driver robot ( )
first movement upper arm ( )
second movement lower arm ( )
third movement upper arm ( )
fourth movement lower arm ( )

Fig. 7: UML complete class diagram of the control system

4.2. Control system implementation

In order to implement the designed control system two serial modules CP340 RS232C (see figure 8) have been used as communication channels among the PLC and the brushless engines drivers.

The PLC is usually programmed with specific purpose languages or notations. The PLC used in this application is a SIEMENS S7-300 and its programming language AWL is a classical IL language.

The control logic that governs the execution of task plans, should hence be translated from the UML activity diagram into the AWL notation. Figures 9, 10, and 11 report a partial view of the AWL program that has been written for executing the task plan of figure 6 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 AWL code. Indeed, by associating:

- every object to hardware components (machines, robots, etc.),
- every condition related to a specific state of an object (note that, on the UML side, a state corresponds to a specific configuration of the object attributes) to the input conditions, which in the AWL notation are defined by using the "U" symbol (note that in AWL, the input conditions represent physical input or specific control variables values that identify a given control sequence status),
- 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, which in AWL are defined by using the "S" symbol (note that in AWL, the output actions identify physical output or specific control variables values as the input conditions), and
- every data transfer (note that in UML, a message recalls a given operation of the receiving object by transferring some parameters) to the loading and transferring operations (denoted in AWL respectively with the symbols "L" and "T")

the shop-floor control operator can program the PLC, by simply interpreting the UML activity diagram in AWL.

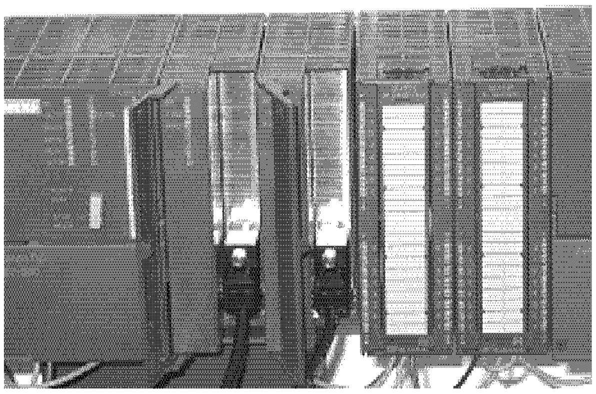


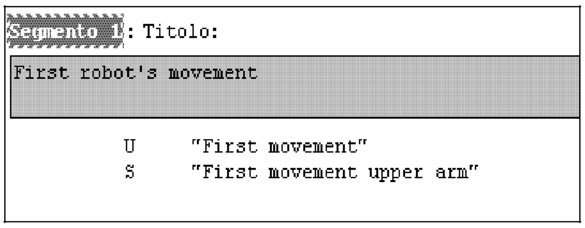
Fig. 8: Serial communication modules CP340 RS232C

Also, the following steps need to be executed:

- Definition of the table of symbols by using the variables which compare in the activity diagram;
- Transfer (by using the commands P_SEND) the strings that are necessary for the arms movement, and which are recorded in a specific database;
- Recall the P_RCV command for the arms position detection;
- Insert additional commands like SPB or other jumping commands to other segment commands, in order to avoid data transferring and/or operations triggering when the specific

state attributes are not activated, as depicted in figure 10.

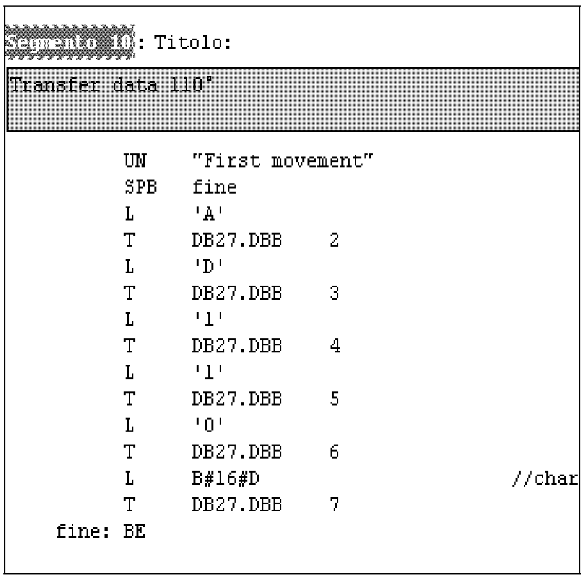
For instance, let's consider the first movement which consists in the 110° upper arm rotation. As already mentioned in the activity diagram description of figure 6, the state S1 "first movement" of the PLC object triggers the activities A1 "First movement upper arm (110°)" and A2 "transfer data AD110". This has been, then, translated in the AWL notation where the input condition (U) "First movement" activates the output condition (S) "First movement upper arm" (figure 9), and also the data AD110 transfer to the database numbered 27 (figure 10). Finally, such data are transferred to the specific driver (figure 11).



```
Segmento 1: Titolo:
First robot's movement

        U     "First movement"
        S     "First movement upper arm"
```

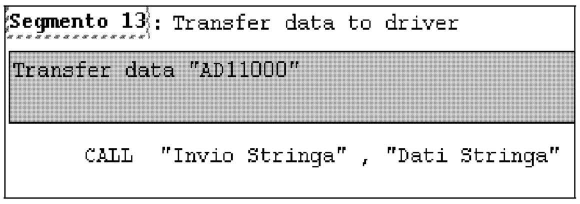
Fig. 9: Sketch of the AWL control program for the workpiece transfer operation



```
Segmento 10: Titolo:
Transfer data 110°

        UN    "First movement"
        SPB   fine
        L     'A'
        T     DB27.DBB    2
        L     'D'
        T     DB27.DBB    3
        L     '1'
        T     DB27.DBB    4
        L     '1'
        T     DB27.DBB    5
        L     '0'
        T     DB27.DBB    6
        L     B#16#D                      //char
        T     DB27.DBB    7
   fine: BE
```

Fig. 10: Sketch of the AWL control program for the workpiece transfer operation

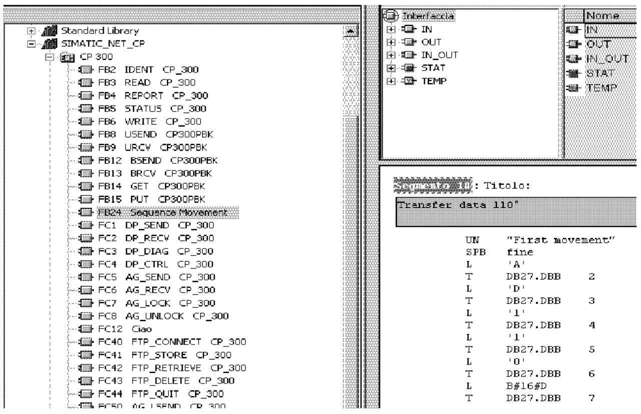


```
Segmento 13: Transfer data to driver
Transfer data "AD11000"

     CALL  "Invio Stringa" , "Dati Stringa"
```

Fig. 11: Sketch of the AWL control program for the workpiece transfer operation

Once the AWL code for the robot movements' sequence has been written, such code can be saved as a specific sub-routine (see "FB24 Sequence Movement" in figure 12) that can be reused in other similar applications by changing only the movement coordinates. In this way it is possible to create a library of sub-routines that can be reutilized in new control software systems' design and development.



```
Standard Library
SIMATIC_NET_CP
  CP 300
    FB2   IDENT   CP_300
    FB3   READ   CP_300
    FB4   REPORT   CP_300
    FB5   STATUS   CP_300
    FB6   WRITE   CP_300
    FB8   USEND   CP300PBK
    FB9   URCV   CP300PBK
    FB12   BSEND   CP300PBK
    FB13   BRCV   CP300PBK
    FB14   GET   CP300PBK
    FB15   PUT   CP300PBK
    FB24   Sequence Movement
    FC1   DP_SEND   CP_300
    FC2   DP_RECV   CP_300
    FC3   DP_DIAG   CP_300
    FC4   DP_CTRL   CP_300
    FC5   AG_SEND   CP_300
    FC6   AG_RECV   CP_300
    FC7   AG_LOCK   CP_300
    FC8   AG_UNLOCK   CP_300
    FC12   Ciao
    FC40   FTP_CONNECT   CP_300
    FC41   FTP_STORE   CP_300
    FC42   FTP_RETRIEVE   CP_300
    FC43   FTP_DELETE   CP_300
    FC44   FTP_QUIT   CP_300
    FC50   AG_LSEND   CP_300

Interfaccia          Nome
  IN                 IN
  OUT                OUT
  IN_OUT             IN_OUT
  STAT               STAT
  TEMP               TEMP

Segmento 10: Titolo:
Transfer data 110°

        UN    "First movement"
        SPB   fine
        L     'A'
        T     DB27.DBB    2
        L     'D'
        T     DB27.DBB    3
        L     '1'
        T     DB27.DBB    4
        L     '1'
        T     DB27.DBB    5
        L     '0'
        T     DB27.DBB    6
        L     B#16#D
        T     DB27.DBB    7
```

Fig. 10: Library of the developed AWL sub-routines

5. Conclusions

This paper describes a study that has been conducted for the development of a control system for a robotized manufacturing cell, and in particular for a specific component, i.e. a robotic manipulator. After a preliminary requirement analysis, the study involved the hardware and the software design of the control system and then its implementation aspects.

The study showed that the object-oriented design surely facilitates the PLC 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 flexible and 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 AWL is easier than writing directly all the AWL programs.

The approach innovation is the proposal of the UML as an object-oriented tool that can be used without other tools for the modeling and design of manufacturing control systems. It has been demonstrated that the UML can support a static and structural modeling of control systems, as well as dynamic one. Since UML offers a complete graphical notation but no modeling methodology, efforts have been focused on the development of a systematic design procedure to make the task of developing the control software system easier.

Also, the control software OO features, allow, as variously demonstrated in the literature, an easy reconfiguration of the control software [10], [11]. 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.

6. Acknowledgements

This work has been possible thanks to the grant of the Italian Ministry of University and Research.

References

[1] Peng S., Zhou M., 2003. Sensor-based stage Petri net modelling of PLC logic programs for discrete-event control design, *International Journal of Production Research*, Vol. 41 No. 3, pp. 629-664.

[2] Jang, J., Koo, P., H., Nof, S.,Y., 1997, Application of design and control tools in a multirobot cell, *Computers & Industrial Engineering*, Vol. 32, n. 1, pp. 89-100.

[3] Pires, J.N., and Sa' Da Costa, J.M.G., Object Oriented and distributed approach for programming robotic manufacturing cells, *Robotics and Computer Integrated Manufacturing*, Vol. 16, 2000, pp. 29-42.

[4] Borchelt, R.D., Thorson J., 1997, Toward reusable hierarchical cell control software, *International Journal of Production Research*, Vol. 35, n. 2, pp. 577-594.

[5] Klein, P. Jonsson, C. Backstrom, Automatic synthesis of control programs in polynomial time for an assembly line, Proceedings of the IEEE Conference on Decision and Control, Vol. 2, pp. 1749-1754, 1996.

[6] Lee J.I., Chun S.W., Kang S.J., 2002, Virtual prototyping of PLC-based embedded system using object model of target and behavior model by converting RLL-to-statechart directly, *Journal of Systems Architecture*, Vol. 48, pp. 17-35.

[7] W. Hu, M. Schroeder, A. G. Starr, A Knowledge-based real-time diagnostic system for PLC controlled manufacturing systems, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Vol. 4, IEEE, USA, pp. 499-504, 1999.

[8] M. Hasegawa, M. Takata, T. Temmyo, and H. Matsuka, Modeling of exception handling in manufacturing cell control and its application to PLC programming, Proc IEEE Int Conf Rob Autom. Publ by IEEE, Computer Society, Los Alamitos, CA, USA, pp. 514-519, 1990.

[9] Ou-Yang C., Guan T.Y., Lin J.S., 2000, Developing a computer shop floor control model for a CIM system – using object modeling technique, *Computers in Industry*, Vol. 41, pp. 213-238.

[10] Kovács G.L., Kopácsi S., Nacsa J., Haidegger G., Groumpos P., 1999, Application of software reuse and object-oriented methodologies for the modelling and control of manufacturing systems, *Computer in Industry*, Vol. 39 pp.177-189.

[11] Kopacek, P., Kronreif, G., Probst, R., A modular control system for flexible robotized manufacturing cells, *Robotica*, Vol. 17, p.p 23-32, Jan-Feb 1999.