JOURNAL OF
# MANUFACTURING SYSTEMS

Volume 2 / No. 1
1983

## CONTENTS



Society of Manufacturing Engineers/Computer and Automated Systems Association of SME
One SME Drive, Post Office Box 930, Dearborn, Michigan 48121

Published by: Society of Manufacturing Engineers • One SME Drive • P.O. Box 930 • Dearborn, Michigan 48121

# Explicit Programming Languages in Industrial Robots

**Giuseppina Gini**, Politecnico di Milano, Milano, Italy
**Maria Gini**, University of Minnesota, Minneapolis, Minnesota

## Abstract

This paper discusses issues of design for software systems for computer controlled manipulators. A short review of the features which have become important in present software systems for industrial applications is presented, including how various desirable system capabilities can be introduced at reasonable computational costs.

The paper is based mainly on the experiences obtained in designing and implementing MAL, a software system for controlling and programming an experimental robot, and VML, a machine independent intermediate language to be used as a target for compilers of high level programming languages for robots.

An explanation of how management of multiprocess capabilities, synchronization of different devices, error handling and other desirable features can be inserted in a simple system, implemented on micro and minicomputers and made suitable for industrial applications will be shown.

***Keywords:*** *Robot Programming, Communication Languages, Task Synchronization.*

In this paper we discuss issues of design for software systems for computer controlled manipulators. The simplest programming method developed for robots is teaching by guiding. Only the meaningful positions and a few functions are stored in a memory, and their sequence can be played back any number of times to repeat the desired movement.

Although this method is quite simple, it has several drawbacks. An error during the teaching phase requires the teaching process to start over from the beginning, unless editing capabilities are available. Teaching of repetitive positions, as positions on a pallet, is too tedious and error prone. Synchronization of the robot with other systems, as loaders or moving belts, can be extremely difficult. Interaction with sensors is quite impossible, unless appropriate extensions are made to the basic method.[1]

During recent years we have seen a significant change in the attitude of manufacturers of robots with respect to this problem. More and more robots are sold with a sort of programming language, allowing the user to write an applicative program, or, at least, to integrate the teaching phase with a debugging activity. More software systems will appear on future robots.

Much time, effort, and resources have been spent in developing different programming systems, for each different robot. One of several available approaches is to take an existent language, FORTRAN for instance, and add to it routines to drive the mechanical devices. This permits the full power of the language to be used, but may require a time expensive process of linking modules.

Another possibility is to write a set of library routines, so that the user program consists of a sequence of calls to these routines in addition to simple control statements. Yet another approach is to design a language specifically for manipulation.

Most of the efforts have been done using the third approach. One of the reasons is that no available language has the desired characteristics for a robot programming language, therefore, no language is a good candidate for extension.

In the meantime, the desirable features for a robot programming language have been identified. The language should be general purpose, to allow any kind of computation from sensors and vision. The language should support cooperation to express cooperative simultaneous operations of multiple robots and devices. Since robots work in a real-time environment, data must be processed within certain time constraints when received. Data may be received synchronously, and the computer should be able to ask for and obtain data at any arbitrary time. The ability to periodically check certain conditions in order to synchronize events which are dependent on those conditions is another important requirement. Specialized data types should be available to ease the expression of manipulator positions in the space.

However, since this great generality may be difficult for the user, a programming environment should be available to support the debugging and testing phases of the applicative programs. Many problems require further investigation as the dreamed independence of the programming system from the physical configuration and the kinematic features of the arm.

The difference between on-line and off-line programming deserves further discussion. On-line systems could provide tools to debug and test programs. However, they require the use of the robot system, which tends to be rather unpleasant in the case in which the robot should be used in production. Off-line programming should allow developing programs without any interaction with the physical world, at the cost of completely losing any form of interaction with it. It is not clear how to deal with sensors, and sensors are too important to be neglected.

While on-line systems may be more directly accessible to inexperienced users, the integration of robots and other machines in a flexible manufacturing system will require more and more off-line programming to make the whole system work without stopping production.

This paper presents a proposal for a classification of robot programming languages. Our classification is independent of the syntactic aspects of the languages, and considers only the ability in expressing robot actions. In this way our proposal could serve as a general purpose scheme.

There are two aspects in a robot programming system. The user language, in which application programs are written, and the run time system, which executes the code that results from compilation or interpretation of a program in the user language.

When a robot is operating without any response to sensory data, a simple run time system can be used to control the robot through a fixed sequence of joint positions. When a sensory response is required,[2] computations of arbitrary complexity are required at run time.

Several responses to data obtained from sensors or vision can be envisaged. For instance, (1) a discrete choice between one action and another is done at run time, (2) sensory values are used to take an action which quantitatively depends upon them, or (3) sensory data are used to continuously control the movements of the robot.[3] The first two are generally available, while the third is much more difficult.

Our experience in developing robot programming systems is described. In particular we will present a software system called Multipurpose Assembly Language (MAL), a BASIC-like language designed for a cartesian robot with two arms.[4] We then extended our system to be used also as an intermediate language. The availability of an intermediate, machine independent, pseudocode can aid the solution of the portability dilemma, making it easier to implement high level languages on different robots.

## Programming Languages

Since computer controlled manipulators have been introduced as a general purpose mechanism for industrial automation, the methodology of controlling and programming them has seen a great deal of development. Some important issues for these systems have gained wide acceptance.[5,6]

Robots should be programmed in a simple way, without extensive users training. This goal has been partially achieved with many industrial robots which are programmed by guiding the arm through the motions of the task and storing the sequence of the obtained positions for further executions.

Teaching-by-guiding has been successful for

tasks where only simple operations or few positions are required. Where complex assemblies are performed, that method does not allow any modification or adjustment of the movements during the execution, and makes it impossible to use force sensors and vision. Even small changes in the assembly station cannot be introduced without repeating all the teaching. It is however, more difficult to write a program to solve this than to guide the arm through the positions it has to reach.

Manipulation and assembly tasks are difficult to program because the expression of movements in terms of manipulator positions requires many details. The intuitive knowledge about physical operations can hardly be expressed in words. For this reason some robot programming languages try to give the system the ability to compute positions and robot control values from a general knowledge about objects and the physical world. The problem is that the user is required to supply specific knowledge. The use of CAD databases could offer an appropriate solution.

The increasing interest in the use of sensory feedback, mainly in assembly operations, makes the availability of a robot programming system a real need.

The robot programming languages can be conceptually divided into classes, according to the level in which operations are expressed. Different authors have proposed partially different classifications.[7,8] In our proposal we will consider joint level, manipulator level, object level, and task level languages.

Joint level languages are at the lowest level of languages. The description of a task is expressed in terms of the control commands required to drive the individual motors and actuators. Each joint is explicitly controlled. This means that the user should program directly in the joint space instead of in the cartesian space. The user should know the geometry of the manipulator, and possibly the law of operations of the motors.

Many available languages fall into this level. This fact is not surprising since it is obviously easier to start with a low level language and improve it little by little. Among these languages are Sigma Language (SIGLA)[9] of Olivetti Company and HELP of Digital Electronic Automation SpA (DEA).

When we refer to low level languages, we should not forget that we mean low level with respect to the expression of manipulator actions. From the point of view of the control structures, HELP is considered in this language class only because it requires the expression of movements in terms of movements on single axes. The MAL language is in this class also. In the case of cartesian robots, the decomposition of movements about different axes is natural, which may explain why so many cartesian robots are in this language class.

Another example is ARMBASIC, the language available on MINIMOVER, which is not a cartesian arm. In this case the movements are expressed in the form of number of steps required for each motor for each axis.

At the manipulator level, we put all the languages in which the user controls the manipulator positions and movements in the cartesian space independently on the arm configuration. There is no explicit representation of the objects present in the real world. A well known example is the VAL language of Unimation.[10] This language allows for the expression of manipulator positions in terms of transforms, which correspond to positions and orientations of the end effector in the space. Even Virtual Machine Language (VML),[11] a language designed as an intermediate code for robotics, has complex data types such as frames, vectors and rotations.

In the object level there are languages which have some knowledge of the objects in the world. This knowledge can be partial, because complete object models are not always needed. Only those features which are relevant for the manipulation task are part of the model. Object models can be used to describe the sequence of operations with less details or to compute collision free trajectories.[12] In this class we may consider languages such as Assembly Language (AL),[13,14] in which objects are represented through six coordinates as rigid bodies in the space, or Robot Automatically Programmed Tools (RAPT),[3] Automatic Programming System for Mechanical Assembly (AUTOPASS),[7] and Language for Automatic Mechanical Assembly (LAMA),[12] in which models are based on geometry. Most of the open research problems in robot programming languages are at this level.

In the task level are systems which are able to understand and execute descriptions of the task. These systems are able to execute tasks even by replanning their actions in order to solve error situations.[15] At this level there are no working

systems, although some of the systems illustrated in the previous class are oriented towards the task level.

The four levels can be briefly illustrated by the example given in *Figure 1*, where some instructions for the different levels and the conceptual operations which transform each level into the lower one, are indicated.

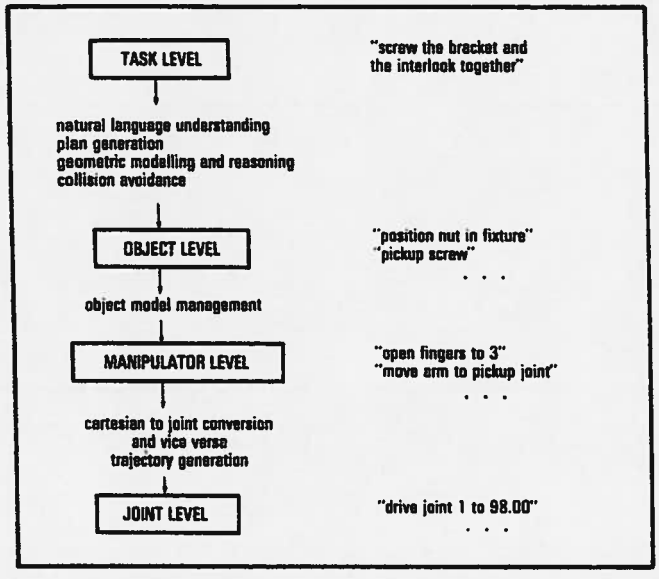


***Figure 1***
**Levels of Robot Programming Languages**

## Design Criteria

The experimental robot is a cartesian manipulator with two arms. Each arm has three degrees of freedom plus the hand opening. The mechanical structure is the same employed in the Sigma of the C. Olivetti Company, while its electronic control has been completely redesigned and implemented with a set of microcomputers.[16]

The structure of the system is illustrated in *Figure 2*. The lack of an adequate control and programming system for our robot convinced us to design and implement a complete software system to support the writing and execution of programs. This system is called Multipurpose Assembly Language (MAL).[4]

MAL is an interactive system, which allows the user to describe the sequence of steps necessary to achieve the task in a BASIC-like language. The system allows the independent programming of different tasks and provides semaphores for synchro-

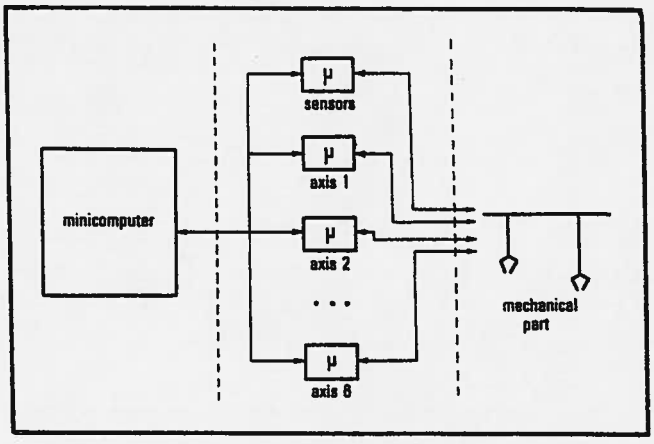


***Figure 2***
**Architecture of the Experimental Robot**

nization. The choice of BASIC is motivated, among other things, by the wide use of this language and the short turnaround time after a revision.

MAL is implemented in FORTRAN IV, except for a small interface written in assembler. The system runs on a LSI 11/02 and requires less than 20k of memory. MAL is implemented in FORTRAN IV (a new implementation in OMSI PASCAL is available also), the interface with the robot is written in assembler. Moreover, MAL is implemented in such a way that a change in the robot would not require a complete rewriting of the system. For instance, the conversion from a cartesian robot to a polar one should require modifying only a module. Likewise the system should be able to control more than one robot, possibly working in cooperation, at the same time.

MAL is composed by two different parts, one devoted to the compilation of the input language and the other to the execution of the intermediate code. The global organization of MAL is illustrated in *Figure 3*.

The compilation module provides facilities to create, update and maintain the source programs. The system is line oriented in the sense that after each line has been typed, the compiler checks for syntactic errors and eventually gives the appropriate error message.

The program may be partially executed by stopping it in correspondence with an instruction or by typing a command. The execution can then be resumed at any point.

MAL uses integer or real variables, and computes general expressions applying arithmetic oper-

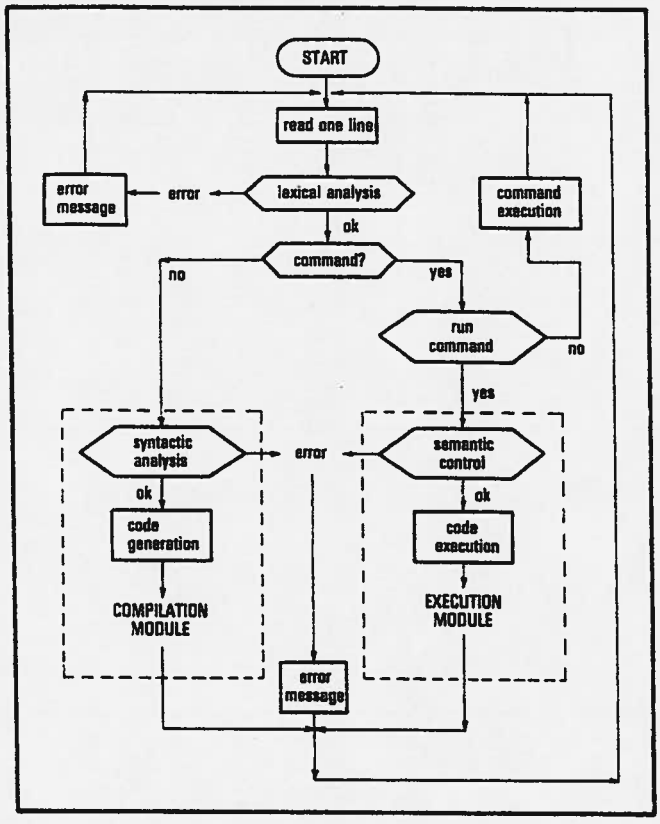


*Figure 3*
**The MAL Organization**

ators and a few defined functions. The instruction set includes DO cycles, if-then tests, goto's and subroutines calls with argument lists. Different tasks can be defined and synchronized through SET-WAIT instructions. Subroutine calls and wait instructions are handled in the same way, using a stack to push the return address.

Absolute and differential movements are available to move the six axes of the two arms. Forces can be sensed, and actuators, as the hands, can be controlled. A bell and a light can be activated to alert the user.

## Task Synchronization

The independent programming of different activities to be executed in parallel seems to be an interesting aspect of robot programming. When we want to program robots to execute parallel tasks, first we ought to isolate every logically independent activity to minimize the intersections between them.

Every activity is programmed as independent. Then the instructions needed to manage synchronization and information flow between the tasks are inserted. These instructions are tests on common variables, used as semaphores.[17] To explain the use of parallel activities we will describe a MAL program in detail.

The following example shows an application to an assembly problem, whose component parts are illustrated in *Figure 4*.

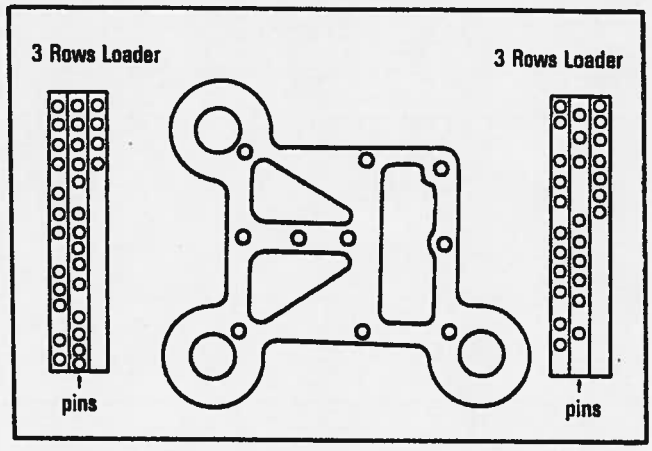


*Figure 4*
**An Assembly Task**

A mechanical part on the working plane has several holes, and a pin should be inserted into each hole. Pins are settled in six rows, three on the left and three on the right. The pins are randomly positioned within each row. Each arm of the robot sweeps the rows on its side looking for pins with its optical sensor. When a pin is detected, the position of the hole is read.

Before actually inserting the pin into the hole, the program must check whether the area over the hole is free, and must wait if the other arm is working over the hole.

This condition is expressed in the semaphore TAKEN, which is set to YES when the central area is occupied and to NO when the area is free. Task synchronization makes it possible to get maximum speed of operations, because most actions can be carried out in parallel by the two arms of the robot. The task for both the arms are the same, the only difference is the position of the three rows. The MAL program is shown in *Printout A*.

The first part of the program moves each arm to its mechanical origin. The names given to the different axes of the two arms are XL, YL, and ZL for the left arm and XR, YR, and ZR for the right

*Printout A*

```
1 SET LOBJ=13,ROBJ=12,FL=0,LH=8,RH=7,XBIT=1 "standard origin routines
2 INCR XL=1, ZL=-1, YR=1, ZR=-1       "initial positioning of axes
3 INCR W XL=-1000, YL, ZL, YR, ZR
4 SET FL=1, XL=0, XR=145, ZL=-195, ZR=-205        "axes are zeroed
5 SET YES=1, NO=0
6 MOVE W XR=720
7 SET CONTA=0, TAKEN=NO
8 TAPE NUM                                          "number of pins
9 TASK, 2,103
10 "
11 " ------------------ LEFT ARM TASK----------------------------
12 "
13 " +++outer loop:3 rows+++
14 DO K=0,2
15    MOVE W XL=60*K, YL=2
16    MOVE W ZL=3
17 " +++inner loop+++
18    DO I=1,250
19       MOVE YL=I
20       IF BIT (LOBJ)=0, GO TO 28                    "a pin found
21    NEXT I
22 "
23    MOVE W ZL=-40
24 NEXT K
25 "
26 PRINT 'few pins for left arm - I try again
27 GO TO 14
28 INCR W YL=2
29 ACT LH,LH                                        "pick up the pin
30 MOVE W ZL=-40
31 TAPE X,Y                                    "read a position for it
32 WAIT TAKEN                          "wait to access the working area
33 SET TAKEN=YES                                "working area occupied
34 MOVE W XL=X+200, YL=Y                         "put the pin in place
35 MOVE W ZL=3
36 DEACT LH,LH                                         "leave it down
37 MOVE W ZL=-40
38 MOVE XL=60*K, YL=I
39 SET TAKEN=NO                                  "working area is free
40 WAIT XL,XL
41 MOVE W ZL=3
42 SET CONTA=CONTA+1
43 IF CONTA=NUM, GO TO 77                          "all pins in place
44 GO TO 21
45 "
46 "-----------------------------RIGHT ARM TASK---------------------
47 "
48 WAIT XR
49 DO KKK=0,2
50    MOVE W XR=720-60+K,YR=2
51    MOVE W ZR=3
52    DO J=1,250
```

*Printout A continued*

```
53        MOVE JR=J
54        IF BIT(ROBJ)=0, GO TO 61                          "a pin found
55     NEXT J
56     MOVE W ZR=-40
57 NEXT KKK
58 PRINT 'few pins for right arm. I try again
59 GO TO 49
60 "
61 INCR W YR=2
62 ACT RH,RH                                              "pick up the pin
63 MOVE W ZR=-40
64 TAPE X1,Y1                                        "read a position for it
65 WAIT TAKEN                              "wait to access the working area
66 SET TAKEN=YES                                    "working area occupied
67 MOVE W XR=X1+200, YR=Y1                           "put the pin in place
68 MOVE W ZR=0
69 DEACT RH,RH                                             "leave it down
70 MOVE W ZR=-40
71 MOVE XR=720-60*KKK, YR=J
72 SET TAKEN=NO                                     "working area is free
73 WAIT XR,XR
74 MOVE W ZR=0
75 SET CONTA=CONTA+1
76 IF CONTA=NUM, GO TO 55                             "all pins in place
77 STOP                                                " end of the job
```

arm. LOBJ and ROBJ stand for the left hand and right hand optical sensors.

Movement instructions are expressed in terms of the absolute position for each joint. The "W" which appears in the instruction indicates that we want the arm to complete the movement before executing the next instruction. ACT and DEACT operate the hands.

Line 7 sets the number of pins already inserted to zero and sets the TAKEN semaphore. The number of pins to be inserted is read from Line 8 of the paper tape.

We have two parallel tasks, one to control the left arm and the other to control the right arm. Each of the two tasks is made of two nested loops; the outer one scans three rows, the inner one sweeps each row looking for pins. When a pin is detected, it is grasped and its destination is read from the paper tape. Before moving to the destination, the program waits for the working area to be free. The pin is thereafter inserted into the hole and the search is started again.

When all the required pins have been inserted, a normal exit is taken. If the pins in the three rows of an arm end before the job is completed, the search is started again. The two tasks are executed at the same time, because all actions referring to the common area are executed by each task as if they were the only task present in the system. In a program without synchronization primitives, only one arm could be moved at a time and many of the advantages of a two arm manipulator would be lost.

## Concluding Remarks

We have presented modern trends in robot programming and have proposed a classification of robot programming systems. The open problems for the next generation of programming languages for robots have been indicated. Our experience in designing, developing, and using MAL, a programming system oriented to industrial applications, has been illustrated.

## References

1. D. Nitzan and C. Rosen, "Programmable Industrial Automation", *IEEE Transactions on Computers*, Volume C-25, December, 1976, pp. 1259-1270.
2. C. Rosen and D. Nitzan, "Use of Sensors in Programmable Automation", *Computer*, 1977, pp. 12-23.
3. R. J. Popplestone, "An Interpreter for a Language for Describing Assemblies", *Artificial Intelligence*, 1980, pp. 79-107.
4. G. Gini, M. Gini, R. Gini and D. Giuse, "Introducing Software Systems in Industrial Robots", *Proceedings of the 9th* International Symposium on Industrial Robots, Washington, D.C., 1979, pp. 309-321.
5. T. O. Binford, C. R. Liu, G. Gini, M. Gini, I. Glaser et al., "Exploratory Study of Computer Integrated Assembly Systems", Progress Report 4, Stanford Artificial Intelligence Laboratory Memo AIM-285.4, Stanford, California, 1977.
6. W. Park, "Minicomputer Software Organization for Control of Industrial Robots", *Proceedings, JACC*, San Francisco, California, 1977, pp. 164-171.
7. L. I. Lieberman and M. A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly", *IBM Journal of Research and Development*, Volume 21, No. 4, 1977, pp. 321-333.
8. J. C. Latombe, "Une Analyse Structuree d'Outils de Programmation Pour la Robotique Industrielle", *Langages et Methodes de Programmation des Robots Industriels*, Institute Recherche d'Informatique et d'Automatique, (Research Institute of Data Processing and Automation), France, 1979, pp. 5-22.
9. T. Banzano and A. Buronzo, "SIGLA—Olivetti Robot Programming Language", *Langages et Methodes de Programmation des Robots Industriels*, Institute Recherche d'Informatique et d'Automatique, (Research Institute of Data Processing and Automation), France, 1979, pp. 117-224.
10. B. Shimano, "VAL: An Industrial Robot Programming and Control System", *Langages et Methodes de Programmation des Robots Industriels*, Institute Recherche d'Informatique et d'Automatique, (Research Institute of Data Processing and Automation), France, 1979, pp. 47-60.
11. G. Gini, M. Gini, E. Pagello and G. Trainito, "Distributed Robot Programming", *Proceedings of the 10th* International Symposium on Industrial Robots, Milano, Italy, 1980, pp. 61-72.
12. T. Lozano-Perez and P. H. Winston, "LAMA: A Language for Automatic Mechanical Assembly", *Proceedings of the 5th* International Joint Conference on Artificial Intelligence, Boston, Massachusetts, 1977, pp. 710-716.
13. R. Finkel, R. H. Taylor, R. Bolles, R. Paul and J. Feldman, "An Overview of AL, a Programming System for Automation", *Proceedings of the 4th* International Joint Conference on Artificial Intelligence, Tbilisi, U.S.S.R., 1975, pp. 758-765.
14. G. Gini and M. Gini, "Interactive Development of Object Handling Programs", *Computer Languages*, Volume 7, No. 1, 1982, pp. 1-10.
15. G. Gini, M. Gini and M. Somalvico, "Emergency Recovery in Intelligent Robots", *Proceedings of the 5th* International Symposium on Industrial Robots, Chicago, Illinois, 1975, pp. 339-358.
16. R. Cassinis and L. Mezzalira, "A Multimicro Processor System for the Control of an Industrial Robot", *Proceedings of the 7th International Symposium on Industrial Robots*, Tokyo, Japan, 1977, pp. 235-242.
17. E. W. Dijkstra, "Cooperating Sequential Processes", Genuys, (Editor), *Programming Languages*, Academic Press, London, 1968, p. 43.

## Author(s) Biography

Giuseppina C. Gini is currently a Senior Research Associate at the Electronics Department, School of Engineering, in the Politecnico di Milano, Milano, Italy. She received her Ph.D. in Physics from the State University in Milano in 1972. Dr. Gini's current research focuses on programming languages, artificial intelligence applications, and assembly automation. She is a member of the European Economic Community Group on robot standardization and of the committee which has defined the goals of the European Strategic Precompetitive Research in Information Technology (ESPRIT) program for the development of computer integrated manufacturing.

Maria L. Gini is an Assistant Professor at the Department of Computer Science at the University of Minnesota in Minneapolis. She received her Ph.D. in Physics from the State University in Milano in 1972. Dr. Gini has previously worked as a Research Fellow at the Electronics Department of the Politecnico di Milano, and as a visiting Research Associate at the Artificial Intelligence Laboratory at Stanford University in California. Her research interests are in the field of artificial intelligence and its applications to robotics and production control.