Standardization Issues in Robot Languages

Giuseppina GINI
Dept. of Electronics, Politecnico di Milao, Piazza Leonardo Da Vinci 32, Milan, Italy

and

Maria GINI *
Dept. of Computer Science, University of Minnesota, Minneapolis, USA

After introducing the basic characteristics of robot programming we will discuss some aspects that make it difficult to approach standardization in this field. After that, the directions taken by the IRSE (Industrial Robotics Standardization Europe) working group of the EEC will be presented.

Giuseppina C. Gini received her doctoral degree in physics from the State University in Milan, Italy, in 1972. Since then she has been research fellow at the Electronics Department of Politecnico, Milan, Italy, where she is now Senior Research Associate. She spent more than two years at the Artificial Intelligence Laboratory of Stanford University where she worked in the Hand-Eye project. She has published papers in the area of programming languages for artificial intelligence, assembly automation, robot programming and program construction.

Maria L. Gini is an Assistant Professor at the Department of Computer Science at the Minnesota University, Minneapolis. She has worked before as a Research Fellow at the Electronics Department of the Politecnico, Milan, Italy, and as a visiting research associate at the Artificial Intelligence Laboratory of the Stanford University, California. Her research interests are in the field of artificial intelligence and its applications to robotics and production control.

* Both authors are members of the Industrial Robotics Standardization Europe Programming Languages subgroup.

North-Holland
Computers & Standards 4 (1985) 13–19

1. Introduction

Today robot applications are generally carried out in an integrated industrial setting, where robots and other equipment manipulate and sense parts to repeatedly perform a task. We may see a typical setting as consisting of a conveyor belt which transports parts, usually partially oriented and separated, a vision system getting information about the incoming parts, and a robot making assembly or inspection or serving other machines.

Those robots are sometimes programmed *on line*, guiding them through their task and storing positions and operations into a memory. The drawbacks of this methods have been stressed enough [3]. Errors during programming require restarting, teaching of many positions is too long and error prone, the programming time is not productive because all the robot related equipment must be stopped during new task programming, synchronization with other equipment is hard, sensors cannot be used to modify actions during program execution.

During recent years we have seen a significant change in the attitude of robot manufacturers, and almost every new robot is today sold with an *off line* programming system, usually a so-called manipulator level language. It means that the programmer supplies the right sequence of values defining the positions where the manipulator should be moved in order to accomplish the task. Those values may be the individual values of the manipulator joints or the cartesian reference frames defining the position of the end effector. In a robot language those frames or their sequence can be modified by run-time events, as sensor outputs or external synchronization signals, to make it possible to perform the task even in case of some unpredictability in the environment.

Programming in terms of frames is not yet a good solution because it requires enough skills in mathematics and programming to supply a lot of detailed and formalized information. It is quite difficult to understand and use positions in 3D space, and this is the basis of robot programming now.

In the future an intelligent robot [5] will be given only a task and the spare parts and materials needed to perform it, instead of programs embedding the complete sequence of actions to be taken. Many research issues are to be addressed by such more advanced systems; between them, world representation and object modelling will play an important role. A representation of the world should allow the robot to manipulate parts, to sense the environment, to reason over the computer models of the real world in order, for instance, to predict collisions. Present industrial robots do not have any world model; their knowledge is encrypted into variables and data structures which only have meaning for their human programmer. An integration of modelling methods and systems used in the design and the production of products is highly desirable. Many systems are today commercially available for CAD/CAM applications; nevertheless, most of them are not of great interest for robotics because information needed for designing parts are often different from information needed to manipulate and assemble them. For instance, robot relevant information, such as the center of gravity, the mass, the grasping point, etc. of the object are not contained in CAD models.

Besides these formidable research issues industrial robots are today in use. Their performances are low with respect to the ultimate tasks of robotics, but they may be considered relevant with respect to their practical results in the manufacturing process.

The problem of robot languages standardization is as follows: today every robot has its own programming system or language, with different notations, symbols, and languages. The integration of different robots in the same line is difficult, and the training of personnel is costly.

2. Robot Languages Today

2.1. Syntax and Implementation

The most obvious ways to implement robot languages are to adapt and extend a general-purpose programming system or to develop robot-specific programming systems.

The first solution is appealing because education and training of robot programmers can be reduced if the language strongly resembles a well-known general-purpose language. Some robot systems now available are extensions of Basic or of Pascal, two of the most commonly known languages. In terms of standardization this solution is even more appealing because the robot language can be a standard extension of a standard language.

Experience has so far shown that this solution, however, has some shortcomings. Among them are the facts that Basic and Pascal do not support cooperation, which could be only obtained at the operating system level, and are poor in file management, which would occur very often in integrated manufacturing.

Other experimental implementation solutions have been writing procedures embedded into general purpose languages. This direction has shown severe limitations when the languages was Fortran, and is now developed around ADA [2], perhaps the first general-purpose language appropriate for automation [3].

The main motivation of using ADA for robot programming is that ADA is a structured and complete computer language, offering some advanced tools as extendability, modularity, real time capabilities, and strong type checking to increase the program's reliability. Moreover, it has been designed to be the only language of the 1980s and claim has been made that ADA could substitute any language in any applications.

The use of packages (a way to simplify program encapsulation) and generic packages (a way to implement abstract data types) can simplify the development of complex software, making it easier to distribute tasks to different people, to integrate them, to modify the manufacturing cell without complex software modifications. We do not know yet examples of robot systems written in ADA, while we may find examples of other automation subsystems developed in ADA. ADA stands a good chance of becoming a reasonable solution for programming robotics cells. But as yet too little experimentations has been done for ADA to be considered a good candidate for standardization. Moreover, a shortcoming often found in ADA is its philosophy of imperative and explicit programming; this is unsuitable for dealing with a complicated cell in which many events may happen, and time and sequence constraints can be met by different solutions. In this case expert systems seem a more flexible and understandable way of planning and controlling the cell.

Since languages for automation did not exist before the introduction of robots, with only the exception of APT, APT has been chosen as a basis for robot languages in a couple of projects, in West Germany and in the UK. Its use in robotics has not yet been demonstrated as truly useful, thus suggesting that it might not be a good candidate for standardization.

The second solution, that of developing new languages for robot programming, is appealing because robot programmers may not be computer experts and what they would need to learn will be exactly what they need to use to program robots. This is why most of the robot manufacturers have preferred the second solution, making standardization even harder.

2.2. Present Robot Languages

Many papers have reviewed existing robot programming languages (among them the most recent are [1], [4], [6]).

A programming system for robots usually includes:

1. A way to express movements (joint level, hand level, object level);
2. a possible expression of trajectories (linear, circular,...);
3. instructions to test or use sensors (force, contact, vision,...);
4. I/O functions, extended to manage sensors and to provide integration with other equipment;
5. possible expression of multitasking, synchronization, and parallelism.

When speaking about robot languages, please note that usually two systems are strictly integrated in a robot programming system. The *user language*, in which application programs are written, and the *run-time system* which executes the code generated by the language translator, driving all the equipment needed. This solution is similar, for instance, to the one used in most of the Pascal systems. It may be used as a way to standardize user languages by changing only the run-time system, as done by Unimation with its VAL system, developed for PUMA robots and then implemented on other Unimation robots. Often the run time system runs on microcomputers and is written in assembly languages. This trend could change when more and more cheap computer power will make it reasonable to write all the software in high-level languages.

Difficult problems in standardization come from the world robot market organization, each manufacturer offering its own language as a prestigious part of its product, as well as from the rapidly changing world of robotics. No language is now sufficiently well-assessed and few applications (mainly pick and place) are now simple and reliable enough with robots. Many new ideas have gained attention, and these are also hard to implement.

2.3. New Issues

One of these new issues is functionality. What makes a functional language attractive is its problem-oriented expression (because it expresses functions without stressing memory locations), its ability to be implemented as a very restricted kernel (the function definition and composition operators) and then to grow in every way using user defined functions, its suitability to run on largely distributed architectures because it does not produce side effects (no global memory is used). We have found an emphasis on functionality in many robot languages. On the other hand, we know that the definition of the syntax and the implementation of a functional language is a very frequently discussed and well-known problem.

Another class of languages more and more used in robot systems development is Artificial Intelligence languages, especially LISP.

As discussed above, robot programming today is a way to define the steps of the algorithm to be executed to perform a task and a way to insert the checks considered necessary to carry out the task in a real environment. These two aspects of robot programming are conceptually different and could be developed in different systems. Most of the pioneering work in creating intelligent robots has been done in the planning phase. Now, we cannot believe that a robot needs intelligence only before executing a task, and not during execution. The lack of intelligence in the monitoring system was one of the main reasons for the failure of those pioneering works.

Intelligent systems able to develop the task algorithm should reason about the geometrical and physical constraints of the parts in order to define the best operations. Intelligent systems able to execute the task algorithm should make the robot more reliable by modifying the algorithm to solve unpredictable problems.

For a few years now, research into industrial robots has continued to focus on those themes of planning and so far concentrates on the problem of automatically developing the task algorithm for high-level specifications and from world models. This is often called task-level language. In a task-level language, the user will supply a generic description of the task and the system finds the set of intermediate positions, the coordinates needed, the path to follow to avoid collisions, the sensors to use to deal with uncertainties in part positions and dimensions. The task-level languages today available in research laboratories demonstrate severe drawbacks. For instance, RAPT, one of the most advanced, does not deal with sensors and collision avoidance, and collision avoidance systems are not integrated into real robot languages.

The situation of intelligent execution systems needs even more advancements. We would have the robot be conscious of what it is doing and why it is doing so. If unexpected problems arise, the robot should find a way to cope with them, redoing failed actions, getting more information, modifying the planned sequence.

The potential applications of such intelligent systems are enormous. However, their impact on industrial environments is not expected in the near future, according to the Delphi forecast reported in Figure 2-1.

Logic programming and Artificial Intelligence programming techniques deserve further attention, being the only candidates for implementing such intelligent systems.

3. Standardization Work in Progress

The Industrial Robotics Standardization Europe Group started its work in 1982. It is headed by Prof. Rathmill (of Cranfield Institute of Technology), and sponsored by the EEC. Standardization suggestions to the standardization committees are expected of this group, as well as the dissemination of information and suggestions related to coordination between different European countries. The group is not a research group, and its activities are not aimed at developing new systems.

The only other organization involved so far in robot languages standardization is CAM-I [8]. They have distinguished five main components of robot software: robot language, robot simulator, robot controller, robot modeler, teaching system. They do not consider Artificial Intelligence problems there, and this seems a dangerous decision that may result in losing too much of the sensorial capabilities that should be added to robots to make them reliable and useful in a broad range of applications. On the other hand, no one can standardize something that is not yet fully understood and developed, and losing a function may be the price of standardization.

The IRSE group has, therefore, suggested that no standardization should be undertaken as yet on the user-level language.

The suggestions of the group is instead that of defining standard software interfaces between the robot and the user level software. Some questions to be answered are the following. What kind of information should we pass on to the robot? Joint positions, or frames? In which order? How may we ask for a point-to-point execution? For a continu-

TIME	SYSTEMS
NEXT 3 YEARS	*explicit robot languages*
1988 TO 1990	*world models and simulators*
AFTER 1990	*task level languages*

Fig. 2-1: Japanese Delphi forecast: impact on industry of new technologies–the case of robot programming languages

ous path? The answer to those questions may allow us to define a low-level language, general and extendable. This level can be considered as the minimum for driving a robot.

This interface will replace, in a standard way, what is not the run-time system input. The run-time system implementation is now heavily dependent on the robot architecture. Even so, when writing it in high-level languages becomes practical, this can be made more readable, portable and general.

One example of this direction is RCCL (Robot Control C Library). This system, developed at the Purdue University, is a set of subroutines written in C which realize all the control strategies illustrated in [7]. It is the most interesting example of a portable package for functions otherwise embedded into the assembly code implementing the run time.

If this standard software interface were provided to, tested, and accepted by any robot manufacturer, we might get any robot language to work for any robot; any user language could be translated into the standard interface. This standard interface should be provided by the manufacturers with the robot itself. Then any user could decide what user-level language to use in its plant or in its application. Any user-level language could be simply installed on any robot, provided that its compiler/interpreter is compatible with the computer available. The homogeneity and modularity so obtained would be valuable in indust ial settings.

The user languages can then grow in every way, using as target the low-level interface. Many blank slots can be left in the interface, to allow insertion of new functions if those are developed in future research.

With reference to the user level language, we strongly suggest that it be robot-independent. There are no reasons why a cartesian robot and a polar robot should be programmed in different ways. The user level language should be problem-oriented and user-oriented more than manipulator-oriented. Many of the industrial systems are too robot-dependent, even in the user-level language.

4. Conclusions

It is hard to not get confused by the many different languages for robots currently available. We have tried to illustrate briefly the main features available today in industrial robot languages. While most of the attention of robot manufacturers is now focused on defining and implementing manipulator-level systems, tailored to a specific robot, we have proposed a feasible way to start standardizing those systems. In our proposal, any development which promises to make robots capable of more ambitious tasks will not be sacrified by standardization.

Acknowledgements

Thank are due to the members of the Programming Languages subgroup of IRSE for helpful comments, and particular to, J. Bach, R. Dillman, and J. Lefebre.

References

[1] Bonner, S. and Shin, K.G., A Comparative Study of Robot Language, IEEE Computer, N. 12, pp 82–96 (1982)

[2] DoD, Reference Manual for the ADA Programming Language. Proposed Standard Document, Dept. of Defense, USA (1980).

[3] Gini, G. and Gini, M., ADA: a Language for Robot Programming?, Computers in Industry, Vol. 3, N. 4, pp 253–259 (1982).

[4] Gini, G. and Gini, M., Robot Languages in the Eighties, Proc. of the Robotics Europe Conference, Bruxelles, Belgium, 1984.

[5] Kempf, K., Artificial Intelligence Applications in Robotics – A Tutorial, IJCAI 83 Tutorial, Karlsruhe, Germany (1983).

[6] Lozano-Perez, R., Robot Programming, Proc. of the IEEE, 17, 7 (1983).

[7] Paul, R., Robot Mathematics, Programming, and Control, MIT Press, Cambridge, Mass (1982).

[8] CAM-I Proposes Standards in Robot Software, The Industrial Robot, pp 252–253 (1982).

Appendix: Annotated Basic Bibliography of Industrial Robot Languages

AL: Assembly Language. Designed in 1974 at the Stanford Artificial Intelligence Laboratory as a frame-oriented language with openings to artificial intelligence concepts, it is now considered the best language available for controlling and programming robots. Its basic concepts have influenced LM and SRL. The language is used to program a set of four robots of different types and a 2D-vision system.

Finkel, R. et al, An Overview of AL, a Programming System for Automation, Proc. 4th IJCAI, Tbilisi, USSR (1975).

It has a programming environment, POINTY, to speed up the writing and testing of programs.

Gini, G. and Gini, M., Interactive Development of Object Handling Programs, Computer Languages, Vol. 7, N. 1 (1982).

AML: A Manipulation Language. The language developed at IBM for various robots. It has a structured PL/I like syntax. Functionality enables AML to use any new procedure as part of the language. Often considered a language difficult to use, it has been designed by IBM as a modular system in such a way that clerks at different levels use only different subparts of the system.

Taylor, R.H., Summers, P.D., and Meyer, J.M., AML: A Manufacturing Language, The International Journal of Robotics Research, 1, 3, pp 19–41 (1982).

HELP. Developed by DEA (Italy) for their Pragma A 3000 (Allegro, in USA) assembly robot. The syntax is Pascal-like: all the manipulation functions are provided as subroutines. Signal and wait provide synchronization between different tasks. Sensors can be connected using a rich set of I/O ports operations. The robot is modular; different arms and different degrees of freedom for each arm can be organized. Major applications are in the automotive industry, electronic assembly, precision mechanics. It is implemented on DEC LSI 11 computers under the RT-11 operating system.

Camera, A. and Migliardi, G.F., Integrating Parts Inspection and Functional Control During Automatic Assembly, Assembly Automation, 1, 2, pp 78–82 (1981).

LM: Language Manipulation. A language developed at the University of Grenoble (France). It has been implemented on different robots, with 6 degrees of freedom. It is Pascal-like and frame-oriented and provides many of the features of AL, with the exception of coordination and parallel execution of tasks.

Latombe, J.C. and Mazer, E., LM: a High-level Programming Language for Controlling Assembly Robots, Proc. 11th ISIR, Tokyo, Japan, pp 683–690 (1981).

It is integrated with LM-Geo, a system used to infer bodies positions from geometrical relations and to write them as LM declarations and instructions.

Mazer, E., Geometric Programming of Assembly Robots, Proc. Advanced Software in Robotics, Liege, Belgium (1983).

PASRO: PAScal for RObots. It is provided by the German company Biomatik. It is a library of data types and procedures used to perform robot specific tasks, and it is callable by any standard Pascal compiler. It is based on the AL experience.

Biomatik, PASRO-Pascal for Robots, Biomatik Co., Freiburg, West Germany (1983).

RAPT: Robot APT. In its actual implementation RAPT is an APT-like language used to describe assemblies in terms of geometric relations and to transform them into VAL programs. A RAPT program consists of a description of the parts involved, the robot, and the workstation, followed by an assembly plan. The assembly plan is a list of geometric relations expressing what geometrical relations should held after a step in the assembly has been done. The program is completely independent of the type of the robot used. Sensors are not integrated, and movements are not checked against collision avoidance. The Computer vision CADD3 system has been used to build the models.

Popplestone, R.J. et al, An Interpreter for a Language for Describing Assemblies, Artificial Intelligence, Vol. 14, pp. 79–107 (1980).

ROBEX: ROBoter EXapt. The off-line programming system developed at Aachen (Germany) as a programming tool for FMS. Its main purposes are to develop APT for FMS and for robot off-line programming, independently of the kind of robot. Applications are in workpieces handling. APT style of programming is used to describe geometry, while the ROBEX extensions are robot movement instructions, interactions with sensor (now only binary ones), and synchronization with peripherals (as machine tools, or conveyor belts). The system generates robot independent pseudo-code which is sent to the appropriate robot for further processing and execution.

Weck, M. and Eversheim, E., ROBEX-An Offline Programming System for Industrial Robots, Proc. 11th ISIR, Tokyo, Japan, pp 655–662 (1981).

SIGLA: SIGma LAnguage. The language for programming Olivetti SIGMA robots and the first commercially available. Now quite obsolete and under replacement, SIGLA includes: a supervisor, which interprets a job control language, a teaching module which allows teaching-by-guiding features, an execution module, editing and saving of program and data. Applications range from assembly to riveting, drilling, milling. All the system and the application programs run in 4K of memory; this compactness was necessary at the time SIGMA was delivered because memory was still expensive.

Salmon, M., SIGLA-The Olivetti SIGMA Robot Programming Language, Proc. 8th ISIR, Stuttgart, Germany, pp 358–363 (1978).

SRL: Structured Robot Language. The language under development at the University of Karlsruhe. It is a successor of Portable AL and is based to some degree on Pascal, too. The declaration part contains declaration of data types and a specification of the system components. Instructions can be executed sequentially, in parallel, or in a cyclic or delayed way. Different motions are available, in particular straight and circular motions. The source SRL code is translated into an intermediate code, IRDATA, which is a machine independent code.

Blume, C. and Jacob, W., Design of a Structured Robot Language (SRL), Proc. Advanced Software in Robotics, Liege, Belgium (1983).

VAL: Vicarm Assembly Language. This language was originally developed for a small, articulated arm no longer produced. It was then extended to all the robots of Unimation. VAL, and now the second release VAL-2, is a programming system with a low-structured but user-friendly language. It allows mixing on-line and off-line programming, and integration of a vision system as well as other sensors.

Schimano, B., VAL: an Industrial Robot Programming and Control System, Proc. Programming Languages and Methods for industrial robots, IRIA, France (1979).

VML: Virtual Machine Language. The language developed in cooperation by the Milan Polytechnic and the CNR Ladseb of Padova (Italy). Intended as an intermediate language between Artificial Intelligence systems and robots, it receives points in the cartesian space and transforms then into joint space. It manages task definition and synchronization as well. It is an example of the functionality to be embedded into a standard interface.

Gini, G. et al, Distributed Robot Programming, Proc. 10th ISIR, Milan, Italy (1980).