W.E. van Waning

Engineering robot actions in a computer integrated manufacturing environment

Department of Computer Science Report CS-R8415 October

ENGINEERING ROBOT ACTIONS IN A COMPUTER INTEGRATED MANUFACTURING ENVIRONMENT

W.E. van WANING

Centre for Mathematics and Computer Science, Amsterdam

A model is presented according to which robot-actions and the activities in manufacturing cells can be designed. In this model, design-activities have three major aspects: specification, analysis and synthesis. Principles are then derived for the construction of programming systems for designing operations of robots and manufacturing cells. The specification describes an external environment (the device to make and the tools to make it with). Given the outer environment and the knowledge specific for the discipline the engineer designs possible inner structures that serve as strategies specifying how to make that device. It is important that the engineer can express the designs symbolically. Finally, when synthesizing the process-structure the designed manufacturing process is matched against the external environment. The need is stressed for simulation environments so that it is possible to test the design thoroughly on the basis of actually observed sensor-data before the programs are taken into production.

1980 MATHEMATICS SUBJECTS CLASSIFICATION: 86B20, 68G99, 68J10.

1984 CR CATEGORIES: D.2.6, I.2.9, J.1.6. 69D26, 69K21, 69L10

KEY WORDS & PHRASES: computer integrated manufacturing (CIM), robotics, engineering design, design and organization of systems, programming environments.

NOTE: This report will be submitted for publication elsewhere.

Report CS-R8415

Centre for Mathematics and Computer Science

P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1. INTRODUCTION.

The purpose of engineering is the design and construction of a specified object. The objects to be designed when we have to do with robots are processes that transform raw material into a device that conforms to an initially given design. The transformations applied to the raw material may be operations such as gripping, spraying, cutting, welding and in the future more involved machining operations.

An engineer analyzes specifications the results of which include:

- a logical structure of (sub-)processes such that the desired transformations are effectuated;
- (sub-)specifications of the functions to be realized and constraints to be satisfied by those sub-processes.

Whether or not a design plan meets its set of constraints will have to be verified. The identified processes must be synthesized so that the specification is satisfied. Establishing the acceptability of a plan with respect to a specification can often be done by computational means: mathematical and physical models of materials, robots and manufacturing cells make it possible to capture essential aspects of a plan and to compute the parameters necessary to judge the acceptability of that plan.

The purpose of engineering manufacturing processes is to transform a specification of a device into a specification that also describes the way it is to be built. Like any problem-solving activity, the activity of engineering can be subdivided into three aspects:

Specification - a (more or less) detailed description of a device to be constructed, workmanship needed and constraints to be imposed on materials - needed to be undertaken by an engineer, architect, &c.

Analysis - the resolution of the device (process) into specifications of simpler elements, the determination of the logical and physical properties of the device (process) and the study of its functionality.

Synthesis - the building up of the simple elements found by analysis into an integrated whole intended to conform to the specification.

The analytic aspect of design work brings a recursive flavour to engineering: a structure is designed in which simple elements (sub-processes in this case) are related to each other. These sub-processes in their turn need to be specified, analyzed and synthesized. The process of engineering design can thus be likened to the traversal of a tree: starting at the root a structure is designed that relates sub-processes. If these sub-processes have not been designed yet, they must in their turn be designed by specification, analysis and synthesis and so the design-tree grows deeper and deeper. The design-process stops when parts are identified that have already been made or designed. Indeed, this aspect of the design process permeates engineering: the recursive flavour of engineering is even reflected by the hierarchy we perceive when trying to understand a complex phenomenon.

It is wise to keep the depth of the tree limited, so that the effort made for the design stays within reasonable bounds. Therefore, the development of libraries of standard actions appears to be desirable when designing manufacturing processes. The benefit of these "action"-libraries would be comparable to the benefits of using of subroutine-libraries for numerical computation or those of using (more or less) standard "high-level" components in mechanical engineering. The availability of standard parts as a basic means to reduce the depth of a design-tree becomes all the more apparent when designing more than one process: the sooner one can identify common components, the less needs to be developed in all of those designs.

In the words of Simon[7] a device can be regarded as an interface between an outer and an inner environment. The advantage of dividing outer from inner environment

"... is that we can often predict behaviour from knowledge of the system's goals and its outer environment, with only minimal assumptions about the inner environment. An instant corollary is that we can often find quite different inner environments accomplishing identical or similar goals in identical or similar outer environments."[7]

Often there is a corresponding advantage in the division from the standpoint of the inner environment:

"In very many cases, whether a particular system will achieve a particular goal or adaptation depends on only a few characteristics of the outer environment, and not at all on the detail of that environment. It is an important property of most good designs, whether biological or artefactual. In one way or the other, the designer insulates the inner system from the environment, so that an invariant relation is maintained between inner system and goal, independent of variations over a wide range in most parameters that characterize the outer environment."[7]

A specification - whatever its level of formality - can be stated in terms of the functions the device should have in the outer environment. Analysis is concerned with the design of an inner environment: possible anatomies of the proposed device are produced. These anatomies contain structural specifications of the device and based on these structures (understood as external environments) functional specifications of the identified parts can be given. The purpose of synthesis, then, is to verify whether the design as conceived by analytic means will function satisfactorily in the external environment. A functional specification shows *what* is needed in terms of an outer environment; a structural specification points out *how* the qualities mentioned in the functional specification can be attained in terms of an inner environment.

2. SPECIFICATION.

If we regard a robot-process as an interface between an inner and an outer environment then a specification should be stated in terms of functions to be performed in an external environment. A modern engineer does not usually design a complete product alone: the design of present-day artefacts is a corporate effort where different specialists cooperate in the design of all aspects of the product. Especially industrial engineering has too many multidisciplinary aspects to be done by one person: a contemporary product does not only have its broad technological aspects, there are also important considerations playing their role in the design that stem from for instance economic areas.

The primary purpose of a specification is to make certain that everyone involved in the project speaks about the same things. This can only be achieved if the specification is indeed stated in functional terms that refer to an external environment.

What does an external environment look like for a robot-programmer in a computer integrated manufacturing environment? First, there is the specification of the part to be manufactured. In a computer integrated manufacturing environment we can expect that this specification comes from a system for computer aided design (CAD). One major concern in the construction of CAD-systems is to have representations of objects that are always guaranteed to be possible in the physical world - although Escher's art is fascinating, many of his best-known works are not very interesting from a purely industrial point of view. Adequate models for representing solid objects exist: Requicha[6] describes representation schemes that guarantee that no impossible objects are ever generated. Kimura *et al.*[3] extend the use of this type of representation to manufacturing activities.

A second important component of the external world for a robot-programmer is the representation of the robot-machinery present in the factory. Just as we need valid descriptions of the objects to be manufactured, so do we need valid descriptions of the tools we use, including the description of the activities we can model with them. The same requirements apply to the representation of tools and operations: the representation of those objects must be valid for the physical world.

The residual component of the external world is the actual assignment to the designer of the manufacturing processes. Usually, the task is assumed to be understood: "Make such and such an object, given the tools we have". Although we can expect that three aspects of the problem are clearly defined - the device to be made, the raw material to make it from and the tools to make it with - these aspects do not say anything about *how* we are to proceed. At best, they imply constraints on the methods to be considered.

If the problem we have to solve is not trivial, or if we have a "creative" attitude, there will probably be no acceptable cookbook-solution for it on our book-shelves or in our memories. The type of problem is completely different from for example the type of problems we had to solve in algebra and geometry in school. Those problems all had a definite criterion against which to check the correctness of the solution we gave. In some cases checking a solution simply came down to comparing two numbers. The methods we had at our disposal were fairly limited: many problems were specifically designed to apply a method that had just been studied and if this method was not correctly used, in the ideal case the answer was either incorrect or we could not find an answer at all. Even the reasoning processes we could use were limited. In short, school problems are well-structured problems. The design aspects of engineering problems usually lack these characteristics and can therefore be called ill-structured problems.[8] Most important, perhaps, is the lack of a definite and machine-checkable criterion for accepting or rejecting a design. But then, most real-world problems are ill-structured and accepting or rejecting a solution to them is essentially an activity for which humans are responsible.

3. ANALYSIS.

Having an understanding of parts of the outer environment and some understanding of what the actual task is, the engineer first "debugs" the outer environment on the basis of his specific expertise: parts or devices that are impossible or very hard to make with the given equipment must be redesigned or else new tools must be purchased, if the piece must be made in-house. In both cases the outer environment is changed. If the outer environment appears to present no immediate problems the engineer designs possible inner environments, possible anatomies of the sequence of actions that have the desired effects. The basic question here is: how does it work?

In answering this question, the engineer gradually transforms the vague and ill-structured parts of the specification into more and more specific versions. Note that this process of transformation, the design-activity *per se*, is itself highly vague and informal: it is impossible to convert the informal to the formal by formal means! If computer-systems are to be of any assistance in this process, these systems must enable the engineer to do rapid prototyping. Rapid prototyping enables us to get a quick, intuitive grasp of a design and change it, if necessary, thus supporting the transformation-process.

The key to a design is the structural model of a device. Disciplines of engineering are based on the premiss that essential properties of a designed device will hold when that device is realized. This predictive quality of engineering is based on the use and development of abstract models specific for a particular engineering domain. Hence it is natural to base an engineering analysis system on the abstract models used in that particular domain. An engineering system so constructed will not only have the *representations* of the designed products available in terms of the abstract concepts the user is familiar with, but also the *communication* between the designer and his design-system can be handled in terms of these domain-specific models. At the same time, a user does not have to learn to translate his domain-oriented concepts into computer-oriented concepts and vice-versa which is one of the major bottlenecks in the use of computers in the application to real-world problems.

The structural part of a design describes, abstracted from the parts it has, how the required characteristics of the device are obtained. It describes the inner environment and from this description the correct or incorrect functioning with respect to the outer environment of the device must be inferred. In this sense an artifact is truly an interface between an inner and an outer environment. Note as well, that the identified parts are a level lower in abstraction than the device being designed at the current level: it is in this way that the design-tree is traversed. The designer, working in a node of the tree, can concentrate on the logical and structural parts of his design and can regard the parts of it simply in terms of the function they have in his structure, irrespective of the internal complexity of those parts.

In relation to our structure/process dichotomy, Pratt[5] considers the competence/performance dichotomy in programming and Kowalski[4] coins his equation "Algorithm = Logic + Control". The structure of a manufacturing process determines its correctness, while the processes - being related to each other by means of this structure - affect the efficiency with which the manufacturing process is executed. Examples of computational formalisms that have a clear separation between these two components are pure LISP, pure PROLOG, relational algebra for databases, parts of APL, &c. In pure LISP programs can be expressed by functional composition of other programs or by the use of recursion equations. The execution of the programs so constructed is left to the LISP-system. The formalism of pure LISP does not allow us to express anything related to the efficiency with which the programs are executed: every program has to be expressed in terms of compositions or recursion equations based on other programs. Similar observations hold for memory-usage: we cannot indicate how many cells a list will contain or how long a certain value will be needed: a component of the processing-system, the garbage collector, takes care of these issues.

The advantage of such formalisms is that once we are familiar with them we can concentrate on the logical *structure* of the solution without concern for matters irrelevant to the problem at this level of abstraction: we simply *cannot* express matters pertaining to runtime-efficiency. This is not to say that becoming familiar with the formalism or programming in the formalism is easy; what it does say is that because one is able to express the logical structure of one's design, abstracted from aspects pertaining to efficiency, the number of errors in a program are usually smaller and have a different, less trivial nature. As we cannot make errors that arise from confounding the logical structure of a program with efficiency-concerns, the errors we make are not merely bugs but they point to some real difficulty in the solution of a problem.

These kinds of computational formalisms make it possible to do rapid prototyping, which gives quick feedback to the user about the behaviour of his (prototype-)system. Using rapid prototyping, we can quickly test several alternative solutions and learn from them so that the solutions can be refined and improved; as indicated before, rapid prototyping can be of considerable assistance when we are working on ill-structured problems. These activities are typical for problem-analysis: step by step we improve our understanding of the problem, while at the same time having more and more accurate solutions.

One of the differences between mathematics and engineering is that in mathematics many worlds with their own laws and abstractions are studied, while engineering is concerned with only one world - the real one. The products designed with engineering formalisms must be realizable in the physical world and if we are designing manufacturing processes, those actions must be representable in the world as well, just as the device produced by means of a CAD-system had to be realizable. Many actions we design are geometrical in nature, and can in principle be represented by the same schemes that are used in some CAD-systems.[6,3] It is therefore natural to consider means of specifying movements graphically (*e.g.* assembly-movements). It would for instance be possible to indicate a movement by specifying a line or curve along which the movement is to take place; the system would take care of representing this movement in coordinates. Moreover, it becomes possible to combine two or more movements, to shrink or enlarge them, speed them up, &c. Libraries of movements and operations on these movements could be defined and used in this way if the representation of those movements are guaranteed to be realizable in the physical world.

There are other aspects in the design of manufacturing processes that are typical for the field. Many actions are done concurrently and there is feedback from sensors that affects the operation of the system. Sensors themselves operate concurrently with the processes designed for the manufacturing task. From a programmer's point of view, feedback is not *very* familiar. In case of an emergency, to give an example, feedback can lead to "attention-shifts" in a system.

In the approach we have been advocating, concurrency and feedback have to be modelled. Concurrency and feedback can be modelled if we view the world in terms of relatively independent objects that send messages to each other.[2] These messages can either be commands ("Go to work!") or sensor-signals ("My temperature is too high!"). Emergency messages are messages with high real-time requirements. The realization of such a network of independently acting objects is purely a matter of control. If we build a network the capacity of which is high enough to guarantee that there will be no queue of messages at any node of the network and if the distance between the node emitting the emergency and the node responsible for its processing is small enough, then the real-time requirements may be satisfiable. If, on the other hand, message-queuing occurs at the nodes of the network, then there must be means to give priorities to messages; the usual buffering mechanisms will have to be transformed into, for instance, priority queues.

A number of formalisms have been developed for programming the tasks of a robot or those of a manufacturing cell. One approach is to associate an action with a state the robot or the cell is in. When a state-description matches the actual state, its associated action is triggered, so that the machinery is in a new state when the action is finished. The action serves as a state-transformer and if a state-action pair is present for the new state, the corresponding action fires again to obtain the next one.[1] Sensor-data give information about the state the machinery is in, so sensors can be accommodated for in this model. Another attractive aspect is the non-procedural character of this formalism: it does not require us to specify the order in which actions are to take place. On the other hand, it is not clear how emergency situations could be handled with this model.

Another approach that, to my knowledge, has not been reported in the literature, is to view the flow of information through a robot or through a manufacturing cell as if it were a real data-flow network.[9] If this is an adequate metaphor, then functional languages would become practical in this area, which would enable us to abstract from the states that the operations induce on the machines. Abstraction from the states of machines while allowing emergency-messages appears to be a necessary step in the direction of machine-independent robot- or manufacturing cell programs, although such a model is likely to make its processing-issues more complicated.

4. SYNTHESIS.

When the internal structure of a process to manufacture a given device with given tools has been designed, it remains to verify whether this internal structure is really compatible with the external environment in which it is to operate. The design, describing an inner environment in structural terms, needs to be checked against the functional specification stated in terms referring to the initially given outer environment. In fact, we are debugging the internal structure of the process just designed.

Simulation is a useful tool in studying the behaviour of complex systems when we know how a system works. Parts of systems may be understood well enough, but their combinations may easily become very complicated, as exemplified by devices developed in electrical engineering or programming. As we know how different components are combined to form a structure, we might as well use this information augmented by the models of their parts and simulate the operation the complex device. Studying unforeseen interactions in programmed operations is a necessary part of the work of a robot-programmer: programming robots is a complex task and we cannot foresee all consequences of decisions that have been taken in earlier or different stages of a project.

Debugging traditional programs can be a time-consuming activity. Debugging manufacturing processes does not appear to be less time-consuming, to say the least. What *is* different from traditional programming however, is that the programs are designed to be run in a factory, which is an environment that differs from the environment in which the programs have been developed. If there is no environment in which we can get a serious impression of how these programs operate, then very complex and very expensive machinery would not only be needed for production purposes, but also for testing purposes, which makes them even more expensive and more vulnerable to serious damage if our process-designs have the right kind of errors. So, in order to keep the manufacturing equipment as dedicated as possible to production-tasks, a separation suggests itself between process-*development* and process-*execution*. In particular, if the process-development system contains a sub-system for simulation of the designed processes, then an additional advantage of such a system is that we can get an impression of how the system operates under controlled circumstances: on the basis of sensor-signals actually observed in the factory one can make statistical models of the behaviour of those sensors. In this way realistic situations in the factory may be closely approximated. Moreover, statistical models do not only give us an idea of what the real world may be like, they also give the opportunity to differentiate between normal and extreme situations. Obviously, it is highly desirable to be able to assess the behaviour of our designs under circumstances that are either hard to create in the factory or potentially very expensive due to the damage that may result from the behaviour of our designs under extreme, but nevertheless important situations.

If we apply the same type of separation to robot-programming and robot-control, similar advantages can be obtained. After a manufacturing process has been developed it can be down-loaded into the robot-controller. This can result in considerable savings in the complexity of the computing machinery that controls the robot. Although the robot may have been programmed in a formalism that is very dissimilar from the commands that actually control the robot, a full translation and optimization phase in the sense of traditional compilation can take place. If such a translation could not be exploited fully, then the robot-controller would need to have a high-level interpreter and run-time system for those movements, which would necessitate more complex and faster computers to control the robot.

The separation between simulation and control-issues does not only help in making systems for robot-control less expensive, at the same time it will facilitate the development of manufacturing programs, as the simulation environment can be integrated with the program development system. A program development system with integrated simulation and debugging facilities will form an excellent programming environment. Separating these two environments will also make it possible to write programs that are entirely independent of the particular robot that happens to be in the factory: a type of device-independence of robot-programs does not appear to be merely desirable, but mandatory in view of the fact that more and more robots will be used in our factories. If those robots can be addressed exclusively by their specific and non-portable programming formalisms, then the cost of reprogramming all of these machines for every new operation will soon outweigh their potential benefits. On the other hand, separating simulation and control-issues leads, in terms of the structure/process dichotomy, to two types of control-system, one system in the factory that manages the operations of the machinery itself, the other system, part of the program development environment which serves as the control-system for the simulations of the designer's ideas.

References

1. Bourne, David A., "A numberless, tensed language for action oriented tasks.," CMU-RI-TR-82-12, The robotics institute, Carnegie Mellon University., Pittsburgh, Pennsylvania, 15213, USA. (Oct. 1982).

2. Hewitt, C., "Viewing control structures as patterns of passing messages.," *Artificial Intelligence* **8** pp. 323-364 (1977).

3. Kimura, F., Sata, T., and Hosaka, M., "Integration of design and manufacturing activities based on object modelling.," in *Advances in CAD/CAM*, ed. O.I. Semenkov, North-Holland Publishing Company., Amsterdam (1983).

4. Kowalski, R., "Algorithm = Logic + Control," *Communications of the ACM* **22**(7) pp. 424-436 (July 1979).

5. Pratt, V.R., "The competence/performance dichotomy in programming," pp. 194-200 in *Proc 4th ACM Sigact/Sigplan symp. on Principles of Programming Languages, Santa Monica, Calif.*, ACM (January 1977).

6. Requicha, A.A.G., "Representations for rigid solids: theory, methods and systems," *Computing Surveys* **12**(7) pp. 437-464 (December 1980).

7. Simon, H.A., *The sciences of the artificial,* The M.I.T. Press, Cambridge, Mass., U.S.A. (1969).

8. Simon, H.A., "The structure of ill-structured problems," *Artificial Intelligence* **4** pp. 181-201 (1973).

9. Treleaven, Philip C., Brownbridge, David R., and Hopkins, Richard P., "Data-driven and demand-driven computer architectures.," *Computing Surveys* **14** pp. 93-143 (March 1982).