AUTOMATIC RESOURCE ALLOCATION IN INDUSTRIAL MULTIROBOT SYSTEMS.

Riccardo Cassinis

Dipartimento di Elettronica - Politecnico di Milano
I-20133 Milano, Italy - Piazza Leonardo da Vinci, 32 - Tel. I-2-2367241
Tlx. 333467POLIMI I

ABSTRACT

The paper proposes a methodology for robot programming, based on the concept of automatically assigning available physical resources to the tasks to be performed during program execution, rather than specifying them in the user program.

This is specially useful in multirobot systems, when several similar machines are included in the same robotized cell, and when the system has to cope with unpredictable length operations.

The most interesting feature of the system appears to be its capability of maintaining good performance in those situations where, due to faulty robots, to sudden changes in the production cycle, etc., traditional scheduling and optimization techniques fail to succeed.

The paper includes a discussion of the main advantages and drawbacks of the technique, together with some experimental results.

1. - INTRODUCTION.

Industrial robots have so far been programmed with several different methods, among which the most interesting is undoubtedly the use of a suitable language. The actual research trend is towards *implicit* programming techniques, as opposed to *explicit* languages that have been used up to now.

Although fully implicit programming is still quite far from being feasible, some intermediate solutions can be found, that can speed up the programming procedure and increase the system efficiency.

The paper deals with the problem, that is usually encountered in multirobot systems, of deciding which machine has do perform each elementary task. As it will be shown in the sequel, the choice should be done at run-time in order to ensure the best performance of the system. The aim of the paper is to show how this match can be done, by introducing the idea that robots should be programmed in terms of *virtual* resources rather than *physical* ones, and that the mapping between virtual and physical resources should only be done at run-time.

This methodology is specially useful when several robots with similar capabilities are included in the system, or when unpredictable length operations must be performed, because it allows a better use of available resources even in dynamically varying environments.

This research is still in the implementation phase, and no final results are available yet. Some practical experiments have already been done, and their results are described in section 7.

2. - EXPLICIT VERSUS AUTOMATIC RESOURCE ASSIGNMENT.

Most of the existing robot programming languages were designed for driving a single robot. It is therefore obvious that, when writing a program, the operations being described by the programmer will be executed from the arm that is being programmed.

In an ever increasing number of cases, however, these languages are to be used to program multi-arm robots or multirobot systems. In this instance, the programmer must always state, for each physical operation to be performed, which arm will have to do it.

Although this procedure is the most logical one (when programming with an explicit language, one must have in mind the complete sequence of operations to be performed), it has several drawbacks, the most conspicuous being that, since the operations to be done can have unpredictable length, the time schedule cannot be completely defined, and this can lead to severe inefficiencies.

It must also be noted that implicit programming does not imply automatic resource assignment, nor does explicit programming require explicit resource assignment: the two concepts are orthogonal, since implicit programming only implies a synthetic description of the operations to be performed, but the resources to be used to perform any task can be explicitly defined.

Automatic resource assignment on the other hand implies that the program does not contain any explicit reference to the resource that must perform each task, but only a description of the kind of resource to be used to perform it. The system will then assign the available resources to the tasks to be executed, using an opportunistic scheduling technique.

It is extremely important to note that an optimal assignment can only be done at run time, because, as it was said before, the exact process time schedule cannot be foreseen [1]. The same approach can however be used also during off-line program generation, as an aid to the correct resource distribution problem.

The problem being investigated is quite similar to the one described in [2], although the main issue is in this case the optimization of the operation of several robots, rather than the optimization of the sequence of operations of a single robot [3].

3. - THE *ACTION POINT* CONCEPT.

In order to make the discussion clearer, it will be useful to introduce a concept that summarizes what was said in the preceding paragraph.

When programming a system with automatic resource assignment capabilities, the user must define the available resources, and write the program in terms of abstract resources that will be mapped with the available ones only at run time.

An **Action Point** can be defined as a resource, that can perform physical operations over the world surrounding the robot [4].

It is identified by several parameters, among which one may list:

- **Kind**: gripper, screwdriver, conveyor, etc.;
- **Physical parameters**: number of fingers, fingers opening, size of screwdriver, etc.;
- **Capabilities**: PTP or continuous path movement, etc.;
- **Current position**: coordinates of the end effector;
- **Physical status**: mounted, not mounted, faulty, borrowed by another robot, etc.;
- **Logical status**: free, busy;
- **Geometric description**: for trajectory planning and coordinate conversion.

This list of parameters can be given in the form of appropriate tables or data structures.

In the sequel, references will be made to a program that is being developed to demonstrate the feasibility of the system. The program will first be run in a simulated environment, and then applied to the multirobot system available at our Robotics Laboratory.

3.1. - Virtual action points.

Virtual Action Points (VAP's) are the action points the programmer refers to when describing the task the robot must accomplish.

One or more VAP descriptions must be included at the beginning of each task to be executed. Each description must contain all necessary information to allow the scheduler to decide which one of the available tools should be used for performing the task.

As an example, the VAP description that is used in the program being developed is the following:

```
type
  Ident = integer;
  ActionPointStatus = (APFree, APBusy, APNotMounted, APFaulty);
  EffectorStatus = (EFFree, EFBusy, EFFaulty);
  MountingFlangeType = (Fixed, Flange1, Flange2);
  Frame = record
      X, Y, Z, Phi1, Phi2, Phi3 : real;
    end;
  Cube = record
      UpperLeft : Frame;
      LowerRight : Frame;
    end;
  ActionPointKind = (Hand, Screwdriver, ConveyorBelt,
                          SuctionCup);
  VAP = record
      IdentNumber : Ident;
      SymbolicName : string[30];
      DOF : 1..6; {Degrees of freedom}
      CP : boolean; {Continuous path capability}
      Reach : Cube; {Required operating area}
      case Kind : ActionPointKind of
        Hand : (
          FingersNumber : 2..3;
          ProporOpening : boolean; {Proportional opening capability}
          MaxOpening : real;
          MinOpening : real;
        );
        Screwdriver : (
          ToolType : (Hex, Nut, Cross, Blade);
          ToolSize : real;
          ScTolerancePlus : real; {These tolerances are used if the
                                   exact screwdriver is not available}
          ScToleranceMinus : real;
        );
        ConveyorBelt : (
        );
```

```
     SuctionCup : (
        Diameter : real;
        SuToleranceMinus : real;
        SuToleranceMinus : real;
     );
  end;
```

3.2. - Real Action Points.

Once the system starts executing the program, a mapping must be done between VAP's and the tools that will actually execute the job, and that will hereafter be referred to as Real Action Points (RAP's).

The mapping must be done by a scheduler that receives requests from the tasks that can be activated, compares them with the available resources and assigns the best fitting resource(s) to each task.

RAP's description is similar to VAP's description, although some parameters are different, and some additional ones are necessary for allowing the scheduler's work.

The RAP's description used in the aforementioned program is:

```
RAP = record
   IdentNumber : Ident;
   SymbolicName : string[30];
   MountingFlange : MountingFlangeType;
   Displacement : Frame;
   RestingPosition : Frame;
   Status : ActionPointStatus;
   BearingEffector : Ident;
   case Kind : ActionPointKind of
     Hand : (
        FingersNumber : 2..3;
        ProporOpening : boolean;
        MaxOpening : real;
        MinOpening : real;
     );
     Screwdriver : (
        ToolType : (Hex, Nut, Cross, Blade);
        ToolSize : real;
     );
     ConveyorBelt : (
     );
     SuctionCup : (
        Diameter : real
     );
  end;
```

3.3. - Effectors.

Since many robots use interchangeable tools, RAP's are not uniquely assigned to any arm. A further concept has to be introduced: the **effector**, that represents any device that can carry a RAP: typically, any robot arm. This is the effector description that is being used:

```
Effector = record
  IdentNumber : Ident;
  SymbolicName : string[30];
  DOF : 1..6;
  CP : boolean;
  Reach : Cube;
  Status : EffectorStatus;
  MountingFlange : MountingFlangetype;
  BearingRAP : Ident;
  ActualCoordinates : Frame;
 end;
```

All the definitions given above are preliminary, and will have to be amended and expanded for any practical use.

4. - THE SCHEDULER.

The job of the scheduler is quite complex, but it must be noted that it is composed of several tasks which appear to be somewhat independent from each other.

There are at least three points that deserve a great attention when designing the scheduler.

First, since the user does not know which RAP will actually execute any part of the program, trajectories cannot be specified by the user. This requires that at least a gross motion planning mechanism should be present in the system.

Second, it may very often happen that a RAP is available to execute a task, but the parts to be operated upon are not within its reach limits. In this case, it is necessary to displace the parts to be handled by some means (for instance, conveyor belts).

Third, the choice of a RAP may be difficult if more than one RAP is available that can execute a particular task. This involves some algorithms to evaluate which RAP could execute the task in the best possible way, taking into account several factors, such as execution time, accuracy, etc.

At the present moment, no technology is available which could allow the construction of a general scheduler capable of dealing with any possible situation, but a great amount of research is being devoted to the solution of problems that are closely related to the ones being discussed. For instance, the trajectory calculation for collision avoidance is exactly the same problem that has been studied by many authors, among which some have considered the problem of real-time trajectory calculation, which is exactly what is needed here.

The same consideration applies to geometric modeling (that is also of course strictly related to the previous problem).

If a more readily available solution is desired, one can simplify the problem by applying some restrictions that, in many cases, do not significantly limit the performance of the system.

For instance, one may assume that no physical interaction can take place between different robots, or that these interactions are easy to handle (as it happens in cartesian robots). In this case, only collisions between arms and objects are to be dealt with.

An important prerequisite the system to be driven must satisfy is that a multirobot real time control system be available [5]. This is mandatory, since the program that explicitly describes the operations to be performed is only created at run time, and no off-line translations are allowed.

The last two tasks the scheduler must accomplish (and the most peculiar ones) are the physical resource mapping and the displacement of parts that are not within reach of the chosen RAP.

VAP - RAP mapping does not require a great skill. Since VAP's specifications for each task are stated in the program, the job of the scheduler is to find, among the available RAP's, the one that can do the job.

This can be simply done by trying to match the required VAP's parameters with the corresponding parameters of each available RAP.

The mapping process is actually composed of two phases: the construction of a tree (a) and the choice of the best solution(b).

a1. The required VAP is inserted as the root of the tree.

a2. All RAP's are compared against the requested VAP, regardless of their status. All RAP's that satisfy matching conditions are appended to the root;

a3. For each RAP that has been included in the list just described, all effectors that could carry it are appended to the node representing the RAP itself. The structure of the resulting tree is shown in figure 1.

b1. The tree is scanned in order to find the best solution. Since the tree is usually quite small no special search techniques were used so far. The criteria for deciding the value of each solution are quite empirical, and further investigation should be done on this particular problem.

b2. A decision is taken, according to the following rules:

   1. If the required RAP is available, it is already mounted on a suitable effector and the parts to be handled are within its reach, the resource is assigned to the task, and the task is started;

   2. If the required RAP is available, but it is already assigned to another task, and if cases 3 or 4 below do not apply, the task is suspended and periodically rescheduled, until it can be run;

   3. If the required RAP is available, but it is not mounted on any arm, and if a suitable effector is free, a routine is entered that dismounts the RAP that is actually mounted on it and mounts the appropriate one; the task is then rescheduled;

   4. If the required RAP is available, or can be made available, but not in the area where the parts are currently placed, an appropriate routine is entered, that displaces the parts until they are within reach of the chosen RAP. This point was not completely investigated so far;

5. If no RAP exists that can do the required operation, the task cannot be executed, and an error message is issued.

A problem that might occur is that, since no reasoning is being done on the instructions that follow a VAP definition, it could happen that, after having assigned a RAP, the system might discover that the chosen RAP-effector combination cannot perform the task because it cannot reach all requested points. In order to prevent this malfunction, that would be very difficult to correct, the field **reach** has been included in the records that describe VAP's and effectors. It is the user's responsibility to define this area, that must include all points the RAP will have to reach during the execution of the task. In order to maintain things simple, this area is always defined as a parallelepiped, although more accurate geometric descriptions could lead to greater efficiency.

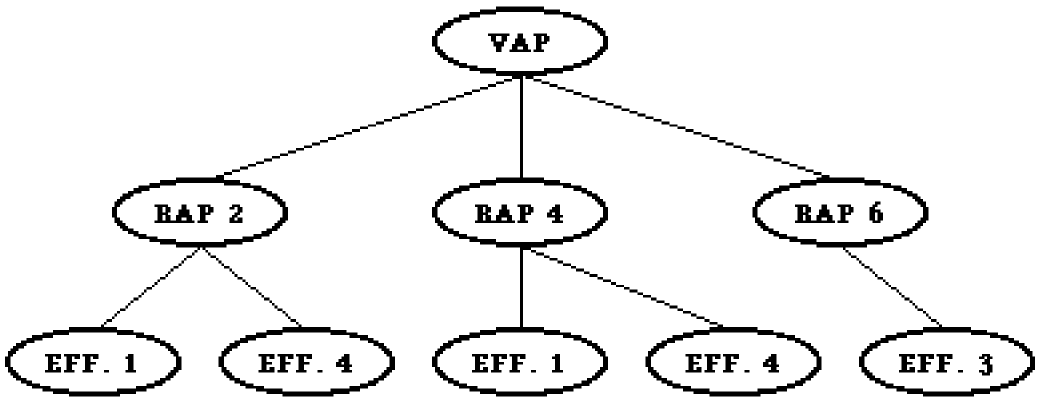

Figure 1. - Resources tree structure.

The first phase of the mapping procedure could also be executed only once prior to program execution. The resulting tree would of course be more complex, because it should include all possible RAP-effector combinations, and the search algorithm would be longer. Investigation is being carried on in order to state which method will be faster to execute.

After a VAP - RAP mapping has been done, all movements of the assigned RAP must be checked for collisions, and, if case 4 has occurred, a coordinate shift must be applied.

The rules just mentioned are now being checked for consistency and inserted in the program. Routines to connect and disconnect RAP's and effectors are being developed.

The block diagram of the resulting run-time system is shown in figure 2.

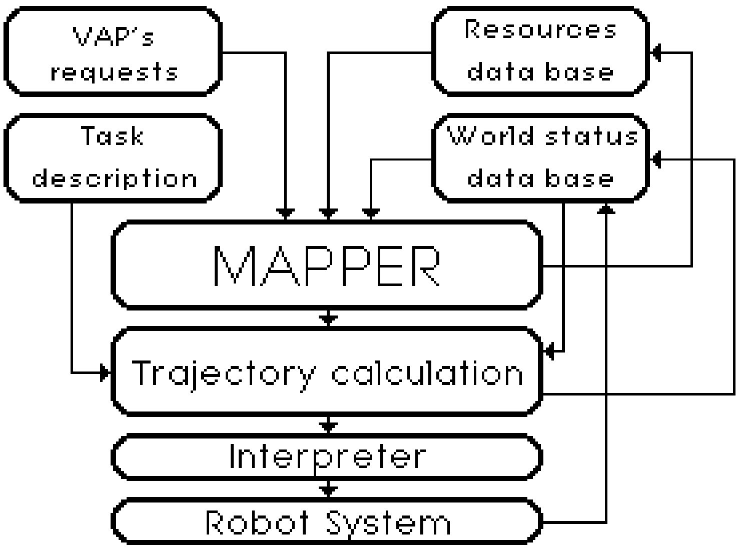

Figure 2. - Run-time system block diagram.

Multiprocessing is performed as in any other time-sharing system, and synchronizing primitives are handled as usual.

5. - THE LANGUAGE.

In order to make a practical implementation of the system possible, a suitable language is required. The Italian standard language VML [6] is thus being expanded to accommodate the new primitives. At the same time, more powerful control structures are being defined and implemented. The resulting language will be named HVML (High level Virtual Machine Language).

In a first phase, rather than actually expanding the existing language, a pre-processor is being implemented (VML in fact already includes all necessary data structures and primitives).

6. - SIDE EFFECTS AND OPEN PROBLEMS.

The programming method just described has a number of interesting side effects.

First, the system will be able to cope with operations of unpredictable length without losing efficiency.

Second, since the resource assignment is only done at run-time, it is obvious that no task will be assigned to an unavailable resource. This means that, if a resource develops a fault during program execution, the system will continue its operation using the other available resources, if a sufficient set of them is provided. The only affected parameter will then be program execution time. This implements a sort of graceful degradation capability of the whole system.

Third, the system will respond very well to dynamic changes of the production cycle: for instance, the activation of asynchronous routines at any time will not affect the execution of the main program.

Although multiprocessors and redundant computing systems are nowadays widely used, transferring their technology to robots is not easy. The main reason for this is that computers operate on data that are easily transferred from one computer to another one, while robots process mechanical parts that are much more difficult to displace. In many cases, however, the problem can be simplified using standard pallets and standard conveyors whenever possible. This is probably the part of the whole system where non-deterministic programs (such as expert systems) could be successfully applied.

Other problems concern the level at which the system should be placed. In theory, such programming method could be applied to the whole factory but, in practice, this would give rise to countless additional problems. Probably the best possible application at the present state-of-the-art is at the processing cell level.

7. - EXPERIMENTAL SETUP.

The described method is at the proposal level, and no practical implementation has been done so far. Some experiments were however carried on to demonstrate the feasibility of the project. For instance, a demonstration program was written on Supersigma robot that behaves as an automatic resource handler.

The goal is to place some bolts on the working table of Supersigma according to a given path. These bolts are placed at random along six straight lines, as it may be seen in figure 3. The bolts are located by passing over the lines with the open gripper, and by using a photocell detector to sense when the gripper is over one of the bolts.

The program was written as two parallel tasks, each one searching on one side of the board. As soon as one of the grippers grasps a bolt, it will read its destination coordinates from a file, that is shared among both tasks.

Therefore, the programmer does not know which arm will place which bolt, because the sequence of the operations depends on the initial position of the bolts.

The program works amusingly, since a very slight difference in the initial position of the bolts may lead to a completely different execution.

A problem is that, in some situation, the two arms may get stuck in a deadlock. In this particular situation, it was easy to implement a third task that monitors the two other and, when it detects that both of them are stuck, it takes some appropriate action (for instance, forcing one of the two arms to go outside the working range of the other one).

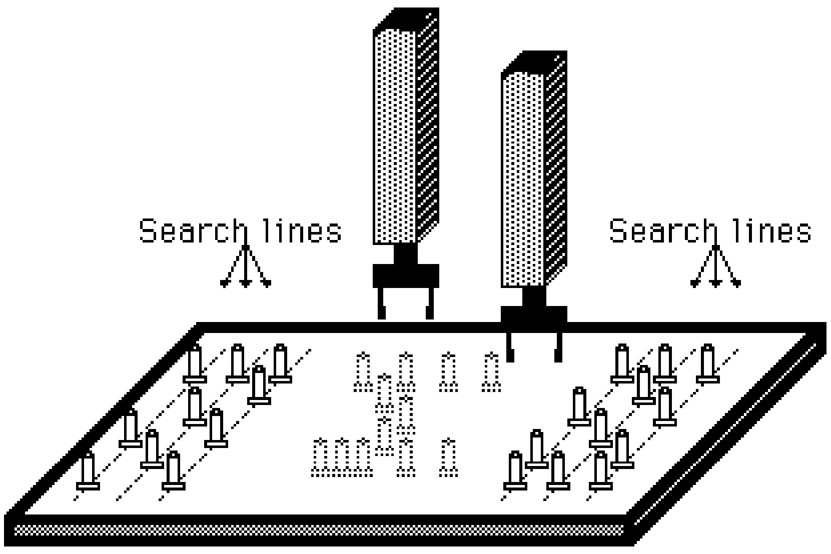

Figure 3 - Experimental setup.

8. - CONCLUSIONS.

The technique of automatic resource assignment described in the paper lays between explicit languages and fully implicit programming systems. Its main advantage is that it can be readily applied in a number of cases where traditional programming would lead to inefficiencies, and that, on the other side, it does not require the sophistication of implicit programming.

On the other hand, the research effort being devoted to the problem will not be wasted when implicit languages will come into use, since in this case automatic resource assignment in any system that includes more than one mechanical arm will be fundamental.

REFERENCES.

[1] Sedillot, S.: "Some Computing Issues in Multiple Robot Systems", in Advanced Software in Robotics, Dantine & Gérardin (eds.), North Holland, 1984.

[2] Fox, B.R., and Kempf, K.G.: "A Representation for Opportunistic Scheduling", in Proc. 3rd International Symposium of Robotics Research, MIT Press, Boston, Ma, 1986.

[3] Cassinis, R.: "An Application of Automatic Resource Sharing to Robot Programming", in Proc. 3rd International Symposium of Robotics Research, MIT Press, Boston, Ma, 1986.

[4] Bisiani, R., and Cassinis, R.: "The Development od a Multi-Micro Processor System to be used in the Control of an Industrial Robot", in Proc. "MI-MI '76", Zuerich, 1976.

[5] Cassinis, R.: "Hierarchical Control of Integrated Manufacturing Systems", in Proc. 13th International Symposium on Industrial Robotics, Chicago, 1983.

[6] Bison, P., Lorenzin, G., and Pagello, E.: "The Formal Definition of VML and a Proposed Portable Implementation", in Proc. 11th International Symposium on Industrial Robotics, Tokyo, 1981.