Planning and Condition Monitoring

in a FMS

Austin Tate

AIAI-TR-2

Paper published in the Proceedings of the International Conference on the Development of Flexible Automation Systems, pp62-69, Institute of Electrical Engineers, Savoy House, London WC2R OBL, UK, 10-12 July 1984.

Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

PLANNING AND CONDITION MONITORING IN A FMS

Austin Tate

Dept. of Artificial Intelligence, University of Edinburgh, U.K.

The paper is intended to introduce Artificial Intelligence work on robot planning systems which may have relevance for production planning, condition monitoring and recovery on a flexible manufacturing system used for small batch production. The paper describes the use of "Goal Structure" as a representation of the "intent" of plan steps and its potential use as the basis for a hierarchic control system for condition monitoring and re-planning.

INTRODUCTION

AI planning and knowledge representation research has provided formalisms for hierarchically representing data to the planning process, i.e. a description of the goals of the plan and the actions (or jobs) of which it might consist at various levels of detail. Planners are able to use this information and a simulation of the effects of the actions to generate, fairly efficiently, plans to solve limited tasks in a wide variety of domains. The methods used allow a natural interface to libraries of detailed pre-programmed sub-tasks (e.g. provided by the manufacturer of a versatile industrial robot).

Present AI systems still fall short of meeting the requirements for a planning and scheduling system which can be employed on complex applications - often typified by the need to interface to several other sub-systems. There is also a need to coordinate planning with condition monitoring and re-planning activities. An approach to the use of information which captures the "intent" of plan steps may offer the chance to make progess in this area.

The techniques are equally applicable to the generation of plans and the monitoring of their execution for a fully automated or fully manual situation as well as the more usual mixed workshop environment. Hence we are suggesting an overall hierarchical organisation within which individual modules for planning or control can operate.

GOAL STRUCTURE AND THE NONLIN PLANNER

Goal Structure (Tate (25)) is a high level representation of information about a plan which states the relationship between the individual actions in the plan and their purposes with respect to the goals or sub-goals they achieve for some later point in the plan. This information may be used by a planner to detect and correct conflicts between solutions to sub-problems when higher level plans are refined to greater levels of detail.

The NONLIN planner (Tate (26)) makes use of Goal Structure for interaction detection and correction. An application domain specification language, "Task Formalism", has been provided to enable the intent of plan steps to be expressed along with levels of detail, ordering of steps and other information about actions.

A Goal Structure table is kept during planning to record what facts have to be true at any point in the network and the possible "contributors" that can make them true. The system is able to plan without choosing one of the (possibly multiple) contributors until this is forced by interaction correction. The Goal Structure table is used to detect important interactions (ignoring unimportant side effects) and can be used to find the small number of alternative links to be added to the plan to overcome each interaction (fully described in Tate (27)). Multiple interactions arising at the same time further restrict the possible solutions and a minimal set of re-orderings can be proposed.

We believe that Goal Structure can be extended to represent information on which an execution monitor can operate effectively. The Goal Structure statements represent precisely the outcome of any operation which should be monitored. If lower level failures can be detected and corrected while preserving the stated higher level Goal Structure, the fault need not be reported to a higher level. The implications of any propogated failure are computable and corrective action can be planned.

APPLICATION AREAS OF AI PLANNERS

General purpose planning systems which can automatically produce plans of action for execution by robots have been a long standing theme of Artificial Intelligence research. A large number of techniques have been introduced in progressively more ambitious systems over a long period. There have been several attempts to combine the planning techniques available at a certain time into prototypes able to cope with realistic application domains:-

- Program generation
  HACKER (23), Waldinger (29)
  Programmer's Apprentice (14,15)
- Robot or factory control
  STRIPS (7), TROPIC (12), ISIS-II (8)
- Engineering maintenance and training
  NOAH (17,18)
- Experimenting in molecular genetics
  MOLGEN (21,22)
- House building and civil engineering
  NONLIN (26)
- Electricity turbine overhaul
  NONLIN (4)
- Journey planning
  Hayes (9), Hayes-Roth & Hayes-Roth (10)
- Spacecraft mission sequencing
  DEVISER (28)
- Aircraft carrier mission planning
  SIPE (30)
- etc, etc.

FEATURES OF AI PLANNERS WHICH IMPROVE THEIR RANGE OF APPLICABILITY

Methods of reducing the size of the search space have been employed in planners. Some are identified here and considered in more detail in the sections which follow:-

- by considering "higher priority" goals first in hierarchic planners
- by detecting and correcting for interactions between solutions to sub-problems in an intelligent fashion
- by checks on resource usage levels, time constraints on actions, etc
- by using symbolic information to restrict ways that goals can be satisfied (e.g. rule-oriented applicability conditions).

In addition, the following sections will describe several features which are essential for the support of a flexible planning system which has to operate in conjunction with other specialised planners or knowledge sources and in the face of execution-time failures:-

- interfaces to sub-planners and other sub-systems
- re-startability at plan-time and execution-time on failures.

Hierarchy/Abstraction Levels

The order in which conjunctive goals are tackled can have a marked effect on the efficiency of the search process for a plan to perform some task. In some planners, it can make the difference between finding a solution and looping round on the same goals repeatedly or getting solutions with redundant steps. One approach to ordering the various goals involves separating them into levels of importance or priority. The more abstract and general goals are worked on first and the more concrete or detailed levels are then filled into the skeleton plan produced. Many AI planners use such a scheme. It was first introduced in the ABSTRIPS (Sacerdoti (16)) and LAWALY (Siklossy and Dreussi (19)) planners.

Goal Ordering and Interaction Detection and Correction

Planners can be split into two basic types with respect to the way that they tackle multiple goals. The "linear" planners make the "linearity assumption"; that solving one goal and then following it by the solution to a second goal is often successful because the solutions to different goals are often decoupled. The "non-linear" planners take a "least-commitment" approach by representing a plan as a partially ordered network (a graph) of actions and goals and only introducing ordering links between actions or goals when the solution demands this.

In both types of systems, the solution to one goal may interact with the solution to the others. Several AI planners can detect and in some cases correct for such interactions.

The "Goal Structure" technique was introduced to record the link between an effect of one action that was a precondition (sub-goal) of a later one. This representation is orthogonal to the temporal links between the actions themselves (as some actions have effects that are used much later in the plan). It also captures the (relatively few) effects of an action that are really required later. Goal Structure was first used in the INTERPLAN linear planner (Tate (24)).

The first non-linear planner, NOAH (Sacerdoti (17)), incorporated code called "critics" which was used to search for interactions between parts of the plan. Critics used a Table of Multiple Effects (TOME) to aid in discovering the interactions. The TOME and critics were themselves based on the Goal Structure tables of INTERPLAN. Once detected, NOAH could correct for interactions in a limited way suited to the applications it was employed on. NOAH did not consider search alternatives - the best choice was committed to at each stage.

Subsequently, the detection of interactions by analysis of the underlying "Goal Structure" was added to a non-linear planner. The combination in NONLIN was more effective than its earlier use in the linear INTERPLAN system. Now, the minimum additional ordering constraints necessary to resolve any interaction could be suggested by the introduction of temporal links into a partially ordered plan only when this became essential. The NONLIN system could consider alternatives if failures on any chosen search branch failed.

Planning with Time, Cost or Resource Limitations

It is becoming increasingly important in planning systems to perform in realistic domains where the use of resources of various types needs to be limited. Also, planners are being used in domains where time considerations and matters beyond the control of the planning system itself must be accounted for. Several systems have explored this area.

An option available when NONLIN is entered brings into play a system that computes Earliest Start Time (EST) and Latest Start Time (LST) information for each node in the network at any stage of planning. The Task Formalism allows a "duration" to be associated with each action in the operator hierarchy. The system also computes the total plan duration and critical path nodes (actions which if delayed will cause the entire plan to take longer to execute).

The NONLIN planner has incremental algorithms to propogate the EST and LST values through the network as expansions are made and links introduced. This continuously maintains the Critical Path Analysis data in a form which allows it to be used during plan alternative selection.

Daniel (3) described a version of NONLIN in which choices of alternative operators and alternative orders of linking to correct for interactions are dependent on a plan cost measure which is the sum of the costs of the nodes in the plan added to some factor of the overall plan duration. She termed this the plan "efficiency". This is used to make

more sensible heuristic choices of alternative search branches. The process is also documented in reference (4). Recent extensions to the NONLIN planner to deal with multiple resources have generalised this scheme to select alternatives according to some specified combination of factors of several limited resources.

NASA's DEVISER planner (28) has taken the ability of hierarchic non-linear planners to cope with time information much further. DEVISER provides the facility to actually specify an EST and LST for any node in the plan (rather than these simply being assumed to be dictated by the ordering of the nodes as in NONLIN). Vere's incremental algorithms for propogating EST and LST information through the plan network can account for these "time window" specifications and can signal when they are violated. The detection and correction of interactions must be sensitive to any temporal displacement introduced. DEVISER also allows externally caused scheduled events to be taken into account during planning and allows for delayed effect events (caused by earlier actions) to be handled (e.g. a spacecraft is steady some time after firing a thruster).

Very much more flexible handling of the propogation of temporal constraints between the steps of a plan is being considered in the widespread research effort on temporal logic (e.g. McDermott (13), Allen and Kooman (1), etc).

The use of objects being manipulated as scarce resources on which usage conflicts occur and need to be avoided was incorporated in the MOLGEN (Stefik (21,22)) and SIPE (Wilkins (30)) planners.

Incremental Algorithms for Computing Time, Cost and Resource Usage

The methods of accounting for time constraints and resource usage in AI planners are characterised by their incremental nature. In Operations Research, it is usual to construct a project network or plan by hand and then to try to balance resource usage or find critical time points or activities in a single computation. This is not suitable where many alternative methods of performing parts of an overall task have to be taken into account or where the situation is highly dynamic and changes must be accounted for at plan time or when failures occur during execution. The balanced use of resources or working within time limits must be used to select between the alternatives available when the plan is actually being constructed.

Hence, incremental algorithms have been developed (4,28) which propogate resource usage levels or time related information through the effected parts of a plan when any change is made. The changes include the adding of a new logical link between activities (effecting the possible start times of later activities), the choice of a particular method of performing a sub-task (effecting resource usage levels, time estimates, etc) and the removal of some plan part to reconsider other alternatives when limits are exceeded or unresolvable interactions are found (again effecting resource usage levels, time estimates, etc).

Task Formalism

The problem domain is described to the NONLIN planner through a completely declarative language called the "Task Formalism". The TF reflects the underlying organising principle of Goal Structure to describe the "intent" of the actions specified for any goals. TF is fully described in reference (26). As an example, a description of a method of performing a decoration task is given in figure 1.

Although the TF parser actually used with NONLIN is very simple, the TF language is designed as an intermediate formalism to which a user front-end with a graphical interface and knowledge based assistance can be added.

The TF operator descriptions allow the specification of "types" for the preconditions. Earlier planning language work (e.g. on PLANNER by Hewitt (11)) has identified the problem with the single precondition type which can either be already true or can induce subgoaling to be made true. The POPLER (Davies (5)) system introduced two precondition types into the PLANNER-like languages - one to simply look up if something is true (perhaps instantiating variables), and the other which is allowed to recursively make the condition true if it does not already hold.

TF extends this notion and mates it with a "process" oriented view of operator descriptions. A TF operator description specifies a method by which some higher level action can be performed (or higher level goal achieved). The operator introduces lower level actions under the control of the operator "manager" (these are his own resources in some sense). He says that something is to be done in order to achieve each part of his job. In TF these are specified as SUPERVISED conditions. The "manager" also relies on other agents to perform tasks that are their own responsibilities, but effect the ability of this manager to do his job. These are given as UNSUPERVISED conditions. There are other conditions which the "manager" may wish to impose on the applicability of particular solutions (i.e. don't bother to try this method for house building if the building is over five stories tall). These are termed HOLDS and USEWHEN conditions in versions of NONLIN.

NONLIN in fact has two other condition types which really relate to other planning abilities and are not strictly part of the process specification aim of the TF. These are QUERY for optional inclusion of sub-plan parts, and COMPUTE to communicate with externally defined computations and data bases or with specialised sub-planners.

The TF condition types provide important search information for the planner and facilitate the Goal Structure based interaction detection and correction facilities. They are used to reduce the alternatives which needed to be considered when dealing with failures at plan-time or during plan-execution.

Interfaces to Sub-planners and other Sub-systems

The AI planning systems so far discussed are only suited to planning at a supervisory level for a manufacturing system. There will be very detailed sub-actions necessary to achieve the stated high-level operations. Some of these will be planned by specialised sub-planners. For example, specialised planners are available for trajectory planning, grasp position planning (Wingham (31), temporary scaffolding erection , etc. In addition, it may be necessary to consult specialised deductive systems (sometimes referred to as "belief" systems) which can answer a query based on computation or rule invocation from the data available at some point in the plan. For example, body modellers (Ambler and Popplestone (2)) to deduce positional information from given relationships between component objects.

The NONLIN planner has a simple interface to such sub-systems. It is based on the "COMPUTE" condition fully described in reference (26). As an example, if a sub-system called POSITION can be given an argument representing an object and can return a vector of its x, y and z positions for its centre of gravity (say), then we can write:

COMPUTE {x y z} = POSITION axle

To make this a useful feature, the following were considered necessary:

a) the ability to pass in the values of the parameters (e.g. axle above) using objects known to the planner

b) the ability to query the state of the facts known at the required point in the plan. This is provided via a planner routine available to COMPUTE function writers which returns the facts matching some given pattern at the point in the plan at which the COMPUTE "condition" is called

c) the ability to match an answer from a COMPUTE function with some pattern which may contain variables to determine whether the outcome is valid. The variables can also be used to pick up partial results needed in the plan or in further COMPUTE conditions

However, it is now realised that it is essential to add the following to any planner/sub-planner interface:

d) the ability to return a set of answers, any one of which would be appropriate as a matching response. Related to this is the ability to accept back a restriction on a matching response which may not be a fully instantiated answer

e) the ability to accept back from a sub-system a set of conditions on the continuing validity of the answer(s)

This final point is vital. For example, if the POSITION routine returns a particular value for the {x y z} of the axle's centre of gravity, this would be based on relationships and other data which were found to hold at the required point in the plan. If the derived information is to continue to be valid, the logical support for the deduction must continue to hold (this is sometimes called "reason or truth maintenance" (Doyle (6)) or "belief").

In addition, the derived information can be used anywhere in the plan where the base information holds. The statement of the support conditions thus specifies the Goal Structure that applies to the results of calling a sub-system. Any interaction with the support for the derived information must be detected and corrected as for any other planner introduced interaction.

A Re-startable Planner

A re-startable planner is necessary to cope with failures at both plan-time and plan execution time. There have been several AI planning systems which are able to keep a record of the inter-relationships between parts of a plan and the dependencies between the choices made to introduce each part (Hayes (9), Stallman and Sussman (20), Daniel (3)). These have been used mostly to limit the amount of an emerging plan that must be discarded when a failure occurs at plan-time to that part which is logically dependent on the failed parts (rather than all plan parts introduced after some particular point which is the cause of the failure). These, so called, "dependency-directed backtracking" systems are also well suited to coping with plan revision after execution-time failures. The plan can be tidied up by undoing all failure-dependent parts and removing all executed sections (altering the initial state model appropriately). Re-planning can then try to generate a repaired plan to achieve the original goals.

PLANNING AND CONDITION MONITORING ARCHITECTURE FOR THE FMS

The control system for a Flexible Manufacturing System is seen as a hierarchy of modules, each providing localised supervision of allocated tasks, condition monitoring, limited recovery and status reporting. At the lower levels, these modules involve hardware sensing and adaptive control or redundancy. At the higher levels they involve business decision support software. The AI techniques may have a part to play in between. This is shown diagrammatically in figure 2.

Figure 3 shows more detail of the interface between the planning system and other parts of a small batch factory automation system. Planning assumes a fixed set of processes. These are supplemented by a product or component design system. The processes are skeleton plans for the manufacture of given products or for the performance of certain manufacturing processes. They are represented at various levels of detail. The skeletons should state as much as is known about the "intent" of each action included (as Goal Structure condition ranges or constraints) to give the planner and run-time execution monitor more information to use during re-planning on failures.

It is the job of the planner to refine a "rough cut" schedule of activities (decided from the order book using an estimate of plant capacity, etc) into a detailed plan of action for the actual plant conditions prevailing. It accounts for consumable

resources and time critical events. The plan is then passed via a resource allocator (at least for resources limited by time) to the execution system. The planner should ensure that sufficient information is available to enable the execution monitor to properly check that plans achieve their purposes.

A MODEL OF PLAN EXECUTION MONITORING BASED ON GOAL STRUCTURE

A plan execution monitor is given a plan generated by a planner together with information on what the individual plan steps achieve by what time for which subsequent tasks (the Goal Structure). It must supervise the allocation of tasks to effectors (based on a capabilities data base which might be trivial or quite complex in nature). It must use any available condition monitoring capabilities to monitor the execution of each task to ensure (as far as possible) that it achieves its purpose(s).

When failures occur, recovery steps may be taken which might be of various types:

- recovery procedures for the effector chosen (e.g. reset and repeat)
- recovery procedures for the task type chosen (e.g. generic procedures for ensuring that a task can be successfully accomplished by passing it to a special purpose effector or skilled supervisor)
- recovery procedure for the particular failed condition (e.g. by replacement of a part or by fix-up actions, etc)

Recovery on failures can be simple or complex depending on the local intelligence of the effectors chosen, the closeness of coupling of tasks in the domain, the predictability of error outcomes, etc. When a failure is found which cannot be locally recovered from within the given Goal Stucture constraints (of required outcomes, resource usage or time limits), the execution monitor must prepare a statement of the failure and changed plan circumstances to pass back to the planner (which can then be used to suggest a plan repair). This is shown diagrammatically in figure 4.

As shown in figure 5, an activity can be executed as soon as all the incomming Goal Structure requirements are satisfied (by any potential "contributor" if there are several alternatives). A decision on the allocation to a particular effector must then be made. The activity is given a tag to identify it through the execution monitor. The activity, its tag and any associated constraint information is then passed to the effector. At the same time, the Goal Structure outcomes of the activity are entered, along with the relevant tag number, into the condition monitoring system (which can examine them to consider how best to use its monitoring systems or sensors to test the conditions).

The relevant effector executes the action and its controller must report when the activity is completed by returning the associated tag to the execution monitor. "Time-out" conditions related to the time limits for the follow on actions to the Goal Stucture outcomes are used to prevent the system hanging up on effector controller failure.

The condition monitor is triggered by receipt of an activity tag to check all associated Goal Structure outcomes of the activity. This same model of execution and condition monitoring applies where the "activity" is the use of a sensor to capture information needed at some point in the plan. The Goal Structure outcomes in such a case may contain variables which will be bound to definite values when the "condition" is checked.

If failures occur, local recovery is possible (by either the effector or by using fix-up plans accessible to the execution monitoring system or condition monitors) within the given resource or time limits set for the follow on activities resultant on each monitored outcome. The parallel Goal Structure (i.e. outcomes of actions before the failed activity which are required later in the plan) provides a guide to the local recovery system on what should be preserved if the local recovery is to avoid interference with other important parts of the existing plan. Any interference with such parallel Goal Structure should be reported to the execution monitor as it must be re-considered by the planner to work out the actual effect on the plan.

ACKNOWLEDGEMENTS

The original work on the use of Goal Structure in AI planners was supported by a U.K Science and Engineering Research Council (SERC) Studentship. The work on NONLIN and Task Formalism was supported on SERC grant number B/RG/9445-5 which was supervised by Professor Bernard Meltzer. Recent work on the use of Goal Structure as a basis for condition monitoring has been supported by an Information Technology Fellowship from Edinburgh University.

REFERENCES

1. Allen, J.F. and Kooman, J.A. (1983) "Planning Using a Temporal World Model" Papers of the International Joint Conference on Artificial Intelligence 1983 pp 741-747 Karlsruhe, West Germany.

2. Ambler, A.P and Popplestone, R.J. (1975) "Inferring the Position of Bodies from Specified Spatial Relationships" Artificial Intelligence, 6 pp 157-175.

3. Daniel, L. (1977). "Planning: Modifying Non-linear Plans" Department of Artificial Intelligence Working Paper 24, Edinburgh University. [NONLIN]

4. Daniel, L. and Tate, A. (1982) "A retrospective on the 'Planning: a joint AI/OR approach' project" Department of Artificial Intelligence working paper no. 125. Edinburgh University. [NONLIN]

5. Davies, D.J.M. (1973) "POPLER 1.5 Reference Manual" Department of Artificial Intelligence Theoretical Pshychology Unit Report no.1. Edinburgh University.

6. Doyle, J. (1979) "A Truth Maintenance System" Artificial Intelligence, 12, pp 231-272.

7. Fikes, R.E. and Nilsson, N.J. (1971) "STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving" Artificial Intelligence, 2 pp 189-208.

8. Fox, M.S., Allen, B. and Strohm, G. (1981) "Job Shop Scheduling: an Investigation in Constraint-based Resaoning" The Robotics Institute, Carnegie-Mellon University, Pittsburgh, Penn. USA. [ISIS-II]

9. Hayes, P.J. (1975) "A Representation for Robot Plans" Papers of the International Joint Conference on Artificial Intelligence 1975, Tbilisi, USSR.

10. Hayes-Roth, B. and Hayes-Roth, F. (1979) "A Cognitive Model of Planning" Cognitive Science, 1979 pp 275-310.

11. Hewitt, C. (1972) "Description and Theoretical Analysis (using schemata) of PLANNER" M.I.T. AI Lab. Memo no. MAC-TR-256.

12. Latombe, J-C. (1976) "Artificial Intelligence in Computer-aided Design - The TROPIC System" Stanford Research Institute AI Center Technical Note 125, Menlo Park, Ca., USA.

13. McDermott, D.V. (1982) "A Temporal Logic for Reasoning about Processes and Plans" Cognitive Science, 6, pp 101-155.

14. Rich, C. (1981) "A Formal Representation for Plans in the Programmer's Apprentice" Papers of the International Joint Conference on Artificial Intelligence 1981 pp 1044-1052, Vancouver, British Columbia, Canada.

15. Rich, C., Shrobe, H.E. and Waters, R.C. (1979) "Overview of the Programmer's Apprentice" Papers of the International Joint Conference on Artificial Intelligence 1979, pp 827-828, Tokyo, Japan.

16. Sacerdoti, E.D. (1973) "Planning in a Hierarchy of Abstraction Spaces" Papers of the International Joint Conference on Artificial Intelligence 1973, Palo Alto, Ca., USA. [ABSTRIPS]

17. Sacerdoti, E.D. (1975) "The Non-linear Nature of Plans" Papers of the International Joint Conference on Artificial Intelligence 1975, Tbilisi, USSR. [NOAH]

18. Sacerdoti, E.D. (1977) A Structure for Plans and Behaviour Elsevier-North Holland. [NOAH]

19. Siklossy, L. and Dreussi, J. (1975) "An Efficient Robot Planner that Generates its own Procedures" Papers of the International Joint Conference on Artificial Intelligence 1973 Palo Alto, Ca., USA. [LAWALY]

20. Stallman, R.M. and Sussman, G.J. (1977) "Forward Reasoning and Dependency Directed Backtracking" Artificial Intelligence, 9 pp 135-196.

21. Stefik, M.J. (1981) "Planning with Constraints" Artificial Intelligence, 16 pp 111-140. [MOLGEN]

22. Stefik, M.J. (1981) "Planning and Meta-planning" Artificial Intelligence, 16 pp 141-169. [MOLGEN]

23. Sussman, G.A. (1973) "A Computational Model of Skill Acquisition" M.I.T. AI Lab. Memo no.AI-TR-297. [HACKER]

24. Tate, A. (1974) "INTERPLAN: a Plan Generation System which can deal with Interactions between Goals" Machine Intelligence Research Unit Report no. MIP-R-109. Edinburgh University.

25. Tate, A. (1975) "Using Goal Structure to Direct Search in a Problem Solver" Ph.D. Thesis, Edinburgh University. [INTERPLAN]

26. Tate, A. (1976) "Project Planning Using a Hierarchic Non-linear Planner" Department of Artificial Intelligence Report 25, Edinburgh University. [NONLIN]

27. Tate, A. (1977) "Generating Project Networks" Papers of the International Joint Conference on Artificial Intelligence 1977, Boston, USA. [NONLIN]

28. Vere, S. (1981) "Planning in time: windows and durations for activities and goals" NASA Jet Propulsion Lab. Technical Report. [DEVISER]

29. Waldinger, R. (1975) "Achieving Several Goals Simultaneously" SRI AI Center Tech Note 107. SRI, Menlo Park, Ca., USA.

30. Wilkins, D.E. (1981) "Representation in a Domain-Independent Planner" Papers of the International Joint Conference on Artificial Intelligence 1983 pp 733-740, Karlsruhe, West Germany. [SIPE]

31. Wingham, M.P. (1977) "Planning How to Garps Objects in a Cluttered Environment" Master of Philosophy Thesis, University of Edinburgh.

```
actschema decor
  pattern {decorate}
  expansion 1 action {fasten plaster and plaster board}
            2 action {pour basement floor              }
            3 action {lay finished flooring            }
            4 action {finish carpentry                 }
            5 action {sand and varnish floors          }
            6 action {paint                            }
  orderings sequence 2 to 5   1 ---> 3   6 ---> 5
  conditions unsupervised {rough plumbing installed   } at 1
             unsupervised {rough wiring installed     } at 1
             unsupervised {air conditioning installed } at 1
             unsupervised {drains installed           } at 2
             unsupervised {plumbing finished          } at 6
             unsupervised {kitchen equipment installed} at 6
               supervised {plastering finished        } at 3 from 1
               supervised {basement floor laid        } at 3 from 2
               supervised {flooring finished          } at 4 from 3
               supervised {carpentry finished         } at 5 from 4
               supervised {painted                    } at 5 from 6
end;
```

Figure 1: NONLIN Task Formalism Schema describing a decoration task

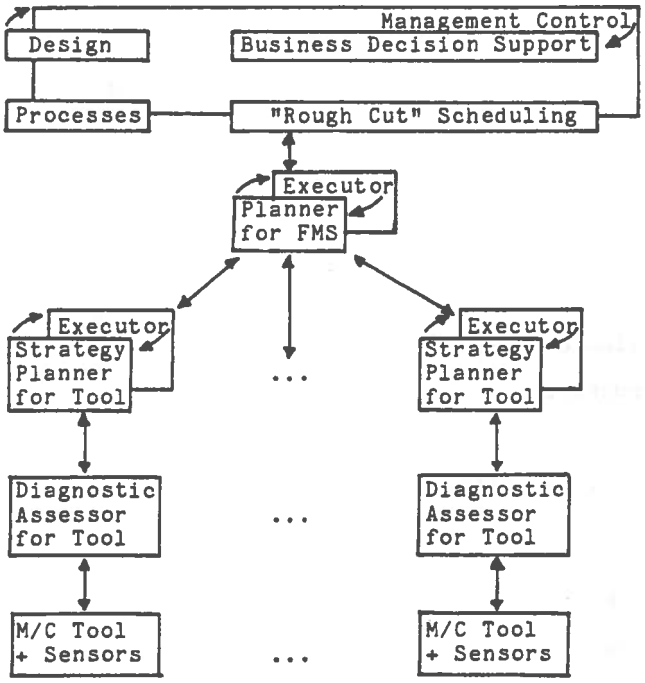

Figure 2: Overall Architecture for a Factory Management System

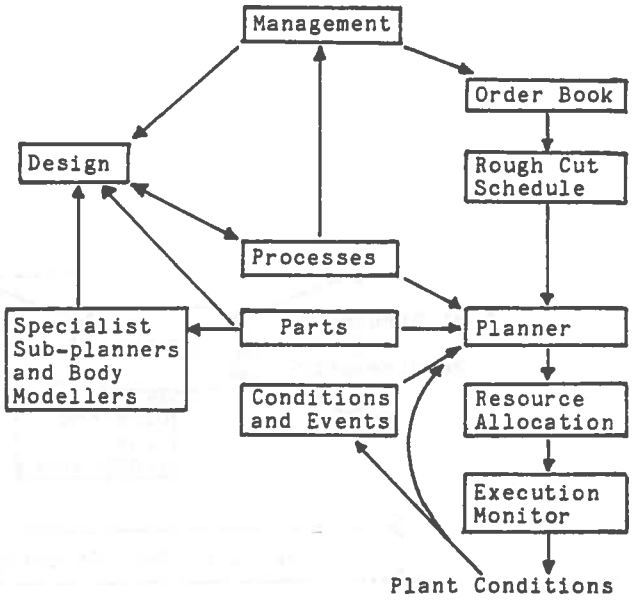

Figure 3: Interfaces between parts of a Factory Management System

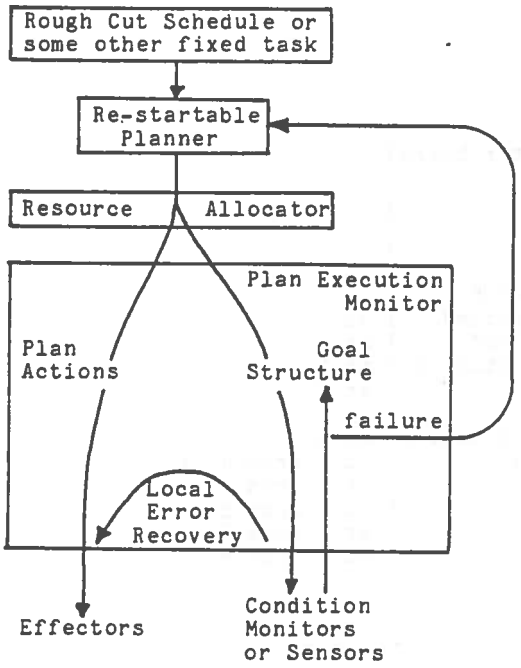

Figure 4: Flow of control of actions and Goal Structure through to execution

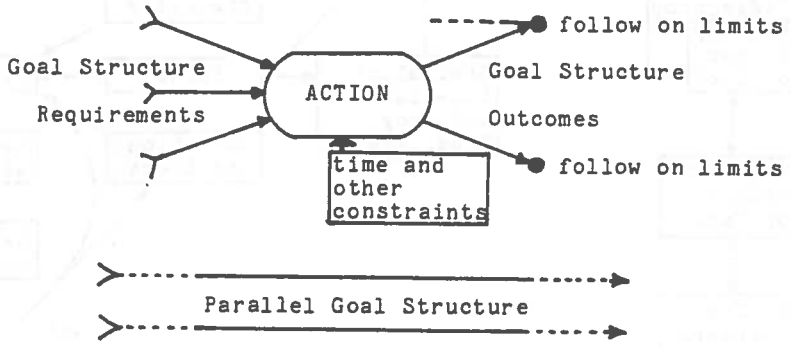

Figure 5: Execution from the viewpoint of a single action