O-Plan: the Open Planning Architecture

Ken Currie & Austin Tate
AI Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

Revised November 1990

Abstract

O-Plan is an AI planner based on previous experience with the Nonlin planner and its derivatives. Nonlin and other similar planning systems had limited control architectures and were only partially successful at limiting their search spaces. O-Plan is a design and implementation of a more flexible system aimed at supporting planning research and development, opening up new planning methods and supporting strong search control heuristics. O-Plan takes an engineering approach to the construction of an efficient domain independent planning system which includes a mixture of AI and numerical techniques from Operations Research.

The main contributions of the work are centred around the control of search within the O-Plan planning framework, and this paper outlines the search control heuristics employed within the planner. These involve the use of condition typing, time and resource constraints and domain constraints to allow knowledge about an application domain to be used to prune the search for a solution.

The paper also describes aspects of the O-Plan user interface, domain description language (Task Formalism or TF) and the domains to which O-Plan has been applied.

1 History and Technical Influences

O-Plan was initially conceived as a project to provide an environment for specification, generation, interaction with, and execution of activity plans. There are three distinct components (see figure 1); the planner's workstation or user interface, the plan generator, and the execution monitoring system. The main effort has been concentrated in the area of plan generation. Plan generation is difficult and it is a classic example of a search problem in AI. The main target of this research is therefore search space control. The outputs of this study are a better understanding of the requirements of planning methods, improved heuristics and techniques for search space control, and a demonstration system embodying these results in an appropriate framework and representational scheme.

The story of O-Plan starts from a software engineering viewpoint, namely how to build an open architecture for an AI planning project with the aim of incrementally developing a system resilient to change. It was our aim at the start of the project to build a system where it was possible to experiment with and integrate developing ideas. Further, the "final" system was to be tailorable to suit particular applications. Section 2 briefly describes the components of the system in which the overall controller and some of the internal "knowledge sources" can be customised or replaced by the end user. O-Plan is intended to be a domain-independent general planning and control framework with the ability to embed detailed knowledge of the domain. Of course, 100% flexibility can never be achieved. The user does have to live with some basic design features and planning philosophies. The primary limitation being the least commitment approach taken by the system.

O-Plan grew out of the experiences of other research into AI planning, particularly with Nonlin [47] and "blackboard" systems [34]. We have included a taxonomy of earlier planning systems (figure 2) which places O-Plan in relation to the influences on its design. It is assumed that the reader is familiar with these works as the bibliography does not cover all of them (see [1] for an introduction to the literature of AI planning).

The main AI planning techniques which have been used or extended in O-Plan are:

- A hierarchical planning system which can produce plans as partial orders on actions (as suggested by Sacerdoti [36]), though O-Plan is flexible concerning the order in which parts of the plan are expanded.
- An agenda-based control architecture in which each control cycle can post pending tasks during plan generation. These pending tasks are then picked up from the agenda and processed by appropriate handlers (HEARSAY-II [28] uses the term *Knowledge Source* for these handlers).
- The notion of a "plan state" which is the data structure containing the emerging plan, the "flaws" remaining in it, and the information used in building the plan. This is similar to the work of McDermott [33].
- Constraint posting and least commitment on object variables as seen in MOLGEN [41].
- Temporal and resource constraint handling, shown to be valuable in realistic domains by Deviser [52], has been extended to provide a powerful search space pruning method. The algorithms for this are incremental versions of Operational Research methods. O-Plan has integrated ideas from OR and AI in a coherent and constructive manner.
- O-Plan is derived from the earlier Nonlin planner [47] from which we have taken and extended the ideas of Goal Structure, Question Answering (QA) and typed preconditions.
- We have maintained Nonlin's style of task description language (Task Formalism or TF) and extended it for O-Plan.

As with most planning systems intended to operate in realistic domains, control of the search and the management of conflict between competing actions has been the focus of the work. The eventual aim of all such systems is to incorporate techniques which should scale up to tackle the expected complexities.

2 O-Plan Architecture

O-Plan is a domain independent planning system and its structure is shown in figure 3.

O-Plan is built up in a succession of layers of functionality in order to support the control requirements in a concise manner. At the lowest level are the basic support modules such as the O-Base *functions in context* database. This is used to provide support for effect and condition maintenance in a context layered fashion. In turn the effect and condition manager maintains "clouds" of (aggregated) side effects and holding periods (ranges) for effects contributing to the satisfaction of necessary conditions in the plan state being developed [49]. Moving up the layers, this is turn provides support for QA (Question Answering) which is the basic reasoning component within the system. QA results drive plan state alterations made by the planner's knowledge sources which in turn are maintained by the net management module. Thus, at the highest level of the structure of O-Plan are the planner's knowledge sources, which include a means to interact with the user.

The Task Formalism or TF domain description is compiled into data structures, to be used during the plan generation process - in particular, activities are represented as *schemas*. The left hand side of figure 3 denotes the *plan state*, which comprises the emerging plan (based on the partial order of activities), the list of plan *flaws*, and internal detail such as the Goal Structure (c.f. Nonlin's GOST [47] and SIPE's plan rationale [53]), the effects of activities (c.f. NOAH's Table of Multiple Effects or TOME [36]) and plan variables. The flaws are posted onto *agenda* lists, which are simply lists of outstanding tasks to be performed during the plan generation phase, and are picked off by an overall controller to be processed by the *knowledge sources* in the middle of the diagram. These in turn may add detail to the plan state, for example by expanding actions to greater levels of detail, establishing how conditions are satisfied (via the QA procedure) or posting new flaws as a result of detecting interactions.

The process has some similarity to the basic flow of control in blackboard systems. The design, coupled with the modularised software engineering approach taken, has provided O-Plan with sufficient flexibility to support our research. There are a range of flaw types and each is matched with an appropriate knowledge source which can process the particular flaw. Recognised flaw types (and hence knowledge sources) include `expand`, `conditions, linkings, effects, variable bindings` and even the `user`. This approach allows for the extension of the capabilities of the system. Table 1 shows the set of knowledge sources used in O-Plan.

Flaw descriptions need not be fully instantiated. However, the knowledge sources can only process fully specified tasks so O-Plan maintains two agenda lists holding outstanding flaws. The main agenda is used to post fully specified flaws, which can be immediately scheduled by the controller for processing, and a second agenda holds those flaws which require further detail before they can be released for handling by the relevant Knowledge Source. Once these are fully detailed they can be moved over to the main agenda and scheduled as normal. There are scheduling related agenda entries to give flaw ordering advice to the controller (we call these `sequence` agenda entries).

Control of the agendas centres around the ability to pick off the "next most opportune" flaw at each planning cycle (a cycle starts and finishes with the picking and processing of a flaw) and this relies on being able to recognise what is the next best thing to do. It is difficult to achieve this ideal at this primitive flaw level for all flaw types as the controller cannot be equiped to know every issue of search space control: some aspects require detailed knowledge which is domain specific, for instance. The various means to achieve conditions, in particular, offer the opportunity of control of the search at a different level in the system from the controller and this became one of the issues of this study. Our understanding of the dynamics of the plan generation process is the limiting

Knowledge Sources	
`expand_node`	Uses schemas to add detail to the plan by expanding actions
`condition`	Satisfies the various types of conditions on actions in appropriate ways
`effect`	Records effects and checks for interactions
`jotter_effect`	Allows for information to be associated with the overall plan state rather than with some specific action
`object_var`	Restricts or binds the value of a variable in the plan state
`alternative`	allows alternative choices to be represented and considered
`or`	Handles the network linking disjunctions and alternative variable bindings suggested by the QA process
`sequence`	Logically relates a number of plan flaws to be scheduled in a pre-fixed order
`user`	In the prototype O-Plan system, the user has access via a high priority knowledge source with interaction capability with the system

Table 1: Main O-Plan Knowledge Sources

factor to the implementation of a totally dynamic flaw controller which can exhibit opportunistic behaviour. One influencing factor, for example, is recognition of flaws on the agendas which have become out of date. If the introduction of an effect or condition requires a series of ordering links to be included in the partial order (held as a type of flaw), then we have to be able to recognise that working on one suggested linking may make other flaws already on the agenda out of date.

Let's be clear about *why* there may be more than one suggested link on the agenda at any point in time (see Sacerdoti [36], Tate [47] or Wilkins [53] for description of resolving interactions). If an operator is introduced into the plan it may itself introduce more than one `effect` (as well as `conditions, actions`, *etc.*). If in introducing these effects into the plan we find that each causes a (different) potentially harmful interaction in the existing partial order then we have to resolve these interactions. This is done in O-Plan, as in Nonlin, by suggesting linking strategies between nodes in the plan which resolve the problem — in general there will be choice of how to link to remove any single interaction. If we maintain the aim of search space completeness then we have no *a priori* reason for performing one linking action before any other. O-Plan recognises this and hence puts these flaws on the agendas awaiting opportunistic scheduling by the controller. When one of these flaws is actually handled then the others may become out of date, since the partial order may further restrict the orderings possible. It has to be reworked to reflect the changes made to the network. O-Plan includes means to efficiently limit outstanding order linking flaws as such changes take place. O-Plan however does not perform true opportunistic scheduling in all respects. It is a manifestation of the general problem of the *ordering* of possible choices which provided the focus of much of our work.

O-Plan currently assigns priorities to every flaw placed on the agendas at the time they are placed. The priorities are calculated from the flaw type and the degree of determinacy of the flaw which is a measure of choice within the flaw. Flaws may still contain choice, even when all detail describing the choice becomes fully ground and the flaw is thus available to be scheduled for processing by the relevant knowledge source. The ability to build up triggering information around an agenda entry in an incremental way prior to knowledge source activation is an important feature that ensures that work done in checking conditions can be saved as far as possible. There are some similarities to newly proposed real-time responsive architectures such as RT-1 [40].

The problem of ordering of choice making is crucial and often ignored. In his work on the TWEAK planner, Chapman [8] says "I'll assume that the nondeterministic control structure always guesses right the first time". He was talking about goal ordering [1] at this point, and this is probably the most important ordering problem in planning but not the only one. To motivate the problem even more, consider how fast the search space defined by Chapman's Modal Truth Criteria (the equivalent of Nonlin and O-Plan QA) might expand. Suppose that a typical partial plan for the block world has on average 4 outstanding goals (hardly an excessive number). Also, suppose that there are on average 3 ways to achieve each of these goals (very plausible, with the possibilities of binding variables, ordering operators, and introducing new operator schemata). This gives us, on average, 12 ways to change an arbitrary block world plan into another one. Each change is designed to remove (at least) one goal (and perhaps introduce others). For an average block world problem, suppose that 7 plan modifications are required to change an initially provided plan into one which has no outstanding goals. This tells us that breadth-first search will explore (at worst) 12^7 partial

[1] AND ordering – which goal to tackle next.

plans. For a seemingly trivial block world domain, a search space of this size is remarkable. If a means of correctly ordering these goals can be found (planning without the guesswork) then this space will be drastically reduced, surely one of the prime aims of planning. We cannot realistically expect such search space control issues to lie solely with the system controller.

Dependency information to guide choice is another story and is difficult to use. This may seem surprising since several systems already claim to be using "dependency directed backtracking". In practice, only limited use of dependencies is made in implemented planners. Dependency directed backtracking (and other backtracking schemes) seems rather impotent when one considers their role in search guidance. They simply allow you to *search* the search space without an excess of repeated effort. What one really wants to do is *prune* the search space. This is the subject of section 4 of this paper.

Finally control in O-Plan can be driven by the user. The user can elect to control each and every cycle of the planning phase, making every choice at every stage, or the user can elect to intervene selectively or even not at all. The (computer) system can detect the user's desire to intervene and can schedule a high priority `user` flaw on the agendas for immediate attention: another example of uniformity in the system. In practice though it is very difficult for even an experienced user to take effective control of the system. This is due to the complexity of the AI planning process itself. Further details of control in the Open Planning Architecture can be found in [12].

3 Domain, Task & Plan Representation

In O-Plan we were guided by the desire for uniformity in task description, goal or problem setting, action representation (the inputs) and plans (the output). The representation chosen is a means of specifying the overall effects of the application of an operator on a world state. Operator descriptions, or operator schemata, are action characterisations. Each schema describes an action (or more generally a process) in terms of the action's preconditions and effects, relevant temporal and resource constraints, and its action description. As with many other planning systems O-Plan adheres to the notions first used in a basic form in STRIPS[25] to describe these preconditions and effects of the operators as well as describe the action associated with the operator. O-Plan operators make use of a basic building block which is a pattern which can represent a function of arguments with a value, not necessarily restricted to binary truth values:

{function arg1 arg2 ...} = value
e.g. {on Block1 Block2} = true
{filter_colour camera_1} = red

Supporting these patterns is the *functions-in-context* datastore which holds the entity/relationship data as statements with values *in context* [48] (also called "viewpoints" or "possible worlds" in other systems). Context layering provides efficient storage of a possibly rapidly changing database by layering alterations made to earlier states of the database. This allows search to be continued in parallel if necessary in the system, by supporting the backtracking functions required during the (failure of the) search processes. The model also allows for retrieval of partially specified items in the database. Except for the first word (the function predicate) the other terms in a pattern can specify ground or non-fully ground atoms. It can even include other nested patterns. Support for these operations in O-Plan is efficient then in terms of storage and lookup. Hardware support for the functions-in-context data model has been considered [31].

Another important building block for the data structures used in O-Plan is that of *min-max* pairs specifying lower and upper bound values on some numeric quantity. Numeric quantities are important in the system as O-Plan uses time and resource constraints to improve search space pruning. The algorithms implemented to perform this pruning are most appropriate in a planner where the values of *plan variables* affect actual resource usage specifications, start times, finish times, durations, and waits (or delays) on activities. A fundamental philosophy in O-Plan is that of a least-commitment approach, thus it is possible that plan variables will not be fully instantiated at intermediate stages of the planning process. So, a pair of numeric bounds are used to represent any uncertain numeric quantity in the system.

The numeric lower and upper bounds (which might be as imprecise as *0* to *infinity*) may be derived from symbolic formulae containing other variables which are not fully instantiated. As variables are restricted or further instantiated, the formulae affected are reconsidered to see if improved (tighter) numeric bounds can be computed. Symbolic interval arithmetic, such as described in Bundy [42], may be used to derive this, though this has not been included at this stage.

Pairs of bounds on quantities which a planner must reason about are useful representational tools in the following situations:

- Aggregation over alternatives (e.g. resource usage may not be uniquely defined until one of several alternative methods of performing an activity is chosen).
- Uncertainty in modelling the application domain (e.g. inability to specify the exact duration of an activity or know the exact execution time situation).
- Flexibility in modelling the application domain (e.g. an activity may start at any time during some given interval).

In practice, there may be a third component of any numeric variable which we term the *projected* value. This is used to record predictions of actual values for heuristic selection, among other purposes.

3.1 The Operators

The actions of the domain are described to the system through the Task Formalism (TF) operator representations. These operators have to include the pattern based descriptions (effects, precondition, etc) as well as the numerical constraints and any plan fragments outlining the action expansion details of the (high level) operator. O-Plan uses the {`function arg1 arg2 ...`} `= value` notation for effects, where the value is `true` by default but can take *any* value including `false`, numbers, elements of enumerated sets, etc. In this way, only an `effect` list is necessary rather than separate add and delete lists that were required in STRIPS since the value `false` can be used to "delete" an effect. The aim of the Task Formalism is to produce an improvement in descriptive power over the previous plan description language used in Nonlin. We are aware of the limitations of the ad-hoc nature of the result. We have been pre-warned; Bundy [7] noted that the AI world "abounds with plausible looking formalisms, without a proper semantics. As soon as you depart from the toy examples illustrated in the paper, it becomes impossible to decide how to represent information in the formalism or whether the processes described are reasonable or what these processes are actually doing". Even though we have gone for expressiveness, coupled with ease of use, the actual coding up of a domain description in TF remains a difficult job.

One powerful means of restricting search in a planner is to recognise explicit precondition types, as introduced into Nonlin. Conditions play a greater role in O-Plan than in previous systems since there is no *special* notion of *goal*. Nonlin style goals are simply `achievable` conditions in O-Plan. Other condition types include:

supervised: satisfied by a sub-activity within the same overall activity.

unsupervised: satisfied outwith the overall activity by some outside agent, possibly another "contractor" working earlier or in parallel.

query: it was recognised that some conditions were used in operator descriptions in earlier planners to ensure a variable was bound at some required point. Making this explicit can give the planner a lot of information about the importance or otherwise of the work it will be involved in. Some of the success of this project relates to the embedding of Goal Structure, or teleological information, into the eventual plan structure and this condition type shows a simple example of noting a low level of commitment to a chosen binding.

only_use_if: of similar intent to the Nonlin `holds` or `usewhen` condition [2] but this tidies up the confusion and commitment to satisfaction that was required in that system. This is satisfied by environmental information known at the time this condition becomes relevant. It has the effect of operating as a gate condition on the applicability of operators.

Other condition types can be identified but the problem is where to stop. This is worthy of a serious study, in its own right, into control in planners via condition types and could form an ideal Ph.D. topic.

Condition typing allows information to be kept about when, how and why a condition present in the plan has been satisfied and the way it is to be treated if the condition cannot be maintained. However use of this information itself will almost certainly commit the planner to prune some of the potential search space thereby losing claims of completeness of search if the TF writer uses an inappropriate condition type. Unfortunately this puts a great burden on the domain writer and makes domain writing a job more suited to the AI or search space expert. The other extreme is the TWEAK type ([8]) formal approach which almost necessarily includes no search control issues. Chapman's work therefore provides a description of the search space, but not a specification of how to control search in that space.

Now let us see how these details are used in writing operator descriptions.

[2] Nonlin employed the `usewhen` condition type as a filter on the applicability of a particular schema to the current situation. For instance if objects to be moved were heavier than the limit on a particular lifting device then schemas mentioning this device should not be considered.

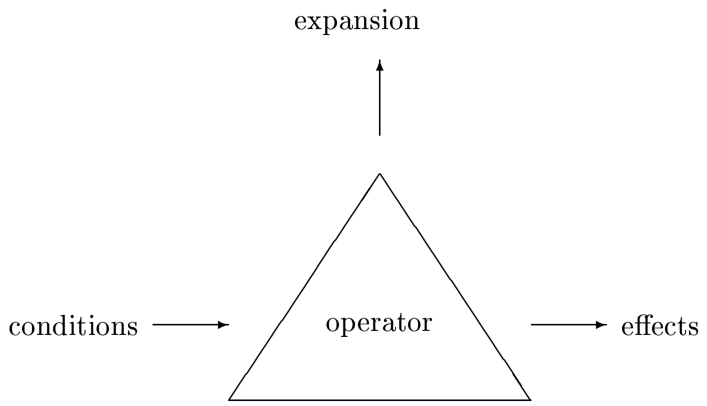
3.2 A Block Operator

The operator schema below shows how a simple `puton` operator for the block world would look. This is not the only way to code up this particular action descriptions.

```
schema puton;
  vars ?x = undef,  ?y = undef,  ?z = undef;
  expands {put ?x on top of ?y};
  only_use_for_effects
              {on ?x ?y}    = true,
              {cleartop ?y} = false,
              {on ?x ?z}    = false,
              {cleartop ?z} = true;
  conditions  only_use_for_query {on ?x ?z}    = true,
              achievable         {cleartop ?y} = true,
              achievable         {cleartop ?x} = true;
endschema;
```

The `expands` statement describes the parameterised action description, used in this instance to annotate a single plan network node that would result from using this scheme. `only_use_for_effects` and general `effects` are separated to differentiate between those effects that would provide a reason for using the schema and those which are regarded as side effects. This operator schema has no side effects - all the effects available are considered as usable to satisfy conditions elsewhere within a plan. Note that there is no separate add and delete list for the effects but that there is the notion of retraction or negation of an effect using the value associated with the effect.

An operator schema can be thought of as a triangle with two points of entry. One relates to the possible use of a schema to provide an effect to satisfy a condition (this is the case for this block world schema), the other as a means of refining or expanding an action to a lower level of detail. Look below for an example of this. The third side of this triangle is the pre-conditions.



The conditions in `puton` are split between two that are required to be `achieved` during the planning process in order to be able to use this schema (equivalent to `goals` in Nonlin), and one which is used to bind a variable. This is an internal consideration for the planning system, but can be read as "I need to know, at some stage, what block ?X is on, and you can be assured that it is on something". This is an indication that condition typing can be extended to assist with the problem of choosing the time to seek to make a variable binding.

3.3 An Action Oriented Operator

The following two operators introduce other features of TF, almost completely orthogonal to those above. However, these operators are more typical of those used in realistic application of O-Plan as against the more *puzzle* oriented operators used in domains such as block stacking. The following house building examples show the use of the `expands` statement as the entry point to the schema in order to refine a high level action into lower levels of detail. This is the more normal use of TF, and is the way in which hierarchical planning operates in O-Plan. The `nodes` and `orderings` statements introduce the plan fragments which provide this expanded detail. We also introduce other condition types into this example, and show a few examples of how the *min-max* bounds on numerical quantities are used (more would be present in a full description of the operators shown). Remember that O-Plan uses both time and resource constraints to prune the search space. The information to allow O-Plan to do this comes through operator and task descriptions. Time windows can be specified on start, finish and duration times of activities and on delays between successive activities, and denote times which are `between`, `occurs_at`, `after`, `before` or even `ideal` for the activity, which in turn is attached to a node in the expansion (and therefore eventually the plan). The notation used below specifies day and time of day; it is terse and future systems would need to associate calendars with the internal time representation. The resources are specified similarly with resource type and the optional `unit` description. The importance and handling of these specifications is explained later. The resource specifications are obviously explicit specifications but we realise that other resource conflicts play a big part in plan generation and we discuss later how these conflicts necessarily introduce the need for a *scheduling* line of reasoning to be taken during planning.

```
schema build;
  expands {build house};
  nodes     1 action {excavate, pour footers      },
            2 action {pour concrete foundations    },
            3 action {erect frame and roof         },
            4 action {lay brickwork                },
            5 action {finish roofing and flashing  },
            6 action {fasten gutters and downspouts},
            7 action {finish grading               },
            8 action {pour walks, landscape        },
            9 action {install services             },
           10 action {decorate                     };
  orderings 1 ---> 2,  2 ---> 3,  3 ---> 4,  4 ---> 5,
            5 ---> 6,  6 ---> 7,  7 ---> 8;
  conditions supervised {footers poured       } at 2 from [1],
             supervised {foundations laid     } at 3 from [2],
             supervised {frame and roof erected} at 4 from [3],
             supervised {brickwork done       } at 5 from [4],
             supervised {roofing finished     } at 6 from [5],
             supervised {gutters etc fastened } at 7 from [6],
           unsupervised {storm drains laid    } at 7,
             supervised {grading done         } at 8 from [7];
  resources bricklayers = between 1 and 2 persons at 4;
  time_window start between 1~11/30/00 and 1~14/30/00 at 2,
              start between 1~12/00/00 and 1~14/00/00 at 3;
endschema;

schema service_1;
  expands {install services};
  only_use_for_effects {services installed} = true;
  nodes     1 action {install drains           },
            2 action {lay storm drains         },
            3 action {install rough plumbing   },
            4 action {install finished plumbing},
            5 action {install rough wiring     },
            6 action {finish electrical work   },
            7 action {install kitchen equipment},
            8 action {install air conditioning };
  orderings 1 ---> 3,  3 ---> 4,  5 ---> 6,  3 ---> 7,  5 ---> 7;
  conditions supervised {drains installed       } at 3 from [1],
             supervised {rough plumbing installed} at 4 from [3],
```

```
              supervised {rough wiring installed  } at 6 from [5],
              supervised {rough plumbing installed} at 7 from [3],
              supervised {rough wiring installed  } at 7 from [5],
            unsupervised {foundations laid        } at 1,
            unsupervised {foundations laid        } at 2,
            unsupervised {frame and roof erected  } at 5,
            unsupervised {frame and roof erected  } at 8,
            unsupervised {basement floor laid     } at 8,
            unsupervised {flooring finished       } at 4,
            unsupervised {flooring finished       } at 7,
            unsupervised {painted                 } at 6;
endschema;
```

Two schemata are included here demonstrating the hierarchical nature of TF, as the `build` schema above contains actions further refined by the `install_services` schema below [3]. In this application most effects are introduced at the lowest level of description. The use of this style of description is in contrast to that in the block world example given earlier.

3.4 A Goal Schema

For uniformity, problems are set to the planner in exactly the same way as describing an operator, though the schema name used starts with the keyword prefix `goal_`. Here is an example of the Sussman anomaly in the block world [44]. Recall that the a condition or effect has a
tt true value by default. In fact much of this rather explicit description would be available by using defaults, essentially allowing only the initial effects list and the final `achievable` conditions list to be given.

```
schema goal_sussman;
  only_use_for_effects {goal} at 2;
  nodes 1 start,
        2 finish;
  orderings 1 ---> 2;

  conditions achievable {on a b} at 2,
             achievable {on b c} at 2;

  effects {on c a} at 1,
          {on a table} at 1,
          {on b table} at 1,
          {cleartop c} at 1,
          {cleartop b} at 1;
endschema;
```

[3]Nodes not explicitly mentioned in the `orderings` statement are considered to be unordered with respect to the other nodes, i.e. they are in parallel with the other nodes.

Although this looks lengthy, it is essentially a template with some detail edited into it. Proformas for goal schemata are used in the front-end to the O-Plan system to simplify this. The `conditions` are used as the means of describing the final (or goal) state requirements and the `effects` are used to describe the initial world state. Planning systems generally assume that the initial state description is *complete*, though the final state need only mention those conditions that must necessarily hold. This requirement may restrict the applicability of such systems. The `nodes` statement introduces a plan expansion, which in this case is a dummy plan: our starting point. At any stage of plan generation there is always a complete plan with outstanding "flaws". Here the outstanding plan contains two dummy nodes one following the other, and the flaws are the two conditions that are not yet achieved. There is a complete plan at any point of the process. At the end, success is implied by there being no outstanding flaws.

To initialise the house building example the `goal_schema` is as follows:

```
schema goal_build_house;
  only_use_for_effects {goal} = true at 2;
  nodes 1 start,
        2 finish,
        3 action {build house};
  orderings 1 ---> 3, 3 ---> 2;
endschema;
```

Here there is a third node in the initial plan which "completes" the work of building a house. The flaws in this plan is that the action mentioned is not primitive and must be further expanded. Note the ordering; our convention always has node 1 as the start node and node 2 as the finish node of a plan. This assists in automatic manipulation of goal proformas in a suitable User Interface.

3.5 Plan Representation

The basic plan representation in O-Plan is based on the partially ordered network of activities as used in NOAH [36] with the action being represented by the node in this network. This partial order is an important component of the main data structure in O-Plan, namely the *plan state*. At any stage of the planning process all relevant information required in the plan is contained in this structure. This includes all information about alternative paths still to be searched in the space, non-fully ground variables in the plan, the Goal Structure and the flaws still outstanding. Within O-Plan, the plan state is held in a "context layered" fashion to allow O-Plan to explore alternatives without generating very large internal structures.

Search in O-Plan is goal directed and progresses through a space of partial plans with the application of a planning operator transforming one partial plan into another. This is a very different search regime from that of simple state space search. The representation of plans as partial orders of action descriptions leads quite naturally to the use of a least commitment approach during search since the partial order deliberately does not commit to total action ordering, until such time as the plan is executed. On top of the least committment to action orderings, O-Plan employs a least committment approach to variable binding and operator selection.

The Goal Structure (GOST) in Nonlin has been adopted and extended to hold information giving detail of the scope of protection of an effect required to satisfy a condition. This is in the form

$\{< how >< what >< from >< to >\}$
e.g. {achieved_by_linking, {on block1 block2} = true, [7 5], 4}

This example says that a condition `{on block1 block2}=true` is required at node 4 and is satisfied by virtue of requisite `effects` being asserted at nodes 7 and 5. The requirement is that there must be at least one requisite `effect`. Therefore, either of node 7 or 5 would be sufficient to contribute to the satisfaction of the condition. The `how` gives a good measure of commitment to a particular Goal Structure entry and can take several forms. A GOST entry does not simply record a range, or holding period, for an effect/condition pair but says a lot about what went on to establish this bond. A GOST entry with a high degree of commitment imposes control by causing a restriction of the search space since the protection interval specified by the GOST entry itself is a constraint on that space.

There is uniformity of representation of plans, actions and goals. This uniformity allows the planner to work to a specific level of detail of the plan which can then be used as aguide for the next level of planning. O-Plan is not confined to expanding nodes at one level to a more detailed level in a set manner. The agenda based nature of the system is also capable of enhancing this abstraction approach provided that the controller is fed sufficient information. The intention of this approach to abstraction is to enable the definition of an abstraction level to be dynamic or opportunistic according to the current context of the planning problem, rather than be statically defined by the input domain description or by the human planner interacting with the system.

The partial order on actions and Goal Structure, augmented by time and resource information at the nodes, makes the basic condition/effect interaction detection and correction algorithms used during the plan reasoning more complex than in the Nonlin system. The mechanism used is the QA (question-answering) process developed by Tate in Nonlin [47] but extended to handle metric time. The goal remains the extension of the reasoning to support resource alloction to activities and disjunctive plan construction in order to increase the flexibility of final plans by offering a measure of contingency. This is a necessary requirement for many practical application domains though it has not yet been solved. We have designed the type of plan representation we wish (C-plans [20]), and we have experimented with a disjunctive form of QA [38]. We remain active in this area of research and in the related area of execution of such disjunctive structures [26, 43].

4 Tackling Search

AI planners to date have generally taken a very simple view of search control propped up by a backtracking scheme. Our aims were to understand the nature of the search space and then to implement search control heuristics which would prune that search space. Further, it was necessary to understand the requirements for effective plan repair on failure either during generation of the plan or on execution. We believe that we have had some success in developing heuristics for pruning search, some of which were achieved through difficulties encountered in developing dependency directed backtracking and plan repair schemes. However we further claim that we have reduced the need for backtracking, and hence plan repair, at plan generation time. The needs for plan generation and execution seem to require:

- schemes to minimise the need for backtracking and repair
- proper repair schemes to excise the faulty parts of a plan whilst effecting a repair to the remaining part of the plan

The former we have looked at, the latter is part of our on-going research work. We shall now describe the search space that O-Plan works in, and then some of the techniques and heuristics developed in O-Plan for control and pruning of that search space.

4.1 A Definition

AI planning systems are charged with trying to produce a *plan* which is a possible solution to a specified *problem*, or some task specification. In O-Plan this works as follows. We start by giving O-Plan an input problem (specified in a complete but flawed partial plan) and a set of action descriptions to work with; it returns a plan for the task in hand in terms of the given actions.

4.1.1 Partial Plans

The plans produced by O-Plan are networks. The nodes in a network denote actions, and the arcs signify an ordering on action execution. Each node has information associated with it which describes the action's preconditions and effects. Preconditions are the "logical" conditions which must hold before the action can occur. Effects are those logical conditions which will hold after the action is completed. Resource information can also be associated with each plan node. Since actions can require and release resources, each plan node can say something about action-specific resource requirements.

Each plan network can be layered into levels of abstraction. This simplifies O-Plan's construction task, since process details can be hidden until they must be considered. It also makes it easier for a user to specify and understand a plan, since the myriad confusing low-level details associated with a complete plan need not be considered at the higher abstraction levels.

A problem is specified to O-Plan as an incomplete or flawed plan. The plan will be incomplete or flawed in the sense that it will have some parts missing (i.e. more detail is required), and other parts that will not work as required (e.g. there is a conflict between competing actions in the plan). O-Plan's task is to correct this input plan so that it *will* work as required. To do this, it needs to modify the input plan until a "flawless" plan is produced. Plan modifications are designed to add a required plan component to achieve an unsatisfied condition, to expand actions to lower levels of detail, or to rearrange the plan so as to remove an interaction flaw, etc. There will often be more than one plan modification possible; that is, there will often be a choice of how to achieve a goal or how to fix a fault. These choices lead to *search*. O-Plan searches through a space of partial plans, modifying one plan to obtain another. It seeks a complete plan that is free of faults.

4.1.2 Plan Modification Operators

To change one partial plan into another, O-Plan uses the application of Plan Modification Operators (PMOs) which are embodied in Knowledge Sources. Example PMOs are available to

- To "expand" an abstract action by choosing from amongst the lower level actions available that match the higher level action description
- To choose action descriptions which will satisfy outstanding, required conditions in the plan
- To correct for interactions against a protected interval in the Goal Structure by proposing minimal changes to the action orderings that remove the interaction
- To select instances for object variables

The Knowledge Sources which cope with expanding an abstraction or achieving an unsatisfied condition use a library of *schemata* to do this. O-Plan must be given schemata if it is to be applied to the problem of generating plans in a given domain. The language used to communicate task specification and schemata to O-Plan is called *Task Formalism*, or TF. Schemata are simply parameterised plans. Each schema looks exactly like a fragment of a regular plan. In place of the names of specific objects, dates, and places, there may be parameters. These parameters can be filled as appropriate for any given problem. With a flawed plan and a set of operator schemata, O-Plan will examine the schemata in order to determine which can expand some high level action or how their addition can remove unsatisfied condition flaws. If no schema is available to remove a flaw, then there is little that can be done. More likely however, there will be a few schemata in the set which offer help in removing a plan flaw. This set of possibilities represents a choice point. O-Plan will choose one of the possible schemata heuristically, and then attempt to remove the remaining plan flaws. If it turns out that the remaining flaws cannot be removed, O-Plan is prepared to *backtrack* to the choice point in order to explore the ramifications of the available alternatives. Backtracking is not necessarily in chronological order of choice point creation. A heuristically weighted search over all open choice points can be made. This means that O-Plan does not commit to any particular solution, and, eventually, is prepared to consider all possible solutions in its search space. Of course in the interest of efficiency, users almost never require all solutions, and are instead happy with a single feasible solution.

4.2 Generators for the Space

O-Plan follows its predecessors INTERPLAN [45, 46, 47] and Nonlin [47] in using a tightly constrained method of generating the nodes considered in the search space. This is based on the use of an initial abstract plan with a least-commitment representation of actions and orderings on those actions, is transformed into a final, acceptable plan by a simple process of expanding the plan as instances of actions are chosen. As this process takes place, the *Goal Structure* [4] of the plan is monitored to ensure that there are no violations of protected intervals over which some condition has to hold. It

[4] Goal Structure is sometimes called the teleology of the plan – the effects of actions and their relationship to the satisfaction of conditions elsewhere in the plan.

is only when such a protection violation ([44]) occurs that the planner considers a different Goal Structure Approach to the problem (hence opening the search space by using a novel teleological approach). Earlier work (see the work in [45] which was generalised for planners with a least commitment representation of action ordering in [47]) has shown that there are at most two new teleological approaches that can be proposed to correct for any single interaction, specifically the odering of the violating condition *before* or *after* the protected interval. In practice, multiple interactions and other constraints in realistic plans can further constrain the number of approaches that are proposed.

O-Plan searches for a final acceptable plan, by using the initial Goal Structure Approach it is provided with in its initial partial plan, and will only increase the size of the search space as protection violations are found whilst using this approach. In this way we class the search as being *teleologically driven.*

4.3 Pruning Techniques

O-Plan incorporates a number of mechanisms, some novel and some derived from Operational Research (OR) or other AI planners, in order to restrict the search space the planner needs to consider. The various mechanisms are described below.

4.3.1 Condition Typing

One main form of search reduction in O-Plan is through the use of condition typing, also used in [47, 52, 53]. This technique allows domain knowledge to be used to prune the search. It is fed into the system via the TF domain description language. The responsibility for engaging this pruning therefore falls on the user which may or may not be a bad thing. What sets this technique aside from the other techniques and heuristics outlined below however is that it necessarily leads to loss of completeness in the abstract search space since the domain writer takes the responsibility for a deliberate pruning of the space. Condition typing can be very successful but there is work to be done on how far this technique can be developed. It is precisely the demands of this technique that caused us to adopt the term *knowledge based planning* to describe our work. In practice condition typing is essential on realistic problems in order to reduce search spaces to a manageable level, and this can be done effectively by a domain writer providing instructions to the system about how to satisfy and maintain conditions required in the plan.

4.3.2 Time

At any stage of planning, an activity is represented by a node in the plan state. Each activity has distinct start and finish instants stored with the activity node in the form of *[earliest time, latest time]* windows. Thus each activity has an earliest start time, latest start time, earliest finish time, and latest finish time. A time window management algorithm has been developed [6] and is used incrementally (whenever the plan network is altered) to revise information contained in time windows. In this revision process, overall consistency of temporal constraints is checked. If the constraints are mutually inconsistent then the algorithm signals this condition and the corresponding plan state is poisoned (abandoned), resulting in an effective pruning of the search. These algorithms have extended those in Deviser [52] and have provable termination criteria in contrast to the algorithms in Deviser.

The time window approach accepts a lack of precise knowledge about the timing of instants in the plan. No attempt is made explicitly to represent probabilistic uncertainty. However, probabilistic uncertainty might well be one reason causing the planner to have imprecise knowledge on the timing of an instant.

In specifying imprecise knowledge, the user is free to supply any range *[l,u]* for the duration of an activity or the required delay between an activity and one of its immediate successors in the plan. l must be non-negative and u must be greater than or equal to l. Thus, *[0,∞]* represents complete lack of knowledge. In general, specification of narrow ranges imposes tighter temporal constraints and enhances the possibility of pruning the search space due to mutually unsatisfiable temporal constraints. Thus, for example, it may be helpful to impose a realistic deadline on the finish time of the overall plan rather than to specify an upper bound of ∞. Analogous comments apply to specifying finish times of subplans represented by action descriptions.

The temporal representation in [6] assumes that an activity is represented by two time points: its start instant and finish instant. A range on the duration of an activity or on the elapsed time between the finish of one activity and the start of another is represented by a pair of arcs. Metric time constraints on the start or finish instant of an activity are represented by a pair of arcs between some em start-of-all-time point and the instant. The plan network orderings explicitly represents relative temporal constraints, each represented by one arc. Using standard network longest path algorithms, one can then discover constraints which are implicitly represented. A longest path of length *lij* between instant i and instant j is equivalent to a direct arc of length *lij* between the same two nodes. Most constraints are computed on demand. Explicitly represented constraints make up a sparse network. The time window information is sufficient to signal mutual unsatisfiability of temporal constraints when this is present. In O-Plan this start and finish time information is attached to a single action node to allow for quicker access to temporal information but this is equivalent to the uniform representation in [6].

The implementation of this time window management algorithm has highlighted the need for AI systems in general to be able to import and incorporate techniques developed in other disciplines. It can be proved that this algorithm operates in polynomial time and its efficiency is enhanced as it operates incrementally, triggered on any local change to the emerging network.

4.3.3 Resources

Three types of resource have been identified as being important during a plan generation process - consumable, renewable and substitutable resources. These resource types are distinct from time, which we do not treat as a resource in its own right.

Consumable Resources Consumable resources are those for which an initial stockpile is available which can only be depleted by actions in the plan. Our problem is one of representing and propagating resource consumption constraints in an efficient manner and guaranteeing that total usage does not exceed the initial availability. We will consider a single resource, widgets, with an overall availability of u (a known quantity). The minimum overall usage of widgets must not fall below l (a known quantity, typically 0). Thus the overall usage of widgets must lie in the range $[l,u]$. In any particular plan state we have lower and upper bounds on the number of widgets used by any action. Our plan will be valid with respect to the widget resource if it is possible to select actual usages of widgets for every action from within that action's known bounds such that the total usage in the plan lies in $[l,u]$. Because plan states can only be further constrained as the planning process proceeds, we must insist that the *[lower bound, upper bound]* widget usage range for action a must apply to the overall usage of widgets in a more detailed subplan which replaces action a.

For each action in the plan, a *[lower bound, upper bound]* range on widget usage for that action is maintained. Our algorithm maintains consistency between all such ranges. If one such range should shrink because the planner is somehow made capable of making a more precise statement of resource usage for a particular action, then this constraint is propagated throughout the plan. Such propagation may cause widget usage ranges to shrink for other actions.

Widget usage constraint consistency is maintained as follows: assume that the current plan state is valid with respect to widget usage and that each node has an attached widget usage window. These windows reflect all known widget usage constraints on the plan as it stands. Now assume that an additional constraint is imposed, i.e. the lower or upper bound of widget usage at some node of the plan is altered so that that node's resource window shrinks. There are 4 conditions which must be maintained (or reachieved) through constraint propagation. These conditions relate the resource window at a particular node n with resource windows at node n's parent, children, and siblings. Maintenance of these conditions thus involves propagation of constraints throughout the hierarchy of nodes. The resource propagation algorithm computes and maintains min-max pair information for the use of a single consumable resource. [5] gives precise details of the algorithms used.

There is value in having bounds as narrow as possible at any node. Narrower bounds represent tighter constraints; if each constraint introduced is as tight as reasonably possible, then the possibility of discovering that all constraints are mutually unsatisfiable is enhanced. This is helpful in pruning the overall search space. Thus there might well be later benefit in immediate investment of computational resources to compute relatively tight bounds rather than to simply use a default value like $[0,\infty]$. Liberally wide min-max intervals are of limited value in pruning the search space.

Shared Resources with Unit Availability A shared resource with unit availability is not consumed but can be used by only one agent at a time. Despite an availability of 1, maintaining constraints on usage of this resource is far more difficult than maintaining constraints on a strictly consumable resource. Usage of a shared resource must be scheduled; scheduling problems are potentially combinatorial because of the number of schedule permutations and may require search to satisfy constraints on such uasge. We see little distinction between the problem of scheduling a shared resource with unit availability and maintaining any other logical condition (e.g. `(on a b) = true`) in a plan. At any instant the gadget may be either in use or not in use. Thus a precondition to allocating the gadget to an agent is that "in use gadget" is false, an immediate effect of this allocation is that "in use gadget" is made true, and an immediate effect of returning the gadget to the resource pool is that "in use gadget" is made false. No special purpose representation is proposed for shared resources with unit availability. Normal effects and conditions can be used to reason about such a resource.

Shared Resources with Availability Greater than 1 Given our observation that there is no essential difference between maintaining constraints on a resource with unit availability and maintaining any other logical condition, one might be tempted to decompose a resource with availability $r > 1$ into r distinctly named resources with availability 1. Unfortunately, the resulting r resources will be substitutable and a planner could expend much useless effort trying to satisfy unsatisfiable resource constraints by trying other permutations of the usage of such substitutable resources. For example, assume that $r = 10$, that the planner has allocated units 1 through 10 to actions $a_1, \ldots, a_{10}$ in parallel, and that the planner discovers that action k also in parallel requires one unit of this resource. The planner correctly diagnoses the conflict that 11 units are required when only 10 are available. In attempting to correct the situation, the planner might then try each of 10! possible reallocations of units 1 through 10 to actions $a_1, \ldots, a_{10}$. Enough said!

It then appears essential that we model a controller of the usage of any particular resource. A plan would be valid with respect to that resource if it contained a schedule of activities which met all temporal constraints required in our temporal management algorithm and also obeyed the resource constraints imposed by the controller of that resource. Resource smoothing (the management of shared resources) is an inherently intractable problem. A least-commitment approach to planning would insist that we could reassure ourselves that a plan exists which is valid with respect to every resource while respecting temporal constraints. This is a notorious problem. Although it is possible to construct simple heuristics for resource smoothing, these heuristics are not guaranteed to find a feasible solution. Thus these heuristics cannot be used if we wish to ensure completeness of the search.

Our problem differs from traditional sequencing and scheduling problems which seek to minimize such objective functions as project makespan, weighted tardiness of jobs, etc., subject to constraints on the flow of jobs through a shop, utilization of machines, etc. The primary differences and analogies are:

- our objective is simply to verify the existence of a feasible schedule (and to construct it when planning is completed),
- some of our resource constraints are analogous to those in traditional job shop sequencing and scheduling. A shared resource with availability n is analogous to n identical machines in parallel,
- many temporal constraints that result from one action's effects being a precondition to a later action do not have analogues in traditional scheduling and sequencing.
- strictly consumable resources and shared resources are not often handled within the same scheduling and sequencing model (although both are common in different models).
- sequencing and scheduling models do not provide for the possibility of resources which are renewable in the sense that some activities may be inserted into the plan which have the effect of increasing the resource's availability. Such resources are discussed briefly below.

The first two items should work toward making our task easier than that of traditional scheduling. However the rest make the typical AI planning task considerably more complicated than scheduling problems which can be handled successfully by techniques reviewed in [4] or other more recent OR texts. To speculate a little, it is reasonable to question whether present domain-independent AI planners can cope with realistic problems that have a heavy sequencing component. Such planners do well in environments where their task of plan synthesis involves primarily finding appropriate schemata to satisfy particular goals or expand particular actions, and their scheduling activity is mainly the correction of interactions. In this case scheduling takes the form of adding to an existing set of temporal constraints which can be represented without resorting to disjunction. Sequencing, on the other hand, requires extensive use of disjunction in representing possible orderings of activities thus implying lots of search. A natural area for further research is to investigate the appropriateness of known sequencing and scheduling algorithms or more flexible disjunctive plan representations within a domain-independent AI planning framework. We are considering such approaches.

Renewable Resources Resource smoothing problems are further complicated by the introduction of renewable resources. This provides a mechanism for the possible substitutability of resources, a concept which is difficult to analyze in a traditional mathematical model. With renewable resources it may be possible to consume resource1 in a task whose effect is to produce additional units of resource2, e.g. money may be converted to fuel. Thus resource1 and resource2 become substitutable in the sense that a valid plan may be constructed out of various combinations of initial availabilities for the two resources. If resource1 and resource2 are both consumable rather than shared then it may be possible to represent constraints on their overall usage. Such constraints would be standard linear programming constraints provided that the process of producing one resource from some combination of other resources had a linear production function. However, if either resource1 or resource2 is a shared resource, the picture is considerably more complicated. Since handling shared renewable resources is inherently more complicated than handling the shared resources mentioned above, this type of resource usage modelling has not yet been approached by us.

4.3.4 Temporal Coherence

There may be constraints on legal states of a world model; some combination of facts may not be able to hold simultaneously in a physical state. Sets of such inconsistent facts have been referred to as domain constraints. Temporal coherence is our method of using such constraints to reduce search. The technique is fully described in [22]. This work was motivated by the difficulties of plan repair and dependency recording and by the failure of some earlier AI planners to generate some obvious plans in the blocks world. It was further motivated by a gap in the work of Chapman [8]. Chapman provided the Modal Truth Criterion (MTC) as a statement of the conditions under which an assertion will be true at a point in a partially ordered plan. Essentially, the MTC says that an assertion p is necessarily true at a point in a plan if and only if 1) there is a point necessarily before the required point where p is necessarily asserted; and 2) for every operator that could possibly come between the point of assertion and point of requirement, if the intervening operator possibly retracts an assertion which might turn out to be p, then there must be another (appropriately placed) operator which restores the truth of p whenever the intervening operator deletes it.

One can take a procedural reading of the MTC to produce a non-deterministic goal achievement procedure (e.g. as in Chapman's TWEAK). Such a procedure will form the heart of any correct "Nonlinear" planner. TWEAK searches a space of partial plans, using the goal achievement procedure as a partial plan generator and in TWEAK the space is explored breadth-first. However planning systems such as Nonlin, SIPE, Deviser and O-Plan strive for realism and so cannot afford this luxury of breadth-first search. Heuristics for selecting among the plan modification operations sanctioned by the MTC are required if plans are to be produced in acceptable time.

The MTC says nothing about an order in which to pursue goals. Possible bindings are determined by goal-ordering, so the MTC gives no guidance regarding sensible bindings for un-bound variables. The heuristic of temporal coherence addresses this problem. It suggests avoiding work on plans whose *bulk preconditions* do not "make sense". The bulk preconditions for a plan are the overall conditions on which the plan depends for its successful execution. We say that these preconditions do not make sense if they do not describe a physically realisable domain state. If a plan's bulk preconditions do not make sense, then the plan has internal inconsistencies, and is best avoided.

The basic principle of temporal coherence is this: *prefer not to work on partial plans which have inconsistent bulk preconditions*. Think of this as follows. At each point in its search a planner will have a partially completed plan. The search begins with a given, or "root" plan, and each partial plan uncovered in the search will differ from the root plan by the addition of some number of operator schemata and binding of variables. Each added operator schema will have preconditions. Unless the plan is complete, flawless down to the truth of each and every precondition, there will be at least one precondition of at least one operator in the plan which is not true by the MTC.

Each assertion (precondition) which *is* true will either be true by some added operator, or true from the initial situation. We are interested in analyzing those assertions which must be true if the partial plan developed so far is to be "executable"; these assertions are the bulk preconditions for the developed plan.

Temporal coherence suggests working on those plans whose bulk preconditions describe a physically possible state of the given planning domain. By "possible" here we mean consistent with certain prespecified physical laws. Suppose that a plan's bulk preconditions do not describe a possible domain state. Why would this happen? It would happen only if the operators in the plan were not "causally independent" of each other, and required further sequencing to form a valid plan. Future goal achievements allowed by the MTC might well do this, but when faced with the choice between a plan which *already* requires a valid state of the world for its execution, and one which does not, it makes sense to choose the former. If possible it is best to avoid plans which require impossible initial states and the corrective work they entail. This avoidance of temporary impossibilities appears to be a good search heuristic. In our experience, this can lead to significant time savings in plan construction.

Temporal coherence is currently implemented in O-Plan in the way described above though we believe it to have much more potential [17]. The basic problem of ordering is a crucial one and goes beyond goal ordering. The ordering problems associated with agenda handling in general also need to be tackled. We have made a start. Full details of the work on Temporal Coherence is given in [22].

4.4 Search Order

Temporal coherence only addresses goal ordering, but the mere fact that there are many different types of choice open to a planning system means that other approaches to search control are necessary. In O-Plan the agendas are again used to record choice left open to the system at any point during the generation of a plan. Agenda records are assigned a priority rating when first added to the agenda lists. This is done in a fairly simple (too simple!) way at the moment as this calculated priority remains with the individual record until such time as it is picked up by a knowledge source for processing. Ideally AI planners need to operate much more opportunistically and be able to select the "next best" agenda record every time.

However O-Plan does calculate these priorities in an informed manner. The agenda record *type* (including a particular `condition` type) says much about which order any particular "flaw" (remember agenda records map onto plan flaws) will be worked on. For instance an `unsupervised` condition has a low priority and should be able to be tidied up after all necessary additions to the plan have been included. An `achievable` condition type however has a higher rating as this has a greater effect on the plan and may involve the generation of other flaw types in turn. At the top level, the `user` may interrupt the system via the generation of the highest priority agenda record. O-Plan also uses a `sequence` agenda record type to logically group together a collection of related agenda records in a way that is presequenced by some knowledge source (i.e. the meta planning level of O-Plan). This seems to be advisable in order to embed programmer knowledge on how to process a comples flaw, but this may alse be handled by an agenda controller with sufficient information to allow it to dynamically focus its attention on the current demands.

A finer level of priority is achieved by maintaining measures of specificity of each agenda record. For instance if there is more than one way to handle a record then this should be rated lower than a record which has a single means of processing as the effects of processing this latter record will definitely appear in the final plan. For example an `expand` (a node) record with only a single matching schema will have a higher priority than a similar record with choice of schema for expansion. We go further than this and also specify a measure of specificity of each record at its "lowest level". What do we mean here? Consider the process of Question-Answering (QA), which asks if a proposition has a particular value at a particular point in the non-linear plan. The answer returned may be an unqualified *yes* or *no*, or it could be a *maybe* in which case this answer will be qualified with a potential tree of suggested node-to-node linkings and plan variable bindings to make in the plan in order to satisfy the truth of the proposition. QA is the process which detects and begins to correct for interactions in the plan. Each branch of this possibility tree represents choice in the search space, and this is captured in an `or` record. The specificity of this record is detailed in the measure of choice at the top of the tree (possibly none) and with an estimate of the number of leaf nodes at the bottom, each representing a possible solution to the QA process. We call these *branch1* and *branchN* estimators. O-Plan will choose to work on the most fully determined records first in order to constrain the search space.

With these measures and with the pruning techniques, such as temporal coherence, in mind O-Plan progresses its search in a local best, then global best manner. It is an aim however to improve the local best component of the system even further.

5 The User

The O-Plan project includes work on the design and implementation of a planning workstation, which supports various styles of interaction with a user. The user may have various *roles* with respect to the AI planner - provider of domain descriptions, requestor of plans, seeker of information on plans, developer and debugger of the O-Plan system, assistant to the planner's search control, etc. Our interfaces have tried to identify clearly the different roles being played by the user with respect to the system in order to present appropriate information and options. Early work was implemented in Pascal on a Perq computer [11] and was eventually reworked for the Sun workstation. O-Plan currently runs on a Sun, using the Poplog environment. New work is underway to provide a Common Lisp implementation.

The O-Plan system is intended to operate with concurrent graphical plan output and plan simulation facilities. There would normally be three "windows" active. The planner would be running in one window, a plan network drawer would be active in a second and would allow the user to view partial or complete plans both during and at the end of planning. A third window would provide a view of the state of the world at some point in the plan that interested the user. A sample display of the O-Plan user interface showing these three windows for O-Plan running on a block stacking problem is show in figure 4.

A Graph Drawing package translates the partially ordered plan produced by O-Plan into a graphic display. It allows various node representations (dots, boxes, annotated boxes, etc), and this includes the ability to specify, through the TF, an iconic description of the activity at the node. The graphs in figure 5 and figure 6 demonstrate plans drawn in standard box mode or in iconic mode. Note that in the second graph the drawing is bigger than the window size. The Graph Drawer allows the graph to be panned by use of the viewer box at the top left of the window, or by use of the scroll bars.

The Graph Drawer supports user interaction through the workstation "mouse". Pointing and clicking the mouse at a node is interepreted by O-Plan as a request to view all planning information relevant up to that point in the plan. Much of this information can only be presented textually (such as the Goal Structure) but the "context" (the aggregate of the effects up to that point in the plan as found by the QA procedure) can often be graphically displayed given a suitable domain state drawer. O-Plan allows a domain *dependent* display process to be specified in the application TF which then takes all responsibility for the interpretation of the effects and their presentation to the user. This can be highly effective. A trivial example is shown in figure 4 for the blocks world and in figure 7 for a more realistic example, in this case the construction of a space platform. After plan generation, user interaction is at a browsing level only. This can be extended to a "simulation" of the plan by flicking through the graphical representation of all states up to the point of the plan of interest. We have developed the interface and context drawing software for the blocks world, for a (real) spacecraft at the level of its wiring harness, and for a space platform construction task.

During plan generation the user has full access to all information in the system, including agendas, partial order, Goal Structure, etc. Access is gained through a request to schedule a user agenda entry. The `user` knowledge source provides a menu of ways in which the user can act to control search or view the emerging plan. The user can therefore elect to exercise any form of control on the system. This is only suited to the developer or extender of the O-Plan system itself at present since the internal information and control procedures within the system become too complex for manual control.

The other form of user interaction is through TF. As has been discussed earlier, Task Formalism can be difficult to use directly for complex domain descriptions. Front end user interface support would be required in a realistic planning system which would automatically generate the revelant TF. The design of the Task Formalism is geared towards this. TF is evolving to cater for new features of new domains. Some experiments with the use of a requirements analysis methodology to assist the domain describer in the generation of TF have been conducted [55].

The O-Plan project has also been concerned with the interfacing of an AI Planner to a Natural Language system for command and control. First results are described in [9]. User interface work undertaken recently has explored the use of a readily available Computer Aided Design package (AutoCAD, [3]) as an interface for operator schema and task input, plan output, plan browsing and domain specific simulation of the context at points in the plan. The output graphs can be generated quickly in this interface and can look natural for small plans. However, they do not replace the need for a precise graph drawing capability in which all ordering links can be unambiguously followed - this continues to be provided by the main Graph Drawer described earlier. The AutoCAD interface has been demonstrated on house building and space platform assembly tasks [51]. A sample screen image from this interface is shown in figure 7.

6 Applications

O-Plan has been used on a variety of applications, with varying degrees of success. The domains have been chosen to test the various capabilities of the system, and are representative of the type of problem we would expect AI Planners to be tackling. Characteristics include action expansion, condition achievement, interaction detection and correction, temporal and consumable resource management, and management of alternatives. Typical plans generated have been of the order 50-70 nodes in size, and have been generated autonomously from specification right through to plan visualisation (graph drawing). The larger plans take a few minutes of workstation CPU time, typically 3-4 minutes, to generate.

The applications include:

- Block World. Many AI planners are tested against the conceptual problems arising in the many flavours of block problems. The simplicity of the domain enables problem scenarios to be isolated and bounded, and described easily. O-Plan has been extensively tested against problems set in this domain. However, O-Plan is not particularly suited to "puzzles" of this kind. It performs best when realistic constraints restrict search in the domain.
- House Building Project Planning. Again a "standard" domain used previously with Nonlin, this application allows useful benchmarking and various scenarios to be created and tested. In particular the domain can include examples of temporal and resource constraints, and can exploit the use of typed preconditions.
- Oil Platform Construction Project Management. This example from [29] is a simple application, but it exhibits features of real interest. Several different ground conditions on which an

oil rig can be constructed leads to suitable problems for exploring disjunctions in plans and disjunctive QA for this problem [37] [38].

- UOSAT-II Scheduling. This problem involved a real, orbiting spacecraft used in a parallel AI in Space study [54], [26], and was chosen as an exercise in coding domains in TF and for exploring the simulation user interface to O-Plan using a display of the state of the wiring harness of the spacecraft. The task of O-Plan is to generate mission sequences to produce the weekly "diaries" relating to the various activities demanded of the craft. This particular application showed the problems of adopting a simple least commitment planning philosophy and the associated problems of plan variable binding. Improved integration of scheduling and constraint reasoning methods are required for a satisfactory solution to the problems in this domain. See [13] for details.

- Space Platform Construction. Nonlin and O-Plan have been applied to the generation of a range of different configurations of space platform made up on joints, trusses, living and working modules, equipment pallets, antennae, solar pannels and radiators. This application was used to explore the use of a widely available Computer Aided Design package (AutoCAD) as a tool for task description, plan output, plan browsing and plan simulation [51].

- PLANIT. The UK Alvey Programme PLANIT Collaborative Club provided data from a Price-Waterhouse software project management application, relating to the control of a project to develop an information processing system. This data was used to generate activity plans for subsequent browsing and manipulation. Both Nonlin and O-Plan were used to generate the required plans. Simplification were made to the real problem description in the area of resource modelling in order to cope with the application on the current O-Plan prototype. The domain provided the opportunity to apply TF to a real problem description, and also the opportunity to generate "knowledge-rich" plans for post-processing and analysis. The generated plans, and the underlying plan representation concepts were used as the basis for the design of the full PLANIT Club IPA, the Interactive Planning Assistant [2].

7 Conclusions

The O-Plan architecture and planning system is a move towards fulfilling the needs of a practical planning system. However as it stands it is not robust or complete enough to justifiably meet that claim. The main features of O-Plan are its overall structure, based around a central plan state data structure and agenda based reasoning, and its concentration on attacking the central problem of planning - search control. In exploring issues of search, we have have extended previous ideas on Goal Structure and condition typing, and heuristics based on the use of time and resources in a planning domain. Our temporal coherence heuristic has focussed attention on the development of an approach to the problem of "goal ordering". These heuristics make some progress in achieving realistic application of AI planning methods though there is still research effort required to advance resource management, condition typing, and the use of domain information (temporal coherence) as a means of controlling search. The use of effective search space pruning methods will not eliminate the need for proper plan repair schemes (as opposed to elaborate backtracking schemes) but will minimise the need for such schemes during plan generation.

O-Plan touches on the need to integrate resource reasoning to support a mix of planning and scheduling. This is seen as essential for resource based, planning domains, and also to extend the features of AI planning to the world of scheduling.

A range of different applications have been run on O-Plan to demonstrate and test various features. Experimentation with user interfaces for AI Planners has taken place.

Future work will provide a portable version of the O-Plan prototype and this work is now underway. Work is also ongoing to develop the dependency recording schemes which can be used for plan repair. The O-Plan system is intended to cope with plan execution and control as well as plan generation. This implies that the system should be able to operate in an incremental fashion with the possibility of adjustment of the goals being worked on and the environment in which the plans are intended to operate. Initial work by us on such an incremental version of O-Plan is described by Tate [50].

8 Acknowledgements

Thanks go to our colleagues on the O-Plan project - Colin Bell, Roberto Desimone, Brian Drabble, Mark Drummond, Anja Haman, Richard Kirby and Judith Secker.

O-Plan was funded by the UK Alvey Programme through the Science and Engineering Research Council on grant number GR/D/58987 (Alvey Directorate project number IKBS/151) and supported by a fellowship from SD-Scicon. Recent work was partially supported by the Artificial Intelligence Applications Institute and the US Air Force/European Office of Aerospace Research and Development by grant number EOARD/88-0044 monitored by Dr Nort Fowler at Rome Laboratory. The views expressed are those of the authors only.

References

[1] Allen, J., Hendler, J. & Tate, A. Readings in Planning. *Morgan-Kaufmann, 1990.*

[2] Alvey Directorate. Alvey Grand Meeting of Community Clubs. *Available through Institution of Electrical Engineers (*IEEE*), London, 1987.*

[3] AutoDESK, AUTOCAD Reference Manual, *AutoDESK Ltd, London, 1990.*

[4] Baker, K. An introduction to sequencing and scheduling. *Wiley, 1974.*

[5] Bell, C.E., Currie, K.W. & Tate, A. Managing scheduling and resource usage constraints in O-Plan. *Alvey Planning* SIG*, No 4, Sunningdale,* UK*. Also Artificial Intelligence Applications Institute* AIAI-TR-6 *1986.*

[6] Bell, C.E. & Tate, A. Using temporal constraints to restrict search in a planner. *Artificial Intelligence Applications Institute* AIAI-TR-5 *1986.*

[7] Bundy, A. The computer modelling of mathematical models. *Academic Press, 1983.*

[8] Chapman, D. Nonlinear Planning: A rigourous reconstruction. *In procs. of the Ninth International Conference on Artificial Intelligence, Los Angeles, 1985.*

[9] Crabtree, B., Crouch, R.S., Moffat, D.C., Pirie, N., Pulman, S.G., Ritchie, G.D. and Tate, A. A Natural Language Interface to an Intelligent Planning System. *Proceedings of the* UK *Information Engineering Directorate IT Conference, Swansea, July 1988. Also available as Department of AI Research Paper No. 407, University of Edinburgh.*

[10] Currie, K.W. & Drummond, M. A case for preferential ordering. *Petri Net newsletter No. 27, 1987.*

[11] Currie, K.W. & Tate, A. The O-Plan Task Formalism Workstation. *Alvey Planning* SIG, *No 4, Sunningdale,* UK. *Also Artificial Intelligence Applications Institute* AIAI-TR-7, *1984.*

[12] Currie, K.W. & Tate. A. O-Plan: control in the open planning architecture. *In proc. of the* BCS *Expert Systems '85, Warwick,* UK, *Cambridge University Press, 1985.*

[13] Currie, K.W., Drummond, M. & Tate, A., O-Plan meets T-SAT: first resuls from the application of an AI planner to spacecraft mission sequencing. *Final report to* SERC *on grant number* GR/E/05421. *Also Artificial Intelligence Applications Institute* AIAI-PR-27, *1988.*

[14] Daniels, L. Planning and Operations Research. *AI Tools, Techniques and Applications, eds. O'Shea & Eisenstadt, Harper & Row, 1984.*

[15] De Kleer, J. An assumption based truth maintenance system. *AI Journal Vol. 28, 1986.*

[16] Dean, T. Temporal imagery: an approach to reasoning about time for planning and problem solving. *Yale University, technical report no. 433, 1985.*

[17] Desimone, R. & Mallen, C. Complete, consistent goal sets: controlling the search for nonlinear plan generation. *In procs. First International Conference on Expert Planning Systems, Brighton,* UK, *1990.*

[18] Doyle, J. A truth maintenance system. *AI Journal Vol. 12, 1979.*

[19] Drummond, M. Plan nets: a formal representation of action and belief for automatic planning systems. *Ph.D. Dissertation, Dept. of Artificial Intelligence, University of Edinburgh, 1986.*

[20] Drummond, M.E., Currie, K.W., & A. Tate. Contingent plan structures for spacecraft. In *Proceedings of the JPL/NASA Workshop on Telerobotics, Pasadena, CA, 1987.*

[21] Drummond, M. & Currie, K.W. Exploiting temporal coherence in Nonlinear plan construction. *Computational Intelligence Journal, 4(4), 1988.*

[22] Drummond, M. & Currie, K.W. Goal ordering in partially ordered plans. *In procs.* IJCAI-89, *Detroit,* USA, *1989.*

[23] Drummond, M., Currie, K.W. & Tate, A. Contingent plan structures for a spacecraft. NASA *Workshop on Space Station Telerobotics,* JPL, *Pasadena, 1987.*

[24] Fikes, R.E., Hart, P.E. & Nilsson, N.J. *Learning and executing generalised robot plans, AI Journal Vol. 3, 1972.*

[25] Fikes, R. & Nilsson, N. STRIPS: a new approach of the application of theorem proving to problem solving. *Artificial Intelligence 2, pp. 189-208, 1971.*

[26] Fraser, J.L., Conway, S., et al. Using on-board AI to increase spacecraft autonomy. *International conference on human-machine interaction & and AI in Aeronautics and Space. Toulouse, France Sept. 1988.*

[27] Hayes, P.J. A representation for robot plans. *In proc. of* IJCAI-75*, 1975.*

[28] Lesser, V. & Erman, L. A retrospective view of the Hearsay-II architecture. *In procs. of* IJCAI-*77, pp. 27-35, 1977.*

[29] Levitt, R. & Kunz, J. Using knowledge of construction and project management for automated schedule updating. *Project Management Journal Vol* XVI *no. 5, 1985.*

[30] Liu, B. Reinforcement planning for resource allocation and constraint satisfaction. *Ph.D. thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1988.*

[31] McGregor, D., McInnes, S. & Henning, M. An Architecture for associative processing of large knowledge bases. In *The Computer Journal, Vol 30, No. 5, 1987.*

[32] Manna, Z. & Waldinger, R. A theory of plans. *In procs. of* CSLI/AAAI *Workshop on Planning and Action, Oregon, 1986.*

[33] McDermott, D.V. A Temporal Logic for Reasoning about Processes and Plans In *Cognitive Science, 6, pp 101-155, 1978.*

[34] Nii, P. The blackboard model of problem solving. *In AI Magazine Vol.7 No. 2 & 3. 1986.*

[35] Sacerdoti, E. Planning in a hierarchy of abstraction spaces. *AI Journal, Vol. 5, No. 2, pp.115-135, 1974.*

[36] Sacerdoti, E. A structure for plans and behaviours. *Artificial Intelligence series, publ. North Holland, 1977.*

[37] Secker, J. Use of O-Plan for oil platform construction project planning. /em Artificial Intelligence Applications Institute AIAI-PR-22, 1988.

[38] Secker, J. A solution to the disjunctive planning problem. *In procs. Eighth Alvet Planning SIG, Nottingham,* UK. *Also Artificial Intelligence Applications Institute* AIAI-TR-64*, 1989.*

[39] Smith, S., Fox, M. & Ow, P.S. Constructing and maintaining detailed production plans: Investigations into the development of knowledge based factory scheduling systems. *AI Magazine Vol. 7 No. 4, 1986.*

[40] Sridharan, N. Practical Planning Systems, *Rochester Planning Workshop,* AFOSR*, 1988.*

[41] Stefik, M. Planning with constraints. *In* Artificial Intelligence*, Vol. 16, pp. 111-140. 1981.*

[42] Sterling, L., Bundy, A., Byrd, L., O'Keefe, R. and Silver, B. Solving Symbolic Equations with PRESS, *In Computer Algebra, Lecture Notes in Computer Science No. 144., Edited by Calmet, J., (Springer Verlag).*

[43] Suen, D. An implementation of a distributed planning system in PARLOG. *M.Sc Thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1988.*

[44] Sussman, G.J. A computational model of skill aquisition. *MIT AI Laboratory* TR-297*, 1973.*

[45] Tate, A. Interacting goals and their use. *In proc. of* IJCAI-75 *1975.*

[46] Tate, A. Project planning using a hierarchical non-linear planner. *Department of Artificial Intelligence Memo No. 25, University of Edinburgh, 1976.*

[47] Tate, A. Generating project networks. *In procs.* IJCAI-77*, 1977.*

[48] Tate, A. Functions in context database. *Second Alvey workshop on Large Knowledge Base Architectures, Manchester,* UK*, July 1984. Also Artificial Intelligence Applications Institute* AIAI-TR-1*, 1984.*

[49] Tate, A. Goal Structure, Holding Periods and "Clouds". *In Reasoning about actions and plans, Morgan-Kaufmann, 1986.*

[50] Tate, A. Coordinating the activities of a planner and an execution agent. *In procs. of the Second NASA Conference on Space Telerobotics, G.Rodriguez), Pasadena, CA, 1989.*

[51] Tate, A. Interfacing a CAD system to an AI planner. *Paper to the* SERC *seminar on Integrating Knowledge-Based and Conventional Systems, Edinburgh, May 1990. Also Artificial Intelligence Applications Institute sc aiai-tr-76, 1990.*

[52] Vere, S. Planning in time: windows and durations for activities and goals. IEEE *Transactions on Pattern Analysis and Machine Intelligence Vol. 5, 1981.*

[53] Wilkins, D. Practical Planning. *Morgan Kaufman, 1988.*

[54] T-SAT: Concluding report on the technology satellite design study. *Rutherford Appleton Labs.* SERC *report* RAL*-88-033. March 1988.*

[55] Wilson, A.C.M. Information for planning. *MSc thesis, University of Edinburgh, 1984.*

Figure 1: O-Plan Overview

Figure 2: Planning Taxonomy

Figure 3: O-Plan Modules

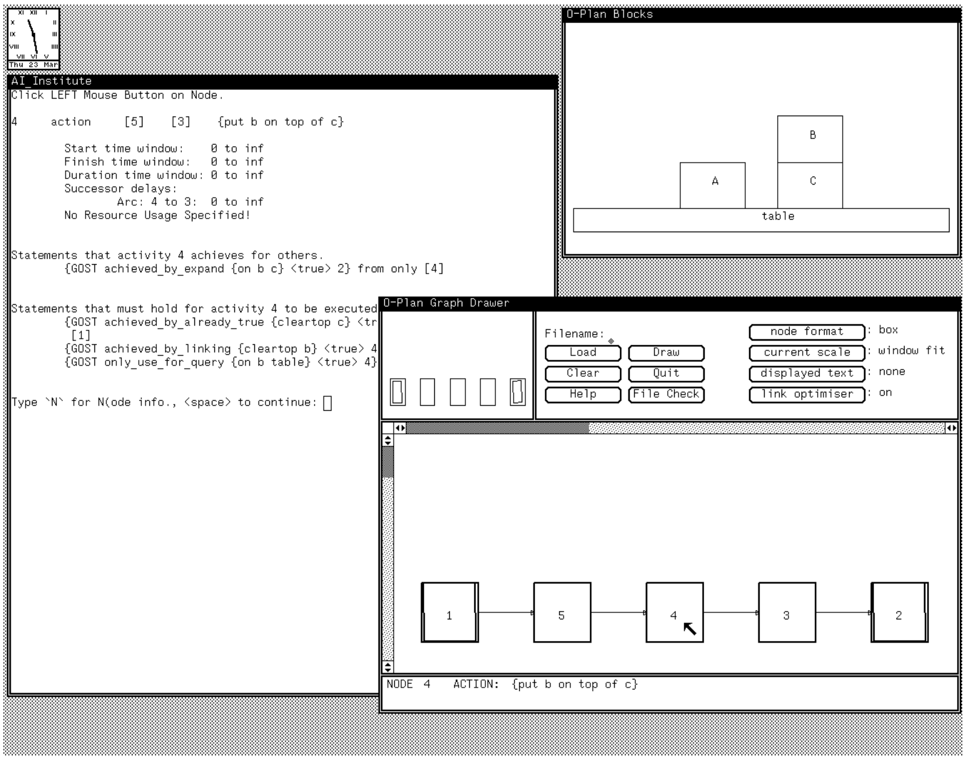


Figure 4: O-Plan Session

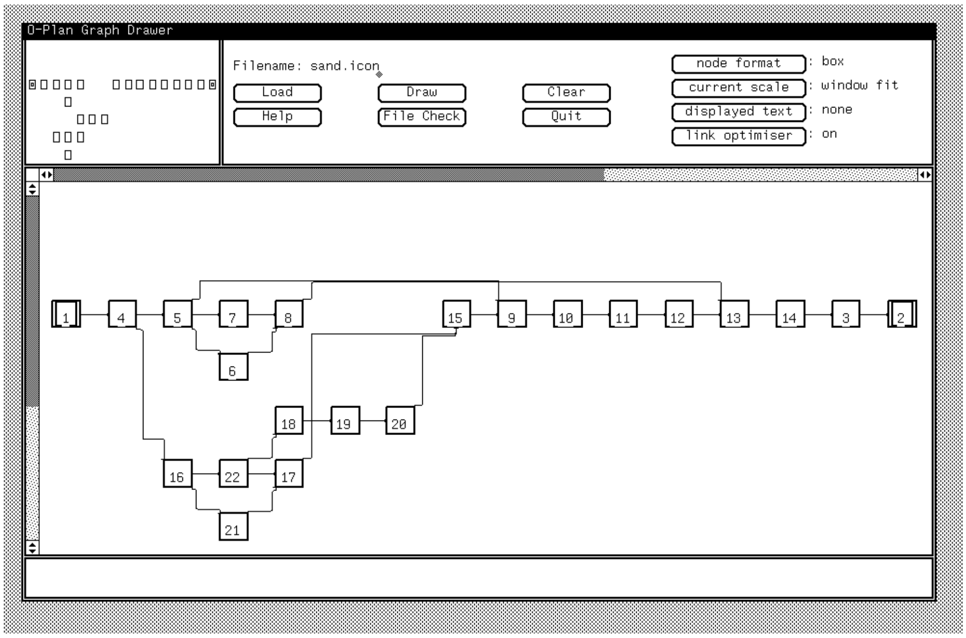


Figure 5: Standard Plan

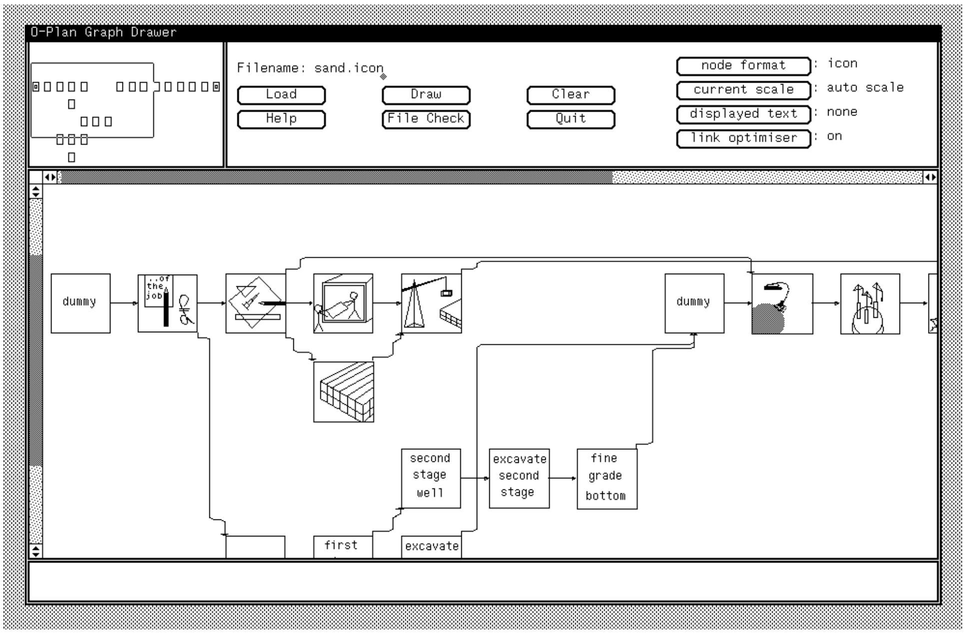


Figure 6: Iconic Display

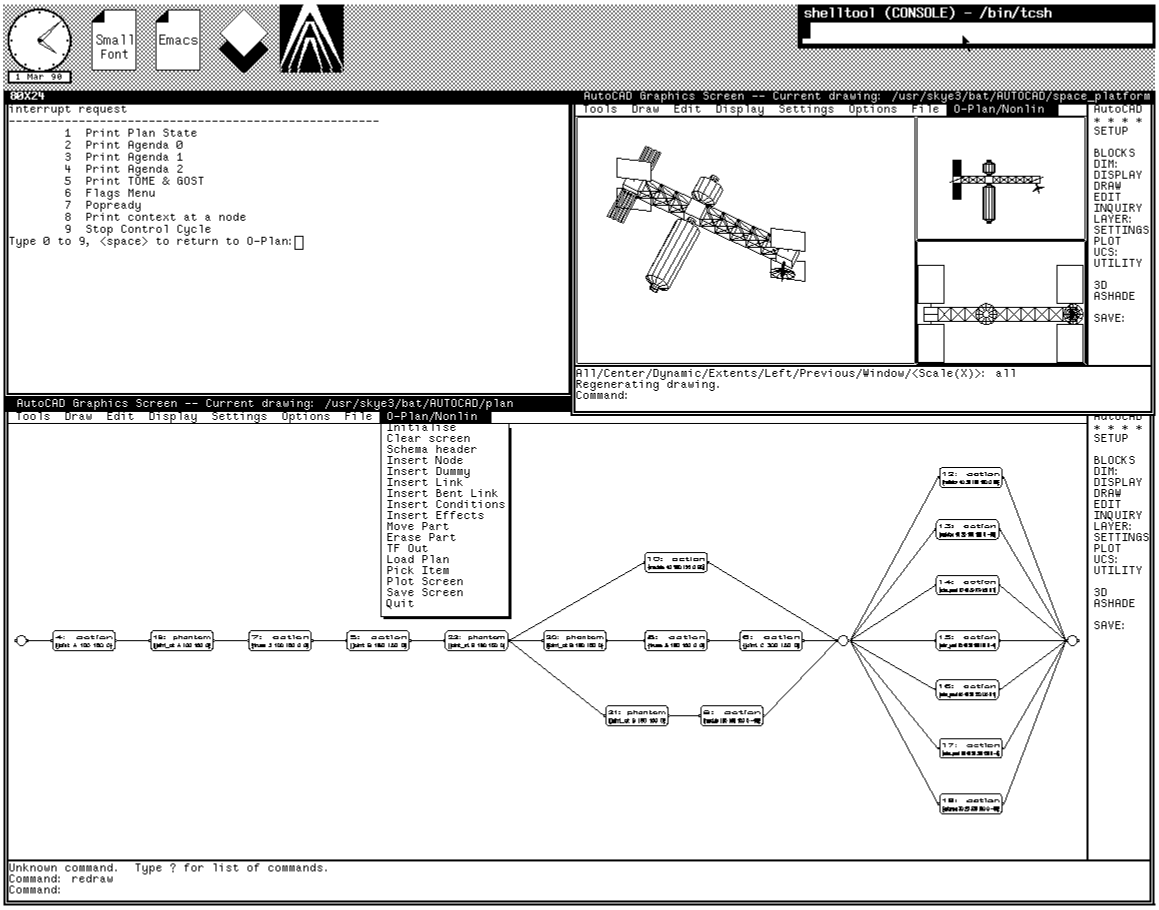


Figure 7: AutoCAD interface to O-Plan