O-Plan

Architecture Guide

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

July 12, 1995

Version 2.3

Acknowledgements

The O-Plan project began in 1984. Since that time the following people have participated: Colin Bell, Ken Currie, Jeff Dalton, Roberto Desimone, Brian Drabble, Mark Drummond, Anja Haman, Ken Johnson, Richard Kirby, Glen Reece, Arthur Seaton, Judith Secker, Austin Tate and Richard Tobin.

Prior to 1984, work on Interplan (1972–4) and Nonlin (1975–6) was funded by the UK Science and Engineering Research Council and provided technical input to the design of O-Plan.

From 1984 to 1988, the O-Plan project was funded by the UK Science and Engineering Research Council on grant numbers GR/C/59178 and GR/D/58987 (UK Alvey Programme project number IKBS/151). The work was also supported by a fellowship from SD-Scicon for Austin Tate from 1984 to 1985.

From 1989 to 1992, the O-Plan project was supported by the US Air Force Rome Laboratory through the Air Force Office of Scientific Research (AFOSR) and their European Office of Aerospace Research and Development by contract number F49620-89-C-0081 (EOARD/88-0044) monitored by Northrup Fowler III at the USAF Rome Laboratory.

From 1992 to 1995, the O-Plan project was supported by the ARPA/Rome Laboratory Knowledge Based Planning and Scheduling Initiative through the US Air Force Rome Laboratory through the Air Force Office of Scientific Research (AFOSR) and their European Office of Aerospace Research and Development by contract number F49620-92-C-0042 (EOARD/92-0001) monitored by Northrup Fowler III at the USAF Rome Laboratory.

Additional resources for the O-Plan and O-Plan projects have been provided by the Artificial Intelligence Applications Institute through the EUROPA (Edinburgh University Research on Planning Architectures) institute development project.

From 1989 to 1993, research on scheduling applications of the O-Plan architecture was funded by Hitachi Europe Ltd. From 1989 to 1992, the UK Science and Engineering Research Council (grant number GR/F36545 – UK Information Engineering Directorate project number IED 4/1/1320) funded a collaborative project with ICL, Imperial College and other partners in which the O-Plan architecture was used to guide the design and development of a planner with a flexible temporal logic representation of the plan state. A number of other research and development contracts placed with AIAI have led to research progress on the O-Plan prototype.

Contact Information

The O-Plan project team can be contacted as follows:

O-Plan Team
Artificial Intelligence Applications Institute
The University of Edinburgh
80, South Bridge
Edinburgh EH1 1HN
United Kingdom

Tel: (+44) 131 650 2732
Fax: (+44) 131 650 6513
Email: oplan@ed.ac.uk

Created:	December 20, 1991 by Austin Tate and Brian Drabble
Last Modified:	July 12, 1995 (14:52) by Brian Drabble
Printed:	August 14, 1995

Contents

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. O-Plan is intended to be a domain-independent general planning and control framework with the ability to embed detailed knowledge of the domain.

O-Plan grew out of the experiences of other research into AI planning, particularly with Nonlin [35] and "blackboard" systems [25]. The *Readings in Planning* volume [1] includes a taxonomy of earlier planning systems 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. The same volume [1] includes 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 in the NOAH planner [29]), though O-Plan is flexible concerning the order in which parts of the plan at different levels are expanded.
- An agenda-based control architecture in which each control cycle can post processing requirements during plan generation. These processing requirements are then picked up from the agenda and processed by appropriate handlers (HEARSAY-II [21] and OPM [20] 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 [24].
- 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 [42], 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.
- Goal Structure, Question Answering (QA) and typed preconditions. O-Plan is derived from the earlier Nonlin planner [35] from which the project has taken and extended these ideas.
- Domain and Task description language (Task Formalism or TF). The project has maintained Nonlin's style of and extended it for O-Plan.

1.1 Early O-Plan

The main effort on the first O-Plan project was concentrated in the area of plan generation. This early work is documented in a paper in the *Artificial Intelligence Journal* [10]. One theme of the early O-Plan research was search space control in an AI planner. The outputs of that work gave a better understanding of the requirements of planning methods, improved heuristics and techniques for search space control, and a demonstration system embodying the results in an appropriate framework and representational scheme.

O-Plan began with the objective of building an open architecture for an AI planning project with the objective of incrementally developing a system resilient to change. It was our aim at the start of the project to build a system in which it was possible to experiment with and integrate developing ideas. Further, the system was to be able to be tailored to suit particular applications.

1.2 O-Plan

The O-Plan project began in 1989 and had the following new objectives:

- to consider a simple "three agent" view of the environment to clarify thinking on the roles of the user(s), architecture and system. The three agents are the task assignment agent, the planning agent and the execution agent.
- to explore the thesis that communication of capabilities and information between the three agents could be in the form of *plan patches* which in their turn are in the same form as the domain information descriptions, the task description and the plan representation used within the planner and the other two agents.
- to investigate a single architecture that could support all three agent types and which could support different plan representations and agent capability descriptions to allow for work in task planning or resource scheduling.
- to clarify the functions of the components of a planning and control architecture.
- to draw on the early O-Plan experience and to improve on it especially with respect to flow of control [38].
- to provide an improved version of the O-Plan system suitable for use outside of Edinburgh within Common Lisp, X-Windows and UNIX.
- to provide a design suited to use on parallel processing systems in future.

The first O-Plan project at Edinburgh, 1984-1988, focussed on the techniques and technologies necessary to support the informed search processes needed to generate predictive plans for subsequent execution by some agent. The O-Plan project continues the emphasis placed on the design of a planning and control architecture, identifying the modular functionality, the roles of these modules, and their software interfaces. O-Plan has resulted in a demonstrator, capable of acting as a foundation for further development, in addition to descriptions of the underlying sub-systems and modules which we feel are important to support a practical planner.

O-Plan is incorporated within a blackboard-like framework; for efficiency reasons we have chosen an agenda-driven architecture. Items on the agenda represent outstanding tasks to be performed during the planning process, and they relate directly to the set of *flaws* identified as existing within the emerging plan. A simple example of a *flaw* is that of a condition awaiting satisfaction, or an action requiring refinement to a lower level. A controller chooses on each processing cycle which flaw to operate on next.

The nature of these flaw types has been influenced by experience from the first O-Plan work, but the main development focus is the handling and processing of the flaws. The "knowledge sources" employed in later versions of O-Plan have cleaner triggering mechanisms and have been given a variable level of granularity, enabling processing to be suspended if needed (we refer to this as knowledge source staging) while further flaw information is gathered. This is particularly useful for a planning system which attempts to be opportunistic and to operate on a least commitment basis, while retaining completeness of search (where possible). It will also simplify the task of maintaining and reasoning with partially bound variables in the plan, which proved to be difficult and limiting in the earlier O-Plan work.

Research in O-Plan has been concentrating on the problems associated with:

- Temporal constraints and reasoning. The underlying data structures have been completely re-designed and reworked from the work on the first version of O-Plan to allow further development of the temporal search based pruning algorithms, and to support the enhanced condition achievement procedure.
- Resource utilisation management. Resources provide the most obvious link to scheduling, where successes in resource utilisation management have been more pronounced, though still limited.
- Plan control. O-Plan is intended to communicate plans to an execution agent who can communicate progress back. Control strategies are therefore required to enable plans to be repaired in the case of simple failure or to begin replanning if required. Earlier work employing qualitative process [12] theory could be used to assist with repair strategies in future.

The end goal is to be able to demonstrate a domain independent AI Planner capable of accepting descriptions of planning domains and generating realistic plans for subsequent execution.

1.3 Characterisation of O-Plan

The O-Plan approach to command, planning, scheduling and control can be characterised as follows:

- successive refinement/repair of a complete but flawed plan or schedule
- least commitment approach
- using opportunistic selection of the focus of attention on each problem solving cycle
- building information incrementally in "constraint managers", e.g.,
  - object/variable manager

  - time point network manager
  - effect/condition manager
  - resource utilisation manager

- using localised search to explore alternatives where advisable
- with global alternative re-orientation where necessary.

O-Plan is aimed to be relevant to the following types of problems:

- project management for product introduction, systems engineering, construction, process flow for assembly, integration and verification, etc.
- planning and control of supply and distribution logistics.
- mission sequencing and control of space probes such as VOYAGER, ERS-1, etc.

These applications fit midway between the large scale manufacturing scheduling problems found in some industries (where there are often few inter-operation constraints) and the complex *puzzles* dealt with by very flexible logic based tools. However, the problems of this type represent an important class of industrial relevance.

1.4 Structure of the Architecture Guide

Having described the background to the O-Plan system, the Architecture Guide will now describe the O-Plan architecture by introducing the 5 major components in the architecture: Knowledge Sources and their computational Platforms; Domain Information; the Plan State; the Controller; and the Constraint Managers and Support Routines. These will be referred to throughout the Architecture Guide and greater detail of the various components are the subject of later sections.

The current O-Plan project has concentrated on the provision of a planning agent within the O-Plan architecture. A section of this guide shows the ways in which the 5 components of the architecture referred to above are developed to enable the system to perform as a planner. There are brief sections to describe the simple task assignment (command) and execution system agents which form a part of the current O-Plan prototype.

The User Interface to the O-Plan system has been designed in such a way that it will allow integration with a number of other sophisticated user tools. A section of the Architecture Guide therefore highlights the issues of user roles with respect to a command, planning and control system and explains the way in which O-Plan characterises user interactions. The section also describes the interfaces built for the current O-Plan planner.

The main theme of the O-Plan research has been the identification of separable support modules, internal and external interface specifications, and protocols governing processing behaviours which are relevant to an AI planning system. These various contributions having been introduced in earlier sections of the Architecture Guide are drawn together in a separate section.

This Architecture Guide describes an *idealised* conceptual overview of the O-Plan design. The current O-Plan Implementation goes some way towards achieving our aims. A separate Implementation Guide is available which describes the current state of the prototype.

The Architecture Guide contains the following sections:

- Section 2 describes our philosophy for communication between agents in a simple command, planning and control environment;
- Section 3 describes the representation of a plan within O-Plan;
- Section 4 explains the mechanisms used in O-Plan for managing concurrent computations and deciding on the order of processing;
- Section 5 describes the major components of the O-Plan architecture;
- Section 6 goes into greater detail on how the planning agent has been provided in the O-Plan architecture;
- Sections 7 and 8 outline the task assignment and execution systems in O-Plan;
- Section 9 describes the user interface which has been designed for O-Plan;
- Section 10 looks at performance issues in the O-Plan prototype;
- Section 11 summarises the various aspects that relate to the modularity, interfaces and internal protocols within O-Plan - an important aspect of the design;
- Related projects and our plans for the future.

2 Communication in Command, Planning and Control

The aim of this section is to describe in broad terms the motivation and reasoning behind the design of the O-Plan architecture. Edinburgh research on planning and control architectures is aimed at building a practical prototype system which can generate plans and can reliably execute the plans in the face of simple plan failures.

2.1 Motivation of the O-Plan Architecture

We are using our experiences in the application of AI planning techniques to practical projects to develop a planning system that closes the loop between planning and executing. There have been some successes with previous attempts at closing the loop [12],[17],[23],[43], but often the plans generated were rather limited and not very flexible. In general, the complexities of the individual tasks of plan representation, generation, execution monitoring and repair has led to research into each of these issues separately. In particular, there is now a mismatch between the scale and capabilities of plan representations proposed for real-time execution systems [18], [26],[28], and those that can be generated by today's AI planners. However, in most realistic domains the demand is for a system that can take a command request, generate a plan, execute it and react to simple failures of that plan, either by repairing it or by re-planning. Explicit knowledge about the structure of the plan, the contribution of the actions involved and the reasons for performing plan modifications at various stages of the plan construction process, provides us with much of the information required for dealing with plan failures. Such knowledge is also essential for further planning and re-planning by identifying generalisations or contingencies that can be introduced into the plan in order to avoid similar failures.

One of the largest simplifications most planners to date have made is to assume plans are constructed with full knowledge of the capabilities of the devices under their control. Thus, executing such plans involves the direct application of the activities within the plan by an execution agent which has no planning capability. Unfortunately, unforeseen events will occur causing failure of the current plan and a request for repair of the plan or re-planning directed at the planning system. Building into the execution agent some ability to repair plans and to perform re-planning would improve the problem solving performance of the execution agent, especially when it is remote from the central planning system.

2.2 The Scenario

The scenario we are investigating is as follows:

- A user specifies a task that is to be performed through some suitable interface. We call this process *task assignment*.
- A *planner* plans and (if requested) arranges to execute the plan to perform the task specified. The planner has knowledge of the general capabilities of a semi-autonomous execution system but does not need to know about the actual activities that execute the actions required to carry out the desired task.

- The *execution system* seeks to carry out the detailed tasks specified by the planner while working with a more detailed model of the execution environment than is available to the task assigner and to the planner.

We have deliberately simplified our consideration to three agents with these different roles, and with possible differences of requirements for user availability, processing capacity and real-time reaction, to clarify the research objectives in our work.

The execution agent executes the plan by choosing the appropriate activities to achieve the various sub-tasks within the plan, using its knowledge about the particular resources under its control. Thus, the central planner communicates a general plan to achieve a particular task, and responds to failures fed back from the execution agent which are in the form of flaws in the plan. The execution agent communicates with the real world by executing the activities within the plan and responding to failures fed back from the real world. Such failures may be due to the inappropriateness of a particular activity, or because the desired effect of an activity was not achieved due to an unforeseen event. The reason for the failure dictates whether the same activity should be re-applied, replaced with other activities or whether re-planning should take place.

2.3 Use of Dependencies

The use of dependencies within planning promises great benefits for the overall performance of a planning system particularly for plan representation, generation, execution and repair.

The notion of the teleology of a plan, which we call the Goal Structure [35], refers to the dependencies between the preconditions and postconditions of activities involved in the plan. Although, such dependencies have been shown to be useful for describing the internal structure of the plan and for monitoring its execution [17], [36], there has been no comprehensive discussion of their use in all aspects of plan generation, execution monitoring and plan repair. Knowledge-rich plan representations of this type were used as the basis for the design of an Interactive Planning Assistant [2] [16] for the UK Alvey PLANIT Club. This allowed for browsing, explaining and monitoring of plans represented in a more useful form than that provided in conventional computer based planning support tools. More recently, O-Plan style plan representations were used within the OPTIMUM-AIV system [3] for spacecraft assembly, integration and verification at the European Space Agency in work conducted by a consortium of which AIAI was a part.

Early work on Decision Graphs [19] at Edinburgh has shown how the explicit recording of the decisions involved in the planning process could be used for suggesting where and how much re-planning should take place when unforeseen situations make the current plan fail. Some work to link these ideas with Nonlin was undertaken during the mid 1970's [11].

2.4 A Common Representation for Communication between Agents

We are exploring a common representation for the input/output requirements and capabilities of a planner and execution agent. This supports the representation of the communication between a user, requesting the plan, and the real world, in which the plan is being executed. Such communication may take place either directly through a planner or indirectly via a central planner and a dumb or semi-autonomous execution agent. In the latter case, the communication between the central planner and the execution agent becomes an interesting research issue.

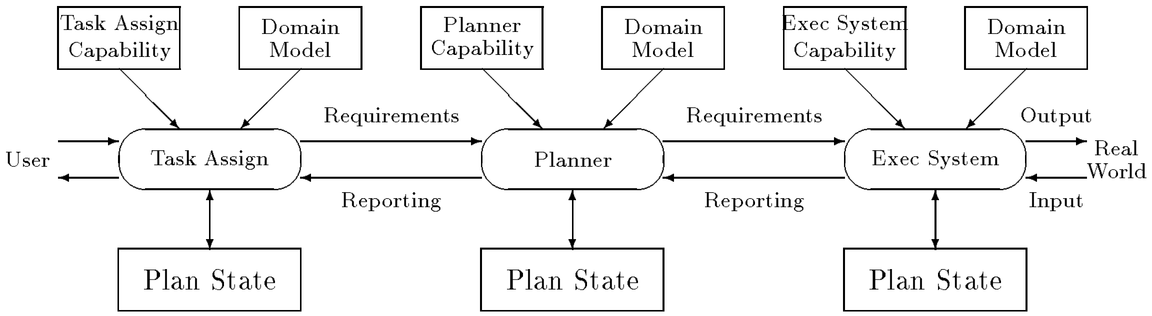

Figure 1: Communication between Task Assignment, Planning and Execution Agents

The common representation includes knowledge about the capabilities of the planner and execution agent, the requirements of the plan and the plan itself either with or without flaws (see Figure 1). Thus, a planner will respond to the requirements of a user. Based on the knowledge of its own capabilities and that of the execution environment, it will generate a plan. This plan may then be executed directly in the real world, or, indirectly via an execution agent. The execution agent executes this plan in the real world and monitors the execution, responding to failures in one of two ways. If it does not have knowledge of its own capabilities, it simply returns knowledge of the failure to the central planner and awaits a revised plan to be sent. In this case, the execution agent is dumb. If it does have knowledge of its own capabilities, it may attempt to repair the plan and then continue with execution. On the other hand, if a repair is beyond the capabilities of the execution agent, then this knowledge is fed back to the central planner and again a revised plan is expected. In this case, the execution agent is semi-autonomous. When failures during the application of the plan are fed back to the planner, these may be acted upon by it and a repair of the plan made or total re-planning instigated. This may, in turn, involve the user in reformulating the task requirement. A revised or new plan is then executed. Finally, success of the execution or partial execution of the plan is fed back to the user.

Other issues relating to the choice of the common representation and communication protocols include:

- when to repair the plan or when to seek re-planning,
- continuing execution of parts of a plan, not affected by the failure,
- continuing to maintain a safe execution state even while awaiting initial commands or the correction of faults in earlier plans,
- maintaining integrity and synchronisation of communicated plans and flaws.

3 Representing and Communicating Plans

3.1 Plan States

One of the most important problems which needs to be addressed in any planning system is that of plan representation. An O-Plan agent's *plan state* holds a complete description of a plan at some level of abstraction. The plan state also contains a list of the current *flaws* in the plan. Such flaws could relate to abstract actions that still must be expanded before the plan is considered valid for passing on for execution, unsatisfied conditions, unresolved interactions, overcommitments of resource, time constraint faults, etc. The Plan State can thus stand alone from the control structure of the AI planner in that it can be saved and restored, passed to another agent, etc.

At any stage, a plan state represents an abstract view of a set of actual plans that could be generated within the constraints it contains. Alternative lower level actions, alternative action orderings and object selections, and so on are aggregated within a high level Plan State description.

3.1.1 Task Formalism (TF)

Task Formalism (TF) (as used in Nonlin and the first version of O-Plan) is a declarative language for expressing action schemata, for describing task requests and for representing the final plan. It allows time and resource constraints in the domain to be modelled. The planner can take a plan state as a requirement (created by a TF Compiler from the user provided task specification in TF) and can use a library of action schemata or generic plan state fragments (themselves created by the TF Compiler from a domain description provided by the user) to transform the initial plan state into one considered suitable for termination. This final plan state could itself be decompiled back into a TF description if required.

Our design intention for O-Plan is that any plan state (not just the initial task) can be created from a TF description and vice versa. This was not fully achieved in the first O-Plan prototype [10], but this remains our goal.

The O-Plan design allows for different plan state representations in the different agents. Task Formalism is particularly suited to the representation of a plan state within the planner agent and, hence, to act as a basis for communication to the planner's superior (task assignment) and subordinate (execution system) agents. The actual plan state inside the task assignment and execution system agents is likely to differ from that within the planner. For example, the execution system may be based on more procedural representations as are found in languages like PRS (the Procedural Reasoning System [18]) and may allow iteration, conditionals, etc.

3.1.2 Plan Flaws

The plan state cannot contain arbitrary data elements. The AI planner is made up of code that can interpret the plan state data structure and interpret the lists of flaws in such a way that it can select from amongst its computational capabilities and its library of domain specific information to seek to transform the current Plan State it is given into something that is desired by the overall architecture. This is defined as the reduction of the list of *flaws* known to the planner. The O-Plan architecture associates a Knowledge Source with each flaw type that can be processed [9]. An agenda of outstanding flaws is maintained in a Plan State and appropriate Knowledge Sources are scheduled on the basis of this.

The O-Plan architecture is designed for operation in an environment where the ultimate aim of termination will not be achieved. There will be new command requests arriving and earlier ones being modified, parts of plans will be under execution as other parts are being elaborated, execution faults are being handled, etc.

We believe that the basic notions described above can serve us well as a basis for an attack on the problem of coordinated command, planning and execution in continuously operating domains. There must be a means to communicate plan related information incrementally between the agents involved with commanding, planning and executing plans. Each of these agents will have their own level of model of the current command environment, plan and execution environment. We will explore the properties that we must seek from our basic notions in the following sections.

3.2 Plan Patches

The requirement for asynchronously operating planners and execution agents (and indeed users and the real world) means that it is not appropriate to consider that a plan requirement is set, passed on for elaboration to the planner and then communicated to a waiting execution agent which will seek to perform the actions involved. Instead, all components must be considered to be operating independently and maintaining themselves in some stable mode where they are responsive to requests for action from the other components. For example, the execution agent may have quite elaborate local mechanisms and instructions to enable it to maintain a device (say a spacecraft or a manufacturing cell) in a safe, healthy, responsive state. The task then is to communicate some change that is requested from one component to another and to insert an appropriate alteration in the receiver such that the tasks required are carried out.

We define a *Plan Patch* as a modified version of the type of Plan State used in the first version of O-Plan. It has some similarity to an operator or action expansion schema given to an AI planning system in that it is an abstracted or high level representation of a part of the task that is required of the receiver using terminology relevant to the receiver's capabilities. This provides a simplified or black-box view of possibly quite detailed instructions needed to actually perform the action (possibly involving iterators and conditionals, etc). Complex execution agent representational and programming languages can be handled by using this abstracted view (e.g., [18], [26]). For example, reliable task achieving *behaviours* which included contingencies and safe state paths to deal with unforeseen events could be hidden from the planner by communication in terms of a simplified and more robust model of the execution operations [23].

Outstanding flaws in the Plan Patch are communicated along with the patch itself. However, these flaws must be those that can be handled by the receiver.

It can be seen that the arrangement above (mostly assumed to refer to the communication between a planner and execution agent) also reflects the communication that takes place between a task assigner and the planner in an O-Plan type AI planner. Requiring rather more effort is the investigation of suitable Plan Patch constructs to allow execution errors to be passed back to the planner or information to be passed back to the task assigner, but we believe that this is a realistic objective.

3.3 Plan Patch Attachment Points

There is a need to communicate the points at which the Plan Patch should be attached into the full Plan State in the receiver. The sender and receiver will be operating asynchronously and one side must not make unreasonable assumptions about the internal state of the other.

We endow all the agents with a real-time clock that can be assumed to be fully synchronised. We also make simplifying assumptions about delays in communication to keep to the immediate problem we are seeking to tackle (while fully believing that extension to environments where communication delay is involved will be possible). Therefore, metric time is the "back-stop" as a means of attaching a Plan Patch into the internal Plan State of the receiver. Metric time is also important to start things off and to ensure a common reference point when necessary (e.g., in cases of loss of control).

However, the use of metric time as an attachment point lacks flexibility. It gives the receiver little information about the real intentions behind the orderings placed on the components of the Plan Patch. It will, in some cases, be better to communicate in a relative or qualified way to give the receiver more flexibility. Suitable forms of flexible Plan Patch Attachment Point description will be investigated in future (such as descriptions relative to the expected Goal Structure [35] of the receiver).

3.4 Incremental Plan States

Our approach is to combine the ideas above to define an *Incremental Plan State* with three components:

- a plan patch,
- plan patch flaws as an agenda of processing requirements,
- plan patch attachment points.

Such Incremental Plan States are used for two way communication between the task assigner and the planner and between the planner and the execution agent. The O-Plan Plan State structures and flaw repertoire has been extended to cope, initially, with a dumb execution agent that can simply dispatch actions to be carried out and receive fault reports against a nominated set of conditions to be explicitly monitored (as described in [36]). In future research, the Plan State data structures and flaw repertoire will be extended again to cope with a semi-autonomous execution agent with some capability to further elaborate the Incremental Plan States and to deal locally with re-planning requirements [27].

A means to compile an Incremental Plan State from a modified type of Task Formalism (TF) declarative description (and vice versa) will be required.

3.5 Plan Transactions

The overall architecture must ensure that an Incremental Plan State can be understood by the receiver and is accepted by it for processing. This means that all the following are understood by the receiver:

- plan patch description is clear,
- plan patch flaws can be handled by the receiver's Knowledge Sources,
- plan patch attachment points are understood.

It is important that the sender and receiver (whether they are the task assigner and the AI planner, the planner and the execution agent, or one of the reverse paths) can coordinate to send and accept a proposed Incremental Plan State which the receiver must assimilate into its own Plan State. We propose to use *transaction processing* methods to ensure that such coordination is achieved.

Specific flaw types and Knowledge Sources have been created in the various agents (task assignment, AI planner, and execution agent) to handle the extraction and dispatch (as an Incremental Plan State) of a part of an internal Plan State in one component, and the editing of such an Incremental Plan State into the internal Plan State of the receiver. The "extraction" Knowledge Sources must be supplied with information on the Plan Patch description, flaw types and attachment points that the receiver will accept. This constitutes the primary source of information about the capabilities of the receiver that the sender has available and its representation will be an important part of the research.

Communication "guards" ensure that the *a priori* criteria for acceptance of an Incremental Plan State for processing by the receiver's Knowledge Sources are checked as part of the Plan Transaction. It may also be the case that initial information about urgency will be able to be deduced from this acceptance check to prioritise the ordering of the new flaws with respect to the existing entries on the agenda in the receiver.

4 Managing Concurrent Computations

The O-Plan architecture has been designed to allow for concurrent processing where possible. The systems implementation itself is composed of a number of parts representing the major components of the architecture. In theory these can be run as separate processes if desired. In addition, the basic flow of processing performed by the architecture allows for a *wavefront* of concurrent threads of computation to be maintained and decisions can be taken about where to deploy any computational effort available (whether this is actually implemented with parallel processors or not).

The first O-Plan project made a start on mechanisms for the implementation of an efficient planning system able to take an opportunistic approach to selecting where computational effort should be concentrated during planning. However, some limitations were observered and taken into account during the subsequent design of O-Plan. The new O-Plan mechanisms are listed in the following sections.

4.1 Choice Ordering Mechanisms in the Earliest Version of O-Plan

4.1.1 Building up Information in an Agenda Record

The first version of O-Plan included the ability to allow a knowledge source to examine a possible decision point (represented by the agenda entry it is asked to process) and to add information relating to the choice to the fields of the agenda record. If the choice did not become suitably tightly restricted as a result of the addition of this information, it was possible to put the agenda entry back onto the outstanding flaws list with improved information for deciding on the time to reselect it for processing. The ability to build up information around an agenda entry in an incremental way prior to a final knowledge source activation is an important feature that ensures that work done in accessing data bases and checking conditions can be saved as far as possible when processing is halted. There are some similarities to mechanisms within real-time responsive architectures such as RT-1 [34].

4.1.2 Granularity of Knowledge Sources

Each knowledge source within the O-Plan architecture encodes a piece of planning knowledge. For example, how to expand an action, bind a variable, check a resource, etc. From a modularity viewpoint, there is some advantage in having a very fine grain of knowledge source to implement planning knowledge. However, this can lead to tens of agenda entries and knowledge source activations with the overheads associated with such activations for even the simplest types of action expansion. In simpler planners, such as Nonlin, an expansion is efficiently handled as an atomic operation. There is a conflicting desire to have efficient large grain knowledge sources implementing planning knowledge and very fine grain knowledge sources detailing each individual step of some higher level plan modification operator.

in the first version of O-Plan, which had finer grain of knowledge sources, it was also found that ordering relationships between agenda entries left in the agenda list had to be stated to ensure efficient processing. The controller was then required to unravel the web of activation orderings that resulted. A special form of agenda entry called a *sequence* was implemented in the first version of O-Plan to assist the controller in this task, it would only consider the head of the sequence for activation at any time, subsequently releasing the following agenda items clustered in the sequence in the order indicated. This process is similar to the control blocks used in the Tecknowledge S.1 system [40].

4.1.3 Priority of Processing Agenda Entries

The first version of O-Plan assigned priorities to every flaw as it was placed on the agendas. The priorities were calculated from the flaw type, the degree of determinacy of the flaw and information built up in the Agenda Record as described earlier. These provide measures of choice within the flaw. Two heuristic measures were maintained in each agenda entry. One called *Branch-1* indicated the immediate branching ratio for the choice point. An upper bound on this can be maintained quite straightforwardly. The second measure was called *Branch-n* and gave a heuristic estimate of the number of distinct alternatives that could be generated by a naive and unconstrained generation of all the choices represented by the choice point.

In the first version of O-Plan, three agendas were maintained to efficiently select between agenda entries which were ready for knowledge source activation and ones awaiting further information to bind open variables in the agenda information. This is described in [9]. Eventually though, the ready to run agenda entries are simply rated according to a numerical priority maintained for each agenda entry on the basis of flaw type and estimators which said how many choices there could be down a particular search branch (the *Branch-1* and *Branch-n* estimators). This forms too simplistic a measure for allowing the controller to decide between waiting agenda entries. Consideration was given to a rule based controller with knowledge of other *measures of opportunism* but no implementation of this was done within the original O-Plan system.

4.2 Choice Ordering Mechanisms now in O-Plan

O-Plan now seeks to provide a more coherent set of mechanisms to enable the planning and control system builder to select suitable implementation methods for describing choices, posting constraints which will restrict choice, postponing choice making decisions until the most opportune time to make them, and triggering choices that are ready to be acted upon. These mechanisms are:

- the use of *stages* in knowledge sources to allow for a linear thread of computation to be defined which can be assumed to run through to completion, but provides a means for interruption at defined staging points.
- the definition of *triggers* on knowledge sources and knowledge source stages to provide a clear means to delegate a higher level of knowledge source activation checks to the controller.
- the use of *compound agenda entries* to put direct dependencies of some tasks on others that must complete earlier. This allows complex computational dependencies and strategies

to be created.

- the use of *agenda manager priorities* to allow the controller to select appropriate ready-to-run agenda entries and match these to waiting knowledge source platforms.

The following sections explain each of these mechanisms in more detail.

4.2.1 Knowledge Source Stages

The mechanism in the first version of O-Plan for building up information in an agenda entry prior to making some selection between alternatives was a very useful feature but proved difficult to use in practice. A knowledge source had to be activated to initiate processing which might simply add a little information to the agenda entries and then suspend to allow the controller to decide whether to progress. This is very inefficient.

In O-Plan, knowledge sources are defined in a series of *stages*. There can be one or more stages, only latter stages may make alterations to the plan state (thus locking out other knowledge source final stages which can write to the same portion of the plan state). Any earlier stages may build up information useful to later stages. At the end of any stage, the knowledge source must be prepared to halt processing if asked to by the controller. If it is asked to halt at a stage boundary, the knowledge source may summarise the results of its computation in a field of the agenda record provided for this purpose. A controller directed support routine is called by the knowledge source at the end of each stage to identify whether it must halt or may continue. This allows the controller to dynamically re-direct computation as it considers all the information available to it, while providing a simple and efficient way for the knowledge source to continue computation without intermediate state saving while it continues to receive a go-ahead from the end of stage continuation authorisation routine.

A *Knowledge Source Formalism* for O-Plan is being designed to allow for stage definition and to assist with declaring the restrictions on the plan state portions affected by the final plan state modifying stage of the knowledge source - to assist in lock management in parallel implementations.

4.2.2 Knowledge Source Triggers

In O-Plan, a mechanism of setting *triggers* on agenda entries for activating knowledge sources (and an individual stage of a knowledge source if desired) is provided. The triggers may use various "items" of data available within the plan state and other global information available to the planner. These may include things such as the availability of a specific binding for a plan variable, the satisfaction of a condition at a specific action node in the plan network, the use of a specific resource, the occurrence of an external event, information from the "clock" within the planner, etc. The Knowledge Source Formalism referred to earlier will also be used to define triggers on knowledge source stages. The triggering constructs in the language are initially quite restrictive to ensure that efficient agenda entry triggering mechanisms can be implemented. However, as we gain experience, we expect the triggering language to be quite comprehensive. A knowledge source may also dynamically create a trigger on a continuation agenda entry when halting processing at a stage boundary.

Only agenda entries which are currently triggered will be available to the controller for decisions on which entries to activate through to a knowledge source running on a knowledge source platform.

4.2.3 Compound Agenda Entries

Individual *simple* agenda entries can be grouped together into *compound* agenda entries. Only the head entries in the compound agenda entry are considered at any time by the controller (and possibly by the triggering mechanism considered above), thus cutting down on the amount of processing required by the controller to select the next agenda entry to execute when such pre-defined orderings can be specified. Compound agenda entries can be made by knowledge sources to act as a meta-processing level to implement some definite planning strategy or to implement planning algorithms with finer grain knowledge sources to provide modularity and real time response improvement.

4.2.4 Controller Priorities

The controller is given the task of deciding which of the current set of triggered agenda entries should be run on an available knowledge source platform. It does this by considering the priority and measures of opportunism of the agenda entry. Four priority levels are available within O-Plan - Low, Medium, High and Emergency. The Emergency priority level is only available to handle incoming external events. The RT-1 system has similar priority based processing arrangements [34]. In certain cases, an O-Plan implementation will possess knowledge source platforms dedicated to processing specific real-time responsive events appearing as agenda entries - thus allowing for reliable real-time response to events categorised as Emergency priority.

A waiting knowledge source platform will be able to run one, several, or all knowledge sources. Any restriction on a specific platform will be known to the controller. Only triggered agenda entries at the highest priority level which can be processed on a waiting knowledge source are considered by the controller on each cycle. Where there is still choice, a range of *measures of opportunism and priority* are employed to make a selection. The underlying principle is to make a selection according to a strategy given to the controller. The strategy will use user selected preferences or by default will seek to reduce search to the extent it can judge this (reflecting the opportunistic generative planning nature of O-Plan). Measures such as *Branch-1* (the immediate branching ratio for the choice point) and *Branch-n* (a heuristic estimate of the number of distinct alternatives that could be generated by a naive and unconstrained generation of all the choices represented by the choice point) are relevant to this. However, the use of a utility function guided by task specifiers given to the controller will be explored for O-Plan when it is used in continuous command and control applications.

5 O-Plan Architecture

This section describes the O-Plan architecture in detail and describes the major modules which make up the system. An agenda based architecture forms the central feature of the system and the design approach. Within this framework, however, the emphasis on and development of search strategies has been concentrated into crisper notions of choice enumeration, choice ordering, choice making and choice processing. This is important as it allows us to begin to justifiably isolate functionality which can be described in terms of:

- triggering mechanisms — *i.e.* what causes the mechanism to be activated,
- decision making roles — precisely what type of decision can be made
- implications for search — has the search space been pruned, restricted or further constrained as far as possible.
- decision ordering — in what order should we choose between the alternative decisions possible.
- choice ordering — for a decision to be made, which of the open choices should we adopt.

The O-Plan approach to command, planning, scheduling and control can be characterised as follows:

- successive refinement/repair of a complete plan or schedule which contains an agenda of outstanding issues
- least commitment approach
- using opportunistic selection of the focus of attention on each problem solving cycle
- building plan information incrementally in "constraint managers", e.g.,
  - time point network manager
  - object/variable manager
  - effect/condition manager
  - resource utilisation manager
- using localised search to explore alternatives where advisable
- with global alternative re-orientation where necessary.

The O-Plan project has sought to identify modular components within an AI command, planning and control system and to provide clearly defined interfaces to these components and modules. The various components plug into "sockets" within the architectural framework. The sockets are specialised to ease the integration of particular types of component. See Figure 2.

The various components of the agent architecture are:

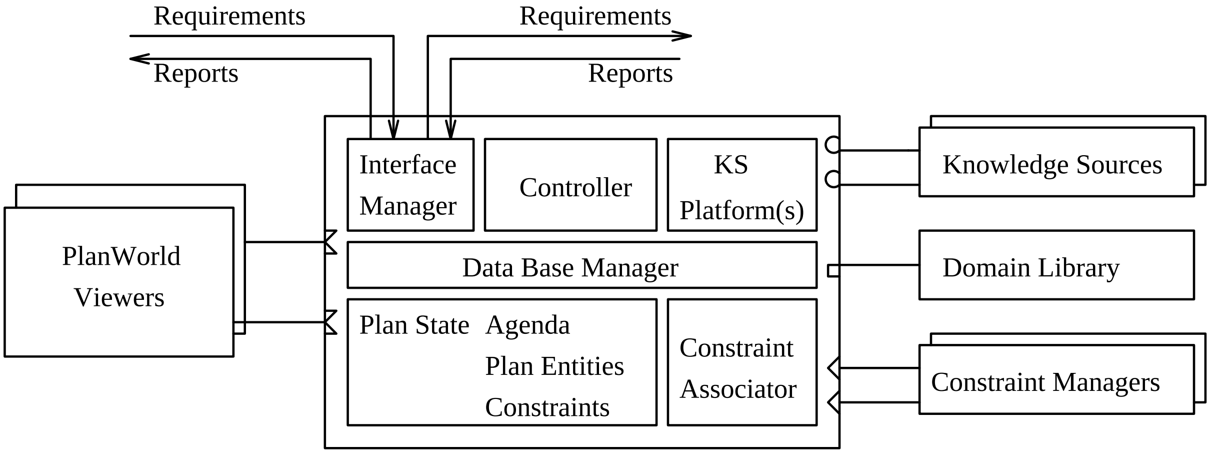

Figure 2: O-Plan Agent Architecture

PlanWorld Viewers – User interface, visualisation and presentation viewers for the plan – usually differentiated into technical *plan* views (charts, structure diagrams, etc.) and *world* views (simulations, animations, etc.).

Knowledge Sources – Functional components which can analyse, synthesise or modify plans.

Domain Library – A description of the domain including a library of possible actions.

Constraint Managers – Components which manage detailed constraints within a plan and seek to maintain as accurate a picture as possible of the feasibility of the current plan state with respect to the domain model.

These plug-in components are orchestrated by an O-Plan agent kernel which carries out the tasks assigned to it via appropriate use of the Knowledge Sources and manages options being maintained within the agent's *Plan State*. The central control flow is as follows:

Interface Manager – Handles external events (requirements or reports) and, if they can be processed by the agent, posts them on the agent *Agenda*.

Controller – Chooses Agenda entries for processing by suitable Knowledge Sources.

Knowledge Source Platform(s) – Chosen Knowledge Sources are run on an available and suitable Knowledge Source Platform.

Data Base Manager – Maintains the Plan State and provides services to the Interface Manager, Controller and Knowledge Sources.

Constraint Associator Acts as a mediator between changes to the Plan State made by the data base manager and the activities of the various Constraint Managers that are installed in the agent. It eases the management of interrelationships between the main plan entities and detailed constraints.

5.1 Domain Information

Domain descriptions are supplied to O-Plan in a structured language, which is compiled into the internal data structures to be used during planning. The description includes details of:

1. activities which can be performed in the domain.
2. information about the environment and the objects in it.
3. task descriptions to describe the planning requirements.

The structured language (we call it Task Formalism or TF) is the means through which a domain writer or domain expert can supply the domain specific information to the O-Plan system, which itself is a domain *independent* planner. O-Plan embodies many search space pruning mechanisms using this domain information (strong methods) and will fall back on other weak (search) methods, if these fail. The Task Formalism is the mechanism that enables the user of the system to supply domain dependent knowledge to assist the system in its search.

5.2 Plan State

In contrast to the infrequently changing domain information outlined above, the plan state (on the left of Figure **??**) is the dynamic data structure used during planning and houses the emerging plan. There are many components to this structure, the principal ones being:

- the plan network itself. O-Plan has retained a partially ordered network of activities as the basis of its plan representation, as originally suggested in the NOAH planner. In O-Plan the plan information is concentrated in the "Associated Data Structure" (ADS). The ADS is a list of node and link structures noting temporal and resource information, plan information and a plan history.
- the plan causal structure (sometimes called the *teleology* of the plan). Borrowing from Nonlin and earlier work on O-Plan, the system keeps explicit information to "explain" why the plan is built the way it is. This rationale is called the Goal Structure (GOST) and, along with the Table of Multiple Effects (TOME), provides efficient support to the condition achievement support module (Question Answerer or QA) used in O-Plan (*c.f.* Chapman's Modal Truth Criteria [8]).
- the agenda list(s). O-Plan starts with a complete plan, but one which is "flawed", hence preventing the plan from being capable of execution. The nature of the flaws present will be varied, from actions which are at a higher level than that which the agent can operate, to notes of linkages necessary in the plan to resolve conflict. "Flaws" may also represent potentially beneficial, but as yet unprocessed, information. The agenda lists are the repository for this information which must be processed in order to attain an executable plan. The original O-Plan system used 3 agenda lists. In later versions of O-Plan, effort has been made to improve the structure of agenda information and the triggering mechanisms. Only one main agenda is kept in a plan state although alternative plan states still require a separate agenda as in the first version of O-Plan.

The plan state is a self-contained snapshot of the state of the planning system at a particular point in time in the plan generation process. It contains all the state of the system hence the generation process can be suspended and this single structure rolled back at a later point in time to allow resumption of the search[1].

5.3 Knowledge Sources

These are the processing units associated with the processing of the flaws contained in the plan and they embody the planning knowledge of the system. There are as many knowledge sources (KSs) as there are flaw types, including the interface to the user wishing to exert an influence on the plan generation process. The KSs draw on information from the static data (*e.g.* the use of an action schema for purposes of expansion) to process a single flaw, and in turn they can add structure to any part of the plan state (*e.g.* adding structure to the plan, inserting new effects or further populating the agenda(s) with flaws).

5.4 Support Modules

In order to efficiently support the main planning functionality and provide constraint management in O-Plan there are a number of support modules separated out from the core of the planner. These modules have carefully designed functional interfaces in order that we can both build the planner in a piecewise fashion, and in particular that we can experiment with and easily integrate new implementations of the modules. The modularity is possible only through the experience gained in earlier planning projects where support function requirements were carefully separated out from the general problem solving and decision making demands of the system.

Support modules are intended to provide efficient support to a higher level where decisions are taken. They should not take any decision themselves. They are intended to provide complete information about the constraints they are managing or to respond to questions being asked of them by the decision making level itself.

The support modules include the following:

- Time Point Network (TPN) Manager to manage metric and relative time constraints in a plan.
- Question-Answering (QA). Akin to Chapman's Modal Truth Criteria [8], this is the process at the heart of O-Plan's condition achievement procedure. It answers the basic question of whether a proposition is true or not at a particular point in the plan. The answer it returns may be (i) a categorical "yes", (ii) a categorical "no", or (iii) a "maybe", in which case QA will supply an alternative set (structured as a tree) of strategies which a knowledge source can choose from in order to ensure the truth of the proposition.

[1]Assuming that the Task Formalism and the knowledge sources used on re-start are the same "static" information used previously.

- TOME and GOST Manager (TGM) to manage the causal structure (conditions and effects which satisfy them) in a plan.
- Plan State Variables Manager to manage partially bound objects in the plan.
- Resource Utilisation Manager to monitor and manage the use of resources in a plan.
- Instrumentation and Diagnostic routines. O-Plan has a set of routines which allow the developer to set and alter levels of diagnostic reporting within the system. These can range from full trace information to fatal errors only. The instrumentation routines allow performance characteristics to be gathered while the system is running. Information such as how often a routine is accessed, time taken to process an agenda entry, etc, can be gathered.

5.5 Controller

Holding the loosely coupled O-Plan framework together is the Controller acting on the agenda. Items on the agendas (the flaws) will have a context dependent priority which the controller can re-compute, and which allows for the opportunism required to drive plan generation. The agenda mechanism and manager have been simplified from the early O-Plan work in that two of the three agendas have been collapsed into a single structure. Entries on this single structure employ a triggering mechanism for activating the knowledge sources via the use of plan state or other data. Triggering on specific occurrences, such as the binding of a variable, the satisfaction of a condition, the occurrence of an external event, etc., allow an efficiency to be built into O-Plan that was missing in the earlier version, which used a priority scheme whereby agenda entries were prioritised at time of entry. This enhanced scheme does have an impact on the extra complexity of knowledge source required, forcing rules to be set regarding the writing of knowledge sources. In return however, this has given us knowledge sources with much greater capability than previously achieved. For example a knowledge source may be able to dynamically create a trigger for the continuation of another agenda entry on suspension of the current entry's processing.

An agenda of alternative plan states is also held by the Controller for search purposes as was the case in the first version of O-Plan.

5.6 Discussion

Having reviewed the main components in the O-Plan architecture, we wish to make some observations on a number of issues.

5.6.1 Knowledge Sources

The first O-Plan planning prototype allowed knowledge sources to perform operations at a relatively low level. This proved unsuitable for some planning activities, such as that of expanding an action, or satisfying a condition, where there is generally a large amount of work involved.

This includes the introduction of structure to the plan and the posting of effects and conditions. All these entities are related and the first version of O-Plan had difficulty treating these sub-operations as separate schedulable agenda entries with suitable priorities. In the later stages of that research we introduced the notion of a *sequence* to re-establish the relationships between the various entries, with partial success. A cleaner mechanism, which we refer to as compound agenda entries, is being explored for later versions of O-Plan to allow for knowledge of complex sequencing of planning decisions to be provided to the planner by the knowledge source writer [38].

In addition, O-Plan employs a knowledge source staging scheme where the knowledge sources allow for work to be deliberately *staged* [38]. At each stage the information within the agenda entry is progressively built up for use in later stages. Only the later stages are given permission to alter the final destination of this information, i.e. the plan state. At the end of each stage the knowledge source needs to satisfy *staging conditions* in order to continue processing to subsequent stages; thus the controller has the ability to halt processing and suspend the knowledge source. The agenda record itself carries all the "state" of the processing, so can safely be returned to the agenda for later resumption; the knowledge sources themselves are stateless.

The advantages of this scheme are many: firstly there is no longer the yes/no situation of whether an agenda entry can be processed as the information can be built up in stages. This in turn offers the controller greater flexibility in its task of dynamically computing priorities for agenda records awaiting processing. This much enhances the ability to exploit parallelism and opportunism in the system.

Knowledge sources run on Knowledge Source Platforms, which are basically processing engines for the knowledge source code. It is hoped that the eventual O-Plan will exploit multiprocessor architectures, where possible, so the current system has a clean separation of its knowledge source platforms from the other system modules, and locking mechanisms need to be put in place to ensure that data in the system is up to date and consistent. Only the final stages of a knowledge source can change any of the plan state; earlier stages merely build up information locally. It is intended to investigate a language for describing Knowledge Sources (Knowledge Source Framework). Amongst other things this may allow for information concerning the selective locking of parts of the database to be gathered.

5.6.2 Controller Strategies

The Controller plays a major role in the operation of the planner, and is largely responsible for achieving the degree of opportunism sought in an agent. Its main role is to choose a candidate from amongst the set of currently triggered agenda entries to be loaded onto an appropriate and available knowledge source platform. For this reason the Controller is also known as the Agenda Manager. In order to do this work effectively and flexibly the controller must consider priorities (attached to or computed) for each of the triggered agenda entries. Priorities can be relatively complex and based around the *type* of the agenda entry and its measure of determinism. The first version of O-Plan used heuristic measures detailing the amount of choice contained in an entry both at the "top" (i.e. the measure of choice seen immediately) and at the "bottom" (i.e. a measure, or estimate, of the eventual choice encountered if the entry is chosen). In the first version of O-Plan these were referred to as the *Branch-1* estimator and the *Branch-n* estimator (a heuristic estimate of the number of distinct alternatives that could be generated by a naive and unconstrained generation of all the choices represented by the choice point). These measures have proved useful in distinguishing between choice items and they ensure that opportunism is exploited where possible.

The controller is designed in such a way that it can operate with different pre-loaded strategies and utility functions. At present the system operates with a simple default strategy (knowledge source priorities fixed by the developer) but as the representational range of the Task Formalism increases it can facilitate the loading of domain-specific and specialised strategies and utility functions. The controller will be the subject of further research as we wish to develop more powerful strategies, including:

- Qualitative Modelling. As O-Plan develops for use in continuous command and control applications the need to predict and recover from situations becomes much more demanding. An important role for the controller then is to behave in a much more pro-active manner, exploiting as much knowledge of the system as possible. The earlier work of Drabble [12] provides a good starting position for how this will be achieved.
- Ordering Mechanisms. Temporal Coherence (TC) [14] showed that algorithms must be developed to address the many variants of ordering problems (TC addressed the problem of *condition* pre-ordering). Effective controller operation requires recognition of triggering mechanisms for appropriate ordering-related algorithms.

6 O-Plan Planner

6.1 Plan State

The planning agent plan state holds information about decisions taken during planning and information about decisions which are still to be made (in the form of an agenda).

6.1.1 Plan Network - ADS and TPN

The Associated Data Structure (ADS) provides the *contextual* information used to attach meaning to the contents of the Time Point Network (TPN), and the data defining the emerging plan. The main elements of the plan are activity, dummy and event nodes with ordering information in the form of links as necessary to define the partial order relationships between these elements. The separation of the ADS level from the time points associated with the plan entities is a design feature of the current O-Plan system and differs from our previous approach in Nonlin and the first O-Plan prototype. It is motivated by our approach to time point constraint management [7] which reasons about both ends of plan entities (such as nodes and links) and which can be more efficiently implemented where there is uniformity of representation.

Time windows play an important part in O-Plan in two ways: firstly as a means of recording time limits on the start and finish of an action and on its duration and delays between actions, and secondly during the planning phase itself as a means of pruning the potential search space if temporal validity is threatened. Time windows in O-Plan are maintained as **min/max** pairs, specifying the upper and lower bounds known at the time. Such bounds may be symbolically defined, but O-Plan maintains a numerical pair of bounds for all such numerical values. In fact, a third entry is associated with such numerical bounds. This third entry is a *projected* value (which could be a simple number or a more complex function, data structure, etc.) used by the planner for heuristic estimation, search control and other purposes[2].

Higher level support modules (such as QA, the TOME and GOST Manager, etc.) rely on the detail held in the ADS and on the functionality provided by the TPN. The ADS is maintained by a set of routines which we refer to as the Network Manager.

6.1.2 TOME and GOST

The Table of Multiple Effects (TOME) holds statements of form:

```
(fn arg1 arg2 ...) = value at time-point
```

The Goal Structure Table (GOST) holds statements of form:

[2] All numerical constraint values in O-Plan are held as such triples.

```
condition-type (fn arg1 arg2 ...) = value at time-point
                                    from contributor-list

where contributor-list is a set of pairs of format:
(time-point . method-of-satisfaction-of-condition)
```

Effects and conditions are kept in a simple pattern directed lookup table as in Nonlin [35]. It is intended that the *Clouds* mechanism [37], which was used in the first version of O-Plan for efficiently manipulating large numbers of effects and their relationship to supporting conditions, will be used in the later versions of O-Plan in due course.

6.1.3 Plan State Variables

O-Plan can keep restrictions on plan state variables without necessarily insisting that a definite binding is chosen as soon as the variable is introduced to the Plan State.

6.1.4 Resource Utilisation Table

The Resource Utilisation Table holds statements of form:

```
set/+/- {resource <resource-name> <qualifier> ...} = <value>
                                                  at <time-point>
```

The statement declares that the particular resource is set to a specific value or changed by being incremented or decremented by the given value at the indicated time point. There can be uncertainty in one or both of the value and the time point which are held as min/max pairs [3]. However, this is a simplification in that a time point is a more complex structure that contains min and max values.

Task Formalism resource usage specifications on actions are used to ensure that resource usage in a plan stays within the bounds indicated. There are two types of resource usage statements in TF. One gives a *specification* of the **overall** limitation on resource usage for an activity (over the total time that the activity and any expansion of it can span). The other type describes actual resource *utilisation* **at** points in the expansion of a action. It must be possible (within the min/max flexibility in the actual resource usage statements) for a point in the range of the sum of the resource usage statements to be within the overall specification given. The Resource Utilisation Table manages the actual resource utilisation **at** points in the plan.

6.1.5 Agenda

The agenda for the current plan state gives details of processing which remains to be done in order that this plan state can be considered to have achieved its task. This defines the *pending*

[3] O-Plan numerical values are held as a triple with a numerical minimum, maximum and a *projected* value.

decisions which remain. The agenda entries each refer to *flaws* in the plan state which require further processing. Each flaw corresponds on a one-to-one basis to a knowledge source name which can process the relevant agenda entry.

An alternatives agenda of plan states other than the current one, which can be considered if this plan state is unsuitable to achieve the task, is kept by the Controller via the Alternatives Handler Support Module. Formally, all possible Plan States known to the Alternatives Handler, including the current plan state, should be considered as the "state" of the agent.

6.2 Planning Knowledge Sources

The O-Plan architecture is specialised into a planning agent by including a number of knowledge sources which can alter the Plan State in various ways. The planning knowledge sources provide a collection of *plan modification operators* which define the functionality of the planning agent beyond its default O-Plan architecture properties (essentially limited to communication capabilities by default).

The planning knowledge sources in the current version of the O-Plan planner are:

- KS_SET_TASK a knowledge source to set up an initial plan state corresponding to the task request from the task assignment agent.
- KS_EXPAND a knowledge source to expand a high level activity to lower levels of detail.
- KS_CONDITION a knowledge source to ensure that certain types of condition are satisfied. This is normally posted by a higher level KS_EXPAND.
- KS_ACHIEVE a knowledge source posted by KS_EXPAND for achieve conditions.
- KS_OR a knowledge source to select one of a set of possible alternative linkings and plan state variable bindings. The set of alternative linkings and bindings will have been created by other knowledge sources such as KS_CONDITION earlier.
- KS_BIND a knowledge source used to select a binding for a plan state variable in circumstances where alternative possible bindings remain possible.
- KS_POISON_STATE a knowledge source used to deal with a statement by another knowledge source that the plan state is inconsistent in some way or cannot lead to a valid plan (as far as that knowledge source is aware).
- KS_USER a knowledge source activated at the request of the user acting in the role of supporting the planning process (Planner User Role). This is used at present to provide a menu to browse on the plan state and potentially to alter the priority of some choices.

In addition, the default knowledge sources available in any O-Plan agent are present and are as follows:

- KS_INIT Initialise the agent.

- KS_DOMAIN Call the Domain Information (normally TF) compiler to alter the Domain Information available to the agent.
- KS_EXTRACT_RIGHT Extract a plan patch for passing to the subordinate agent to the 'right' of this agent - i.e the execution agent.
- KS_EXTRACT_LEFT Extract a plan patch for passing to the superior agent to the 'left' of this agent - i.e the task assignment agent.
- KS_PLANNER_FINISHED is used to inform the task assignment process that the planner has completed its task.
- KS_PATCH Merges a plan patch on an input event channel into the current plan state. In fact, in the current implementation, there is no use made of KS_PATCH.

It is intended that communication between the three agents in the O-Plan system (task assigner, planner and execution system) will respect the philosophy of communication via plan patches and that the KS_EXTRACT_LEFT, KS_EXTRACT_RIGHT and KS_PATCH knowledge sources will be the only ones which will make use of the event channels directly.

6.3 Use of Constraint Managers to Maintain Plan Information

O-Plan uses a number of *constraint managers* to maintain information about a plan while it is being generated. The information can then be used to prune the search (where plans are found to be invalid as a result of propagating the constraints managed by these managers) or to order search alternatives according to some heuristic priority. These managers are provided as a collection of *support modules* which can be called by knowledge sources to maintain plan information.

6.3.1 Time Point Network Manager (TPNM)

O-Plan uses a point based temporal representation with range constraints between time points and with the possibility of specifying range constraints relative to a fixed time point (time zero). This provides the capability of specifying relative and metric time constraints on time points. The functional interface to the Time Point Network (TPN), as seen by the Associated Data Structure (ADS) has no dependence on a particular representation of the plan. For example, rather than the simple 'before' relationship used in the O-Plan planner's plan state representation, a parallel project exploring temporal logics, reasoning mechanisms and representations for planning is investigating alternative higher level Associated Data Structure time relationships.

The Time Point Network is the lowest level of temporal data structure and consists of a set of points (and associated time constraints) each of which has an upper and lower bound on its temporal distance from:

1. other points in the network

2. a (user defined absolute) start time reference point

The points held in the TPN may be indirectly associated with actions, links and events, with the association being made at the Associated Data Structure level. This structure allows points to be retrieved and compared through a suitable module interface and with a minimum of overhead. The interface is important and reflects the *functionality* required of the TPN, and hides the detail. This ensures that we have no absolute reliance on points as a necessary underlying representation. As well as its use in the O-Plan activity orientated planner, the current TPNM has also been applied to large resource allocation scheduling problems in the TOSCA scheduler [6] where the number of time points was in excess of 5000 and the number of temporal constraints exceeded 3000.

Figure 3 and Figure 4 show the use of the TPN for applications involving task planning and resource allocation.

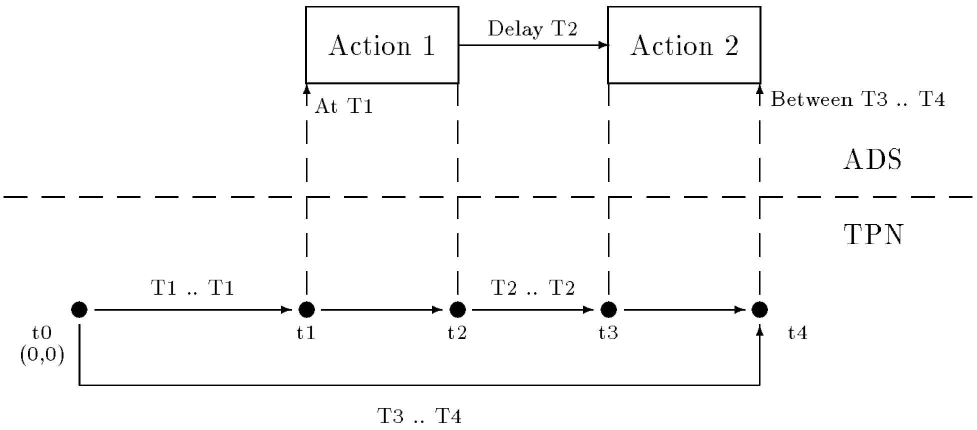

Figure 3: Example of activity planner at ADS using TPN

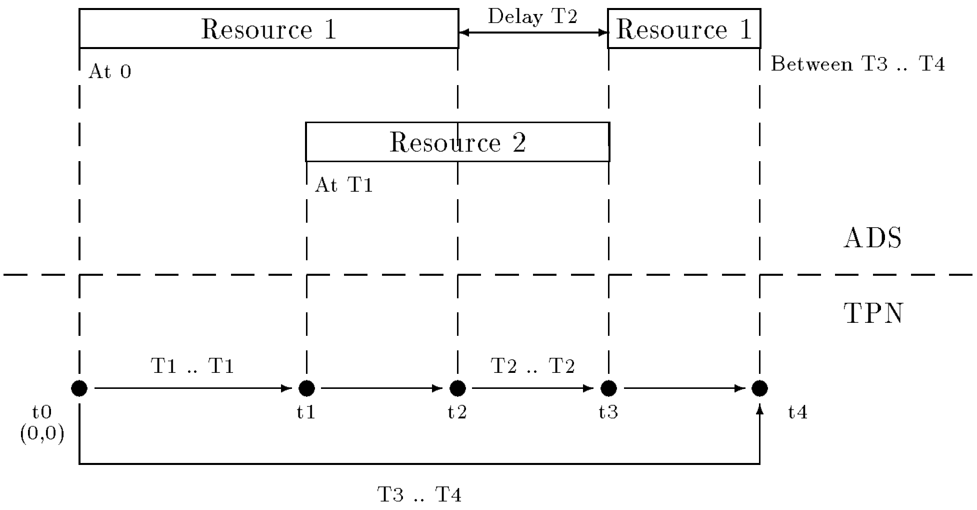

Figure 4: Example of resource allocation at ADS using TPN

6.3.2 TOME/GOST Manager (TGM)

The conflict-free addition of effects and conditions into the plan is achieved through the TGM, which relies in turn on support from the QA support module to suggest resolutions for potential conflicts.

6.3.3 Resource Utilisation Management (RUM)

O-Plan uses a Resource Utilisation Manager to monitor resource levels and utilisation. Resources are divided into different types such as:

1. Consumable: these are resources which are "consumed" by actions within the plan. For example: bricks, petrol, money, etc.
2. Re-usable: these are resources which are used and then returned to a common "pool". For example, robots, workmen, lorries, etc.

Consumable resources can be subcategorised as *strictly consumed* or may be *producible* in some way. Substitutability of resources one for the other is also possible. Some may have a single way mapping such as money for petrol and some can be two way mappings such as money for travellers' cheques. Producible and substitutable resources are difficult to deal with because they *increase* the amount of choice available within a plan and thus *open up* the search space.

The current O-Plan Resource Utilisation Manager uses the same scheme for strictly consumable resources as in the original O-Plan. However, a new scheme based on the maintenance of optimistic and pessimistic resource profiles with resource usage events and activities tied to changes in the profiles is now under study.

6.3.4 Plan State Variables Manager (PSVM)

The Plan State Variable Manager is responsible for maintaining the consistency of restrictions on plan objects during plan generation. O-Plan adopts a least commitment approach to object handling in that variables are only bound as and when necessary. For example, in a block stacking problem, moving block A to block B means that it is necessary to consider the object which A was previously on top of and from which it was moved. This object is introduced as a plan state variable whose value will be bound as and when necessary. The first version of O-Plan used a separate agenda to hold variable binding agenda entries. This scheme proved to be difficult to use due to the number of constraints which were built up between agenda entries and within agenda entries. The constraints were specified as:

- **Sames:** This specifies that this plan state variable should be the same as another plan state variable
- **Not-Sames:** This specifies that this plan state variable should not be the same as another plan state variable

- **Constraint-list:** This specifies a list of attributes which the value to which the plan state variable is bound must have. For example, it must be green, hairy and over 5ft tall.

The Plan State Variables Manager within the Database Manager (DM) maintains an explicit "model" of the current set of plan state variables (PSV).

When a Plan State variable (PSV) is created by the planner the Plan State Variables Manager creates a plan state variable name (PSVN), plan state variable body (PSVB) and a set list from which a value must be found. For example, the variable could be the colour of a spacecraft's camera filter which could be taken from the set (**red green blue yellow opaque**). A plan state variable must have an enumerable type and thus cannot be, for example, a real number. The PSVB holds the **not-sames** and **constraint-lists** and is pointed to by one or more PSVNs. This allows easier updating as new constraints are added and PSVBs are made the same (are collapsed to a single PSVB). Two or more PSVBs can be collapsed into a single PSVB if all of the constraints are compatible. e,g. the **not-sames** and **constraints-list**. A PSVN pointing to a collapsed PSVB is then redirected to point at the remaining PSVB. This scheme is a lot more flexible than the previous "sames" scheme as it allows triggers to be placed on the binding of PSV's (e.g. do not bind until the choice set is less than 3) and allows variables which are creating bottlenecks to be identified and if necessary restricted or bound.

6.4 Support Mechanisms in O-Plan

As well as the managers referred to above, a number of other support routines are available for call by the Knowledge Sources of O-Plan. The main such support mechanisms which have been built into the current O-Plan Planner include:

- **Question Answerer** (QA)
  The Question-Answering module is the core of the planner and must be both efficient and able to account for temporal constraints. QA supports the planner in satisfying and maintaining conditions in the plan in a conflict-free fashion, suggesting remedies where possible for any interactions detected. QA as implemented in O-Plan is an efficient procedural interpretation of Chapman's Modal Truth Criteria [8], which was distilled from QA in Nonlin [35]. QA provides support for the TGM in the system, and is supported in turn by another low level module, the Graph Operations (GOP)

- **Graph Operations Processor** (GOP)
  The GOP is a software implementation of a graph processor, providing efficient answers to ordering related questions within the main plan (represented by a graph). The GOP works within temporally ordered, as well as partially ordered, activities in the graph.

- **Contexts**
  All data within the O-Plan plan state can be "context layered" to provide support for alternatives management and context-based reasoning. The support module provides an efficient way of context layer any data structures accessor and updator function in Lisp. This is particularly useful for the underlying content addressable database in the system: O-Base.

- **O-Base**
  This database support module supports storage and retrieval of entity/relationship data with value in context. This model allows for retrieval of partially specified items in the database.

In addition, there are support modules providing support for the User Interface, Diagnostics, Instrumentation, etc., and there are others which still need further development (e.g., variable transaction management).

6.5 Alternatives Handler

There is an additional support module capability in O-Plan which is used by the Controller. This provides support for handling alternative plan states within an O-Plan agent.

If any stage of a knowledge source finds that it has alternative ways to achieve its task, and it finds that it cannot represent all those alternatives in some way within a single plan state, then the controller provides support to allow the alternatives that are generated to be managed. This is done by the knowledge source telling the controller about all alternatives but one favoured one and asking for permission to continue to process this (by the equivalent of a stage check). This reflects the O-Plan search strategy of *local best, then global best*. A support routine is provided by the controller to allow a knowledge source writer to inform the controller of all alternatives but the selected one.

A knowledge source which cannot achieve its task or which decides that the current plan state is illegal and cannot be used to generate a valid plan may terminate and tell the controller to poison the plan state. O-Plan will normally initiate consideration of alternative plan states by an exchange between the controller and the alternatives handler. A new current plan state will be selected and the planning process will be allowed to continue. Concurrently running knowledge sources working on the old (poisoned) plan state will be terminated as soon as possible (at the next stage boundary) as their efforts will be wasted.

As well as having the existing system's option to explore alternative plan states, future versions of O-Plan will consider ways to *unpoison* a plan state by running a nominated *poison handler* associated with the knowledge source that poisoned the plan state or with the *reason* for the plan state poison.

6.6 Implementation as Separate Processes

In the UNIX and Common Lisp based implementation of O-Plan the main managers and knowledge platforms are implemented as separate pseudo-processes running in a round-robin manner. One advantage of this approach is that knowledge sources can be run in pseudo-parallel with one another, and that external events can be processed by the Interface Manager (the manager in charge of all interaction, event handling, diagnostic handling and instrumentation) as they occur. The implementation could be moved to a true multi-processors system which would take into account the multi-tasking aspects very easily. It would then be possible to measure the reaction time performance of the system. This is measured by the time taken to post an agenda entry by the Event Handler and it being picked up by the agenda manager once triggered. The cycle time performance of the system is measured by the reaction time plus the time to assign the agenda entry to a knowledge source and have it run to completion.

7 O-Plan Task Assigner

In the current implementation of O-Plan, task assignment is a simple process with a menu of options available to the user. Communication between the task assignment agent and the planning agent of O-Plan does not currently reflect our intentions of communication via plan patches.

A menu of choices is provided:

- Initialise Planner
- Input TF (via pop-up menu of TF files available)
- Set Task (via pop-up menu of tasks available in current TF file)
- Add to Task (via a pop-up menu and input of the selected action from those available in the current TF)
- Plan View
- World View (at nominated node)
- Replan
- Execute Plan
- Quit

The task assignment process maintains the set of open command choices depending on the current status of the planning agent (whether it has been given domain information, set a specific task or is currently planning or has already generated a complete plan).

O-Plan will eventually support clear *authority* models which will allow a task assigner to authorise planning (on parts of a plan) to a given level or the execution of (parts of) a chosen plan. At the moment the authorities are driven from the menu of choices provided only.

The planner views the task assignment process as if it was a full O-Plan agent and takes requirements and commands in the form of events from the task assigner. The planner also packages its responses to the task assigner in the form of simplified events.

8 O-Plan Execution System

One of the aims of the O-Plan project is to investigate the issues involved in linking an intelligent planner with a remote execution agent. In order to investigate these issues a version of the O-Plan architecture has been configured to act as an execution agent. To configure O-Plan as an execution agent required a new set of knowledge sources to be defined which allowed the system to *follow* a plan rather then generate one.

The O-Plan execution agent accepts a "plan fragment" from the planner (this is created through the use of a knowledge source KS_EXECUTE in the planner) together with a set of monitoring instructions specifying how the actions of the plan should be monitored. The plan fragment consists of:

1. the plan specified as a partially-order network of activities
2. the TOME, GOST and temporal information built up during plan generation
3. the attachment point to be used by the execution monitor

The message is received by the left input guard of the execution agent's Event Handler and converted to an agenda entry. When the agenda entry is processed it causes the knowledge source KS_BREAKUP to be run in the execution agent. KS_BREAKUP posts the appropriate entries to the Diary Manager of the execution agent. The Diary Manager is set up to initiate triggers at the appropriate time. When triggered, the agenda entry is added to the triggered agenda list to await the availability of a knowledge source platform on which to run. In order to simulate the execution of the plan a World Process has been provided which allows the user a limited amount of interaction with the plan as it is being executed. For example, removing events such as the start of an action, delaying the completion of an action beyond its proposed start or end time.

The work to date on the execution agent within the O-Plan architecture is only at a very simple level and has mostly been concerned with ensuring that the communication capabilities are present to address issues of inter-agent plan fragment passing. Further work to characterise the requirements for and capabilities of a reactive execution agent have been undertaken [27] and an associated research project is now underway to explore how the O-Plan architecture can support these requirements.

9 O-Plan User Interface

9.1 Planner User Interface

AI planning systems are now being used in realistic applications by users who need to have a high level of graphical support to the planning operations being considered. In the past, our AI planners have provided custom built graphical interfaces embedded in the specialist programming environments in which the planners have been implemented. It is now important to provide interfaces to AI planners that are more easily used and understood by a broader range of users. We have characterised the user interface to O-Plan as being based on two *views* supported for the user. The first is a *Plan View* which is used for interaction with a user in planning entity terms (such as the use of PERT-charts, Gantt charts, resource profiles, etc). The second is the *World View* which presents a domain-orientated view or simulation of what could happen or is happening in terms of world state.

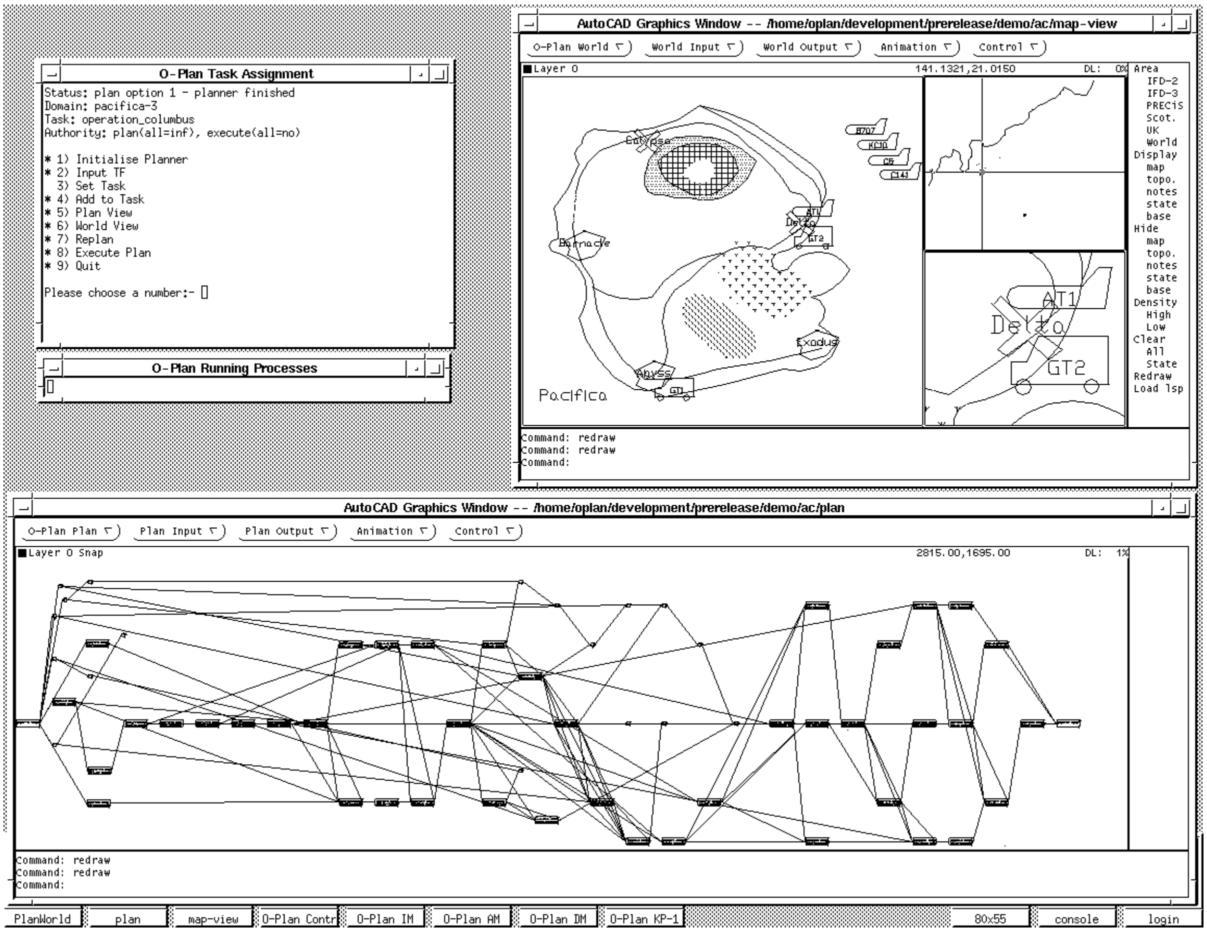

Figure 5: Example Output of the AutoCAD-based User Interface

Computer Aided Design (CAD) packages available on a wide range of microcomputers and engineering workstations are in widespread use and will probably be known to potential planning system users already or will be in use somewhere in their organisations. There could be benefits to providing an interface to an AI planner through widely available CAD packages so that the time to learn an interface is reduced and a range of additional facilities can be provided without additional effort by the implementors of AI planners.

Some CAD packages provide facilities to enable tailored interfaces to be created to other packages. One such package is AutoCAD [4], [32] - though it is by no means unique in providing this facility. AutoCAD provides AutoLISP, a variant of the Lisp language, in which customised facilities may be provided [5], [33]. This is convenient for work in interfacing to AI systems as workers in the AI field are familiar with the Lisp language. However, the techniques employed would apply whatever the customisation language was.

We have built an interface to the Edinburgh AI planning systems which is based on AutoCAD. A complete example of the interface has been built for two different domains:

- Space Platform Building
  O-Plan Task Formalism has been written to allow the generation of plans to build various types of space platform with connectivity constraints on the modules and components.
- Non-combatant Evacuation Operation (NEOs)
  O-Plan Task Formalism has been written to model the evacuation of nationals from the mythical island of Pacifica in which unrest has broken out.

A domain context display facility has been provided for both applications through the use of AutoLISP. This allows the state of the world following the execution of any action to be visualised through AutoCAD. Means to record and replay visual simulation sequences for plan execution are provided.

A sample screen image is included in Figure 5. There are three main windows. The planner is accessible through the Task Assignment window to the top left hand corner which is showing the main user menu. The planner is being used on a Pacifica NEO task and has just been used to get a resulting plan network. In the *Plan View* supported by O-Plan, this has been displayed using the *Load Plan* menu item in the large AutoCAD window along the bottom of the screen. Via interaction with the menu in the AutoCAD window, the planner has been informed that the user is interested in the context at a particular point in the plan - the selected node is highlighted in the main plan display. In the *World View* supported by O-Plan, the planner has then provided output which can be visualised by a suitable domain specific interpreter. This is shown in the window to the top right hand corner of the screen where a map of the topology of the island, information on it's capital Delta and its "position" relative to other countries are simultaneously displayed.

The O-Plan Plan View and World View support mechanisms are designed to retain independence of the actual implementations for the viewers themselves. This allows widely available tools like AutoCAD to be employed where appropriate, but also allows text based or domain specific viewers to be interfaced without change to O-Plan itself. The specific viewers to be used for a domain and the level of interface they can support for O-Plan use is described to O-Plan via the domain Task Formalism (TF). A small number of *viewer characteristics* can be stated. These are supported by O-Plan and a communications language is provided such that plan and world viewers can input to O-Plan and take output from it.

Sophisticated Plan and World Viewers could be used in future with O-Plan. We believe that time-phased tactical mapping displays of the type used in military logistics can be used as a World Viewer. We have also considered interfaces to a Virtual Reality environment we term PlanWorld-VR.

9.2 System Developer Interface

When O-Plan is being used by a developer, it is usual to have a number of windows active to show the processing going on in the major components of the planner. There is a small window acting as the task assignment agent with its main O-Plan menu. There is then separate window for the Interface Manager (IM), through which the user can communicate with other processes and through which diagnostic and instrumentation levels can be changed. The Agenda Manager/Controller (AM), the Database Manager (DM) and the Knowledge Source Platform(s) (KP) then have their own windows. Further pop-up windows are provided when viewing the plan state graphically or when getting detail of parts of the plan, etc.

A sample developer screen image is shown in figure 6.

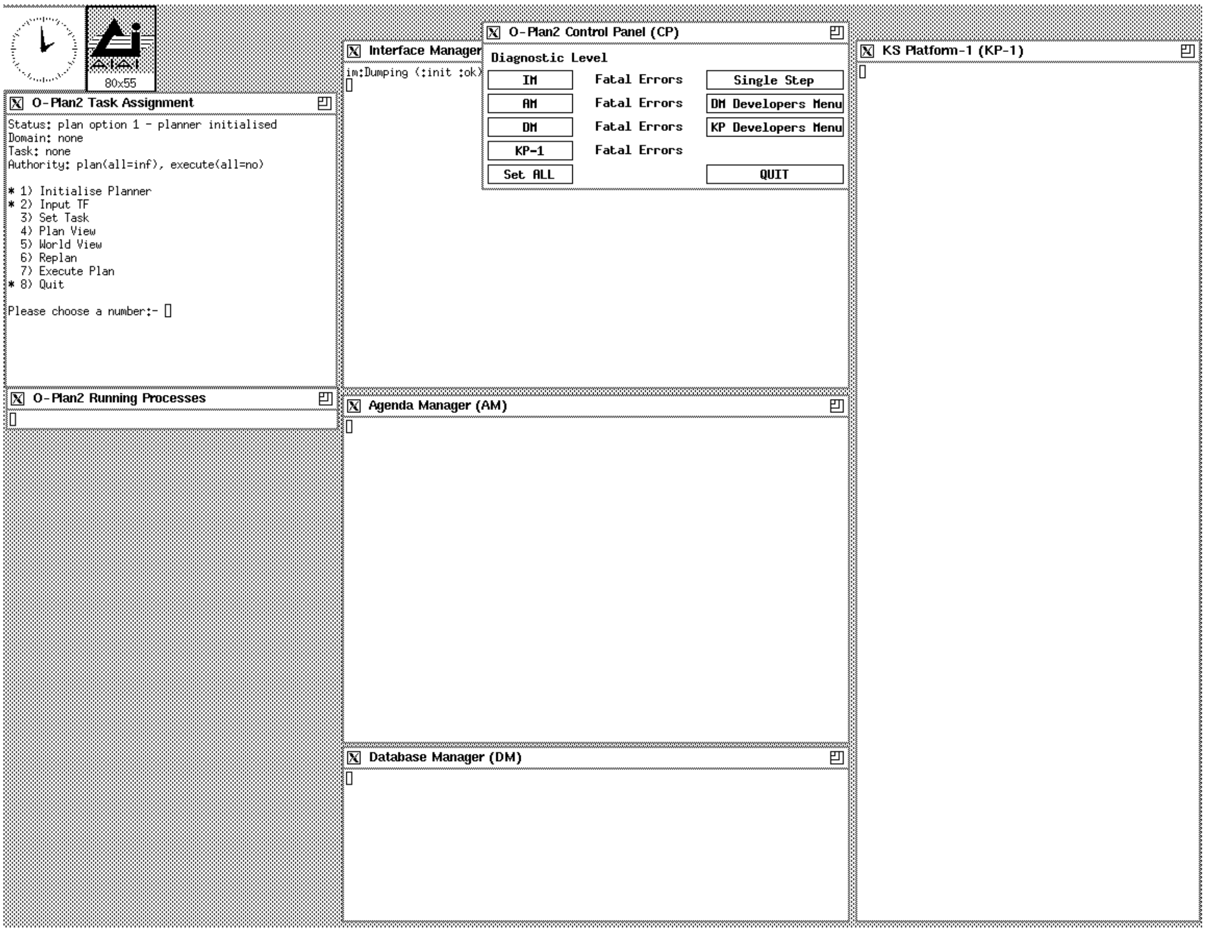

Figure 6: Example Developer Interface for the O-Plan Planning Agent

9.3 O-Plan User Roles

User interaction with O-Plan can occur for a variety of purposes. Various *roles* of an user interacting with O-Plan are defined and are supported in different ways within the system. We consider the identification of the different roles to be an useful aid to guide future user interface support development.

9.4 Domain Expert Role

A single user responsible for defining the bounds on the application area for which the system will act. The domain expert user may directly or indirectly specify O-Plan Task Formalism to define the domain information which the planner will use.

9.5 Domain Specialist Role

One or more domain specialists may define information at a more detailed level within the framework established by the domain expert. Once again, the domain specialist may directly or indirectly specify O-Plan Task Formalism to provide the detailed domain information which the planner will use.

9.6 Task Assignment User Role

The command user interacts only with the Task Assignment Agent to provide user requirements or commands. This user is responsible for the selection of the task which the system will try to carry out. The current system provides a menu which allows for a domain to be selected and for a choice to be made from the task schemas within the Task Formalism for that domain. Future management of alternative plan options, plan analysis support and the provision of authority to plan or execute the plan are to be supported at this level.

9.7 Planner User Role

The planner user is the user responsible for ensuring that a suitable plan is generated to carry out the given task. This may involve the selection of alternatives, the restriction of options open to the planner and browsing on the emerging and final plan to ensure it meets the task requirements set by the task assignment user. Since the planner user can perform decision making in the planner agent, the planner user is supported by a knowledge source called KS-USER. This knowledge source can be added to the agenda for the current plan state on demand (via an user request). Since the KS-USER knowledge source normally has high priority, it will normally be called as soon as possible. The KS-USER knowledge source activation has access to the current plan state to allow for decisions on user intervention to depend on the contents of the current plan state.

9.8 Execution System Watch/Modify Role

The user may interact with the execution system to watch the state of execution of the plan and perhaps even to modify the behaviour of the execution system.

9.9 World Operative

Any users who are required to carry out activities in the world (acting as an *effector*) or who report aspects of the environment (acting as a *sensor*).

9.10 World Interventionist

If a world simulation is being used to demonstrate the O-Plan execution system, an user may be given facilities to intervene in the world simulation to cause events to happen and problems to occur such that execution of plans in uncertain situations can be tested.

9.11 User Support to Controller Role

The user may assist an O-Plan agent's controller to decide which knowledge source to dispatch to a waiting knowledge source platform or to decide on when to direct a running knowledge source to stop at a stage boundary.

9.12 User Support to Alternatives Handler

The user may assist an O-Plan agent's Alternatives Handler to decide which alternative to select when one is needed or to suggest an alternative is tried rather than continuing with the current plan state.

9.13 System Developer Role

The system developer has access to the diagnostic interface of the system running within each agent. This is supported by the Developer Diagnostic Interface of each O-Plan agent. The behaviour of this interface can be set and modified via a Control Panel which allows for the setting of levels of diagnostics using buttons, etc.

9.14 System Builder

The O-Plan Agent Architecture is intended to be sufficiently flexible to allow a system builder to create a system with defined behaviour. To this end, it is possible to have radically different plan state data structures, knowledge sources, domain information and controller strategies. For example, the O-Plan Architecture already has been used to provide a Manufacturing Scheduling System which uses a resource orientated representation for the plan state rather than the action orientated plan representation in the O-Plan Planner. This scheduler, called TOSCA (The Open SCheduling Architecture), also has different knowledge sources than those used in the O-Plan Planner.

10 Performance Issues

O-Plan has been designed in such a way that components can be improved within the specifications adopted. Performance issues have been considered in establishing the interfaces and protocols used. The current prototype includes only very simple implementations of some of the components.

10.1 Architecture Performance

An early consideration for the O-Plan project was to ensure that the agent orientated design would not introduce overheads of computation which would be unacceptable. A number of designs for the multi-process structure required to support O-Plan were discussed. These included shared memory processes and processes which used a server for access to the shared data elements. At the time that these discussions were taking place there was little uniformity of handling concurrent processes in Common Lisp systems. Tests were conducted with complete O-Plan systems which had only a trivial knowledge source included. These were implemented in versions of Common Lisp and the C language.

Two measures were tested:

Agent Latency This measure shows the minimum time for an event at the agent boundary to be noted by the Event Handler, communicated to the Agenda Manager/Controller, triggered (where the trigger is null), communicated to a Knowledge Source Platform (which is waiting and idle) and an appropriate Knowledge Source activated on the platform to process the agenda entry corresponding to the event.

Agent Cycle Time This measure shows the minimum time for a Knowledge Source to post an agenda entry back to the Agenda Manager/Controller and terminate its processing, for the agenda entry to be triggered (where the trigger is null), communicated to a Knowledge Source Platform (which is waiting and idle) and an appropriate Knowledge Source is activated on the platform to process the agenda entry. This corresponds to a single cycle of the agent internally when only one Knowledge Source Platform is available.

Our main performance goal at the outset of the O-Plan research was to allow the generation of a plan with a few hundred nodes, which we judge would require 500-1000 agenda cycles, in about 3 minutes. Subjectively, we judged that 3 minutes was an acceptable period for a user to sit awaiting a result in our demonstrations. Recent versions of O-Plan are exceeding this performance goal.

10.2 Constraint Manager and Support Routine Performance

Our experience with earlier AI planners such as Nonlin and the first version of O-Plan was that a large proportion of the time of a planner could be spent in performing basic tasks on the plan network (such as deciding which nodes are ordered with respect to others) and in reasoning about how to satisfy or preserve conditions within the plan. Such functions have been modularised and provided as Constraint Managers (Graph Operations Processor, Time Point Network Manager, TOME/GOST Manager, etc) and Support Routines (Question Answering, etc) in O-Plan to allow for future improvements and replacement by more efficient versions.

11 Modularity, Interfaces and Protocols

This section provides a summary of contribution of the O-Plan project towards the identification of separable support modules, internal and external interface specifications and protocols governing processing behaviours which are relevant to an AI planning system.

11.1 Components

The O-Plan project has sought to identify modular components within an AI command, planning and control system and to provide clearly defined interfaces to these components and modules.

The main components are:

1. Domain Information - the information which describes an application domain and tasks in that domain to the planner.
2. Plan State - the emerging plan to carry out identified tasks.
3. Knowledge Sources - the processing capabilities of the planner (*plan modification operators*).
4. Support Modules - functions and constraint managers which support the processing capabilities of the planner and its components.
5. Controller - the decision maker on the *order* in which processing is done.

11.2 Support Modules

Support Modules may either be Constraint Managers or other types of modules intended to provide efficient support to a higher level where decisions are taken. They should not take any decision themselves. They are intended to provide complete information about the constraints they are managing or to respond to questions being asked of them by he decision making level. The support modules normally act to manage information and constraints in the plan state. Examples of Support Modules in O-Plan include:

- Effect/Condition (TOME/GOST) Manager including Question Answering (QA)
- Resource Utilisation Manager
- Time Point Network Manager
- Object Instantiation (Plan State Variables) Manager
- Alternatives Handler
- Instrumentation

- Monitors for output messages, etc.

A guideline for the provision of a good support module in O-Plan is the ability to specify the calling requirements for the module in a precise way (i.e. the *sensitivity rules* under which the support module should be called by a knowledge source or from a component of the architecture).

11.3 Protocols

In addition, a number of external interface specifications and protocols for inter-module use have been established. Only first versions of these interfaces have been established at present, but we believe that further development and enhancement of the planner can take place through concentrating effort on the specification of these interfaces. This should greatly assist the process of integrating new work into the planning framework.

The protocols for regulating the processing conducted by components of O-Plan are:

1. *Knowledge Source Protocol*
   for the ways in which a Knowledge Source is called by the Controller, can run and can return its results to the Controller and for the ways in which a Knowledge Source can access the current plan state via the Database Manager.
2. *KS_USER Protocol*
   for the ways in which the user (in the role of *Planner User*) can assist the planning system via a specially provided knowledge source.
3. *Inter-agent Communications Protocol*
   controls the way in which the KS_EXTRACT_LEFT, KS_EXTRACT_RIGHT and KS_PATCH Knowledge Sources operate and may use the Interface Manager's support routines which control the agent's input and output event channels (LEFTIN, LEFTOUT, RIGHTIN and RIGHTOUT).

11.4 Internal Support Facilities

The internal support provided within the planner to assist a Knowledge Source writer includes:

1. *Knowledge Source Framework* (KSF)
   is a concept for the means by which information about a Knowledge Source can be provided to an agent. This will ensure that a suitable Knowledge Source Platform is chosen when a Knowledge Source is run inside an agent. It will also allow a model of the capabilities of other agents to be maintained. The KSF will also allow for triggers to be set up for releasing the Knowledge Source for (further) processing. It will allow a description of the parts of a plan state which can be read or altered by each stage within the knowledge source (to allow for effective planning of concurrent computation and data base locking in future).

2. *Agenda Trigger Language*
   gives a Knowledge Source writer the means by which a computation can be suspended and made to await some condition. The conditions could relate to information within the plan, for external events or for internally triggered Diary events. O-Plan provides a limited number of triggers of this kind, but we anticipate this being expanded significantly in future.

3. *Controller Priority Language*
   allows the input of guidance rules for the ordering decisions taken by the O-Plan Controller on which triggered agenda entries to process next. Currently, only simple numerical priorities are used to guide the controller.

The following sections give further details of these facilities.

11.4.1 Knowledge Source Framework (KSF)

The KSF allows information about a knowledge source to be provided to the O-Plan architecture. A KSF description of a knowledge source is the mechanism by which a new capability is declared to an O-Plan agent.

The KSF gives the following details:

- the name of the knowledge source. This is used by other knowledge sources to indicate that they want to call this capability by posting suitable agenda entries as they run. It is also used for nominating a knowledge source to deal with an event.
- parameters match description. This is used to restrict the legal parameters that may be passed to the knowledge source.
- agenda posting/information field match description. This is used to restrict the legal entries for the posting/information field accepted back by a knowledge source (used for information temporarily kept between stages of a suspended knowledge source). It can also be used to ensure that any modification of this posting information field not done by the knowledge source itself is verified as being acceptable to the knowledge source itself.
- stages information in the form
  - <stage number> <trigger> -> <ks stage function> <locking information>
  - the stage number need only be given if there is more than one stage.
  - The <trigger> description can be composed from the O-Plan Trigger Language - which itself will evolve over time.
  - the <ks stage function> is the actual procedure that will be called to implement the current knowledge source stage.
  - <locking information> is provided to define whether this stage needs the plan state in READ mode or WRITE mode. If not provided, the default assumption is that

the stage is a READ mode stage and that all effects of the stage are created by communication with the controller (normally also saving information in the information field of the agenda record when the knowledge source terminates if asked to do so at the stage end). It is also possible to give information about the specific parts of the plan state that can be READ by or WRITTEN to by this stage to allow for selective locking strategies to be explored in future versions of O-Plan.

- controller priority function. To provide heuristic guidance to the controller based upon the overall information in the agenda record nominating this knowledge source. This will only be applied to triggered agenda entries. It may use Branch 1 and Branch N information [10] in an agenda entry to provide heuristic guidance to the controller.
- plan state poison handler. The function to be called whenever *this* knowledge source terminates with a request to poison the plan state (i.e. when this knowledge source thinks that the plan state is inconsistent and that it cannot recover from the problem itself).

The KSF is used to build a capability library for the agent and to define the event information that may be passed by the guards on the external event channels of the agent. Extensions to the KSF will be needed as further refinement of the agent properties of an O-Plan system are defined.

11.4.2 Agenda Trigger Language

An agenda entry can be set to await a *trigger*. This trigger can relate to information in the plan state, to external events, etc. The facility can be used by a Knowledge Source writer to allow Knowledge Source processing to be suspended at a *stage* boundary and made to await the trigger condition before resumption. The responsibility for reactivating the computation is taken by the O-Plan system using facilities within the Database Manager.

The trigger can be composed from the O-Plan Trigger Language - which itself will evolve over time. Triggers will be composed to form a boolean function.

Example triggers available within the O-Plan planner are:

- "always triggered".
- dependencies on the plan state to be selected including:

  wait for a suitable effect matching some specification.

  wait for a fully instantiated binding for a Plan State Variable.
- links to events triggers at a specific time via the O-Plan Diary.
- empty agent agenda.

11.4.3 Controller Priority Language

Currently, the O-Plan Controller selects agenda entries based on a numerical priority which is simply a statically computed measure of the priority of outstanding agenda entries in a plan state. Our aim for the future is to provide a rule based controller which can make use of priority information provided in the form of rules in an O-Plan Controller Priority Language. This concept will allow us to clarify our ideas on what information should govern controller ordering decisions. Domain information linking to generic Controller Priority Language statements which can affect the controller decisions is likely to be considered as part of a link between Task Formalism (TF) and the operation of the Controller.

11.5 External Interfaces

The external interfaces provided by the planner are:

1. *Task Formalism* (TF) as the language in which an application domain and the tasks in it can be expressed to the planner.
2. *Plan View User Interface* which allows for domain specific plan drawing and interaction to be provided.
3. *World View User Interface* which allows for domain specific world state simulation facilities and interaction to be provided.
4. *External System Interface* provided by TF **compute conditions** for ways in which external data bases, modelling systems, simulations, CAD packages, geographical information systems, route finders, look-up tables, etc., can be used and for ways in which these external systems can access plan information and provide qualifications on the continued validity of their results where appropriate.

12 Related Projects

O-Plan is one of a set of projects at Edinburgh grouped under the title of EUROPA (Edinburgh University Research into Open Planning Architectures). The combined research of these projects cover issues in Knowledge Based Planning and Scheduling and are anchored around the two main, long term research projects of O-Plan and TOSCA (The Open SCheduling Architecture). TOSCA is a variant of the same ideas applied to the area of operations management in the factory (job shop) environment [6]. TOSCA employs appropriate knowledge sources for its domain of application (*e.g.* resource assignment, bottleneck analysis) which operate on an emerging schedule state, similar to the notion of the plan state mentioned above.

Another project is investigating temporal representations for Planning and Scheduling to provides a more flexible representation of plans and schedules based on temporal logics. Planning and Scheduling are often considered to be similar activities, though the reality is that they are quite different. However there is undoubtably a great deal of overlap, particularly with respect to resource handling. Our aim is to develop designs and architectures suited to both types of problem and to develop as much common ground as is possible. O-Plan plays a key role in this plan.

Work has been undertaken with the UK's Defence Research Agency into the problem of supporting staff in Scotland's main Search and Rescue Coordination Centre. The problem they face is coordinating a number of rescue assets (helicopters, search teams, lifeboats, etc) and monitoring the plans developed as the rescue progresses. The work has centred on the development of a generic approach (with appropriate tools) to assist in reliable capture of knowledge related to planning, scheduling and resource allocation. This approach will be validated through the production of a demonstration system for the the Search and Rescue Centre.

A student research project [27] investigated the requirements for a reactive execution agent and exploring the O-Plan architecture to meet the requirements.

13 Future Plans for O-Plan

The O-Plan project is continuing actively to develop in a number of ways.

The current O-Plan prototype now gives us a basis for a complete system for command specification, planning and execution control. A number of parts of the system are provided in a very simplistic way at present. It is intended that effort will be devoted to replacement of components in the current system with improved versions. Some components in the first version of O-Plan, for example (such as the "Clouds" data structures for helping with TOME and GOST management [37]), have yet to be re-implemented in the current prototype. Experiments with the integration of constraint managers or support modules provided by others for time point network management and for world modelling will be conducted. The extremely simple controller strategies used in the current implementation will be improved upon.

Further effort is required to clarify and further develop the O-Plan component and support module definitions, the protocols (such as the Knowledge Source protocol and the KS_USER protocol), external interfaces (such as TF and the external systems interface) and internal support facilities (such as the KSF and the agenda triggering language). Involvement with other projects and research groups will be sought to broaden our perspective during this further development.

A language KRSL is now being developed on the US ARPA/Rome Laboratory Knowledge-based Planning and Scheduling Initiative to act as a domain description language for command, planning, scheduling and control applications. It is envisaged that a pre-processor to take descriptions of application domains in KRSL and to create O-Plan TF from them will be investigated.

In future, we anticipate employing qualitative world modelling within the O-Plan planning agent as demonstrated by Drabble [12] in his Excalibur system (based on Nonlin). This will be used to model processes not under the control of the planner and to predict the impact of plans on the execution environment.

It is intended that communication between the three agents in the O-Plan system (task assigner, planner and execution system) will be brought fully into line with our philosophy on communication via plan patches and via the KS_EXTRACT_LEFT, KS_EXTRACT_RIGHT and KS_PATCH knowledge sources which are the only ones which should make use of the event channels directly.

In the near future a great deal of work will be carried out on the task assignment agent. We have a desire to improve the quality of the User Interface and the support available for the effective writing of domain information about an application (in TF), the specification and alteration of tasks set for the planner and execution system, and the maintenance of a user view of the state of planning, execution and the external world model. This will involve research related to clear *authority modelling* in the O-Plan architecture.

14 Bibliography

References

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

[2] Alvey Directorate (1987) Alvey Grand Meeting of Community Clubs. Available through IEE, Savoy Place, London.

[3] Arentoft, M.M., Parrod, Y., Stader, J., Stokes, I. & Vadon, H. *OPTIMUM-AIV: A Planning and Scheduling System for Spacecraft AIV* Telematics and Informatics Vol. 8, No. 4, pp. 239-252, Pergamon Press.

[4] AutoDesk AutoCAD Reference Manual, 1989.

[5] AutoDesk AutoLISP Reference Manual, 1989.

[6] Beck, H.A. *Constraint Monitoring in TOSCA*, in "Working Notes from the 1992 AAAI Spring Symposium on Practical Approaches to Scheduling and Planning", (eds. M.E.Drummond, M.Fox, A.Tate and M.Zweben), NASA Ames Research Center, AI Research Branch Technical Report FIA-92-17.

[7] Bell, C.E. and Tate, A. Using Temporal Constraints to Restrict Search in a Planner. Presented at the Third Workshop of the Alvey IKBS Programme's Special Interest Group on Planning, Sunningdale, Hants, January 1985. Also available as AIAI-TR-5.

[8] Chapman, D. Planning for conjunctive goals. *Artificial Intelligence Vol. 32, pp. 333-377, 1987.*

[9] Currie, K.W. and Tate, A. (1985) *O-Plan: Control in the Open Planning Architecture*, Proceedings of the BCS Expert Systems 85 Conference, Warwick, UK, Cambridge University Press.

[10] Currie, K.W. & Tate, A. O-Plan: the Open Planning Architecture, *Artificial Intelligence* Vol 51, No. 1, Autumn 1991, North-Holland.

[11] Daniel, L. (1983) *Planning and Operations Research* in Artificial Intelligence: Tools, Techniques and Applications (eds. O'Shea and Eisenstadt), Harper and Row, New York.

[12] Drabble, B. Planning and reasoning with processes. *Procs. of the 8th Workshop of the Alvey Planning SIG, The Institute of Electrical Engineers, November, 1988.* Full paper to appear in *Artificial Intelligence Journal*, 1992.

[13] Drabble, B. and Tate, A., *Using a CAD system as an interface to an AI Planner*, European Space Agency Conference of Space Telerobotics, European Space Agency, 1991, Noordwijk, Holland.

[14] Drummond, M. & Currie, K. Exploiting temporal coherence in nonlinear plan construction. *Procs. of IJCAI-89, Detroit.*

[15] Drummond, M.E., Currie, K.W. and Tate, A. (1988) *O-Plan meets T-SAT: First results from the application of an AI Planner to spacecraft mission sequencing*, AIAI-PR-27, AIAI, University of Edinburgh.

[16] Drummond, M.E., & Tate, A. (1992) *PLANIT Interactive Planners' Assistant - Rationale and Future Directions*, AIAI-TR-108, AIAI, University of Edinburgh.

[17] Fikes, R.E., Hart, P.E. and Nilsson, N.J. (1972) *Learning and Executing Generalized Robot Plans*, Artificial Intelligence Vol. 3.

[18] Georgeff, M. P. and A. L. Lansky (1986) *Procedural Knowledge*, in Proceedings of the IEEE, Special Issue on Knowledge Representation, Vol. 74, pp 1383-1398.

[19] Hayes, P.J. (1975) *A representation for robot plans*, IJCAI-75, Proceedings of the International Joint Conference on Artificial Intelligence, Tbilisi, USSR.

[20] Hayes-Roth, B. & Hayes-Roth, F. A cognitive model of planning. *Cognitive Science, pp 275 to 310, 1979.*

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

[22] Liu, B., Ph.D Thesis, *Knowledge Based Scheduling*, Edinburgh University, 1988.

[23] Malcolm, C. and Smithers, T. (1988) *Programming Assembly Robots in terms of Task Achieving Behavioural Modules: First Experimental Results*, in Proceedings of the Second Workshop on Manipulators, Sensors and Steps towards Mobility as part of the International Advanced Robotics Programme, Salford, UK.

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

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

[26] Nilsson, N.J. (1988) *Action Networks*, Proceedings of the Rochester Planning Workshop, October 1988.

[27] Reece, G.A. (1992) *Reactive Execution in a Command, Planning and Control Environment*, Ph.D Dissertation Proposal, Department of AI Discussion Document, The University of Edinburgh.

[28] Rosenschein, S.J., and Kaelbling, L.P. (1987) *The Synthesis of Digital Machines with Provable Epistemic Properties*, SRI AI Center Technical Note 412.

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

[30] Sadeh, N. and Fox, M.S., *Preference Propagation in Temporal/Capacity Constraint Graphs*, Computer Science Dept, Carnegie-Mellon University, 1988, Technical Report CMU-CS-88-193.

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

[32] Smith, J. and Gesner, R. (1989) Inside AutoCAD, New Riders Publishing Cp., Thousand Oaks, Ca.

[33] Smith, J. and Gesner, R. (1989) Inside AutoLISP, New Riders Publishing Cp., Thousand Oaks, Ca.

[34] Sridharan, N. Practical Planning Systems, *Rochester Planning Workshop, Rochester University, 1988.*

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

[36] Tate, A. (1984) *Planning and Condition Monitoring in a FMS*, Proceedings of the International Conference on Flexible Automation Systems, Institute of Electrical Engineers, London, UK.

[37] Tate, A. (1986) *Goal Structure, Holding Periods and "Clouds"*, Proceedings of the Reasoning about Actions and Plans Workshop, Timberline Lodge, Oregon, USA. (eds, Georgeff, M.P. and Lansky, A.), published by Morgan Kaufmann.

[38] Tate, A. & Drabble, B. O-Plan2: Choice Ordering Mechanisms in an AI Planning Architecture in Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, California, USA on 5-8 November 1990, published by Morgan-Kaufmann. Also updated with B.Drabble as AIAI-TR-86, AIAI, University of Edinburgh.

[39] Tate, A., Drabble, B. and Kirby, R., O-Plan2: an Open Architecture for Command, Planning and Control, in Intelligent Scheduling, (eds, M.Zweben and M.S.Fox), Morgan Kaufmann Publishers, Palo Alto, CA., USA, 1994.

[40] Tecknowledge, S.1 Product Description, Tecknowledge Inc., 525 University Avenue, Palo Alto, CA 94301. 1988.

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

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

[43] Wilkins, D.E. (1985) *Recovering from execution errors in SIPE*, Computational Intelligence Vol. 1 pp 33-45.

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

15 Abbreviations

ADS Associated Data Structure - the level of data structure in O-Plan at which a plan is represented. This is "associated" with an underlying Time Point Network (TPN).

AM O-Plan Agenda Manager - one of the main processes of the O-Plan system and the main part of the "Controller" which decides on what can be processed next in an O-Plan agent.

AT Agenda Table - used to represent outstanding activities for an O-Plan agent.

DM O-Plan Database Manager - one of the main processes of the O-Plan system which manages the plan state and gives access to it on behalf of other modules.

GOP Graph Operations Processor - a set of support routines in O-Plan used to process networks or graphs (especially the Time Point Network - TPN).

GOST Goal Structure Table - used to hold conditions associated with a plan and their method of satisfaction.

IM O-Plan Interface Manager - one of the main processes of the O-Plan system which manages inter-module, inter-agent and user communications.

KP O-Plan Knowledge Source Platform - one of the main processes of the O-Plan system on which Knowledge Sources can be run.

KS Knowledge Source - a computational capability in O-Plan.

KSF Knowledge Source Framework - a proposed language for describing an agent's capabilities (its Knowledge Sources).

MTC Modal Truth Criterion - another name adopted by other researchers for a process similar to Question Answering (QA).

PSV Plan State Variable - an object in a plan which is not fully defined.

PSVB Plan State Variable Body - the body associated with a Plan State Variable used in a plan and containing its constraints.

PSVM Plan State Variables Manager - the Constraint Manager in O-Plan which builds and looks after PSVs.

PSVN Plan State Variable Name - the name associated with a Plan State Variable used in a plan. Several PSV names can be associated with a single PSV body.

QA Question Answering - the O-Plan support routine which finds the ways in which a plan condition can be satisfied.

TC Temporal Coherence - a search ordering heuristic.

TD Trigger Detector - used to recognise when an O-Plan agent's outstanding agenda entries can be passed to the Agenda Manager (AM) for processing.

TF Task Formalism - the domain description language for the O-Plan planner.

TGM TOME/GOST Manager - the Constraint Manager in O-Plan used to look after effects and conditions.

TOME Table Of Multiple Effects - used to hold effects associated with the actions of a plan.

TPN Time Point Network - used to hold time points associated with a plan and constraints between these time points.

TPNM Time Point Network Manager - the Constraint Manager in O-Plan which builds and looks after the TPN.