The Emergence of “Standard” Planning and Scheduling System Components – Open Planning and Scheduling Architectures

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

Abstract.
As realistic systems are built and commercial toolkits are created for planning and scheduling applications, it becomes increasingly important that modularisation of these “standard” components is attempted. This can lead to re-usability, embeddability and improved implementation provision.
One early example of a standardised component is the Truth Criterion condition establishment algorithm which is now found at the heart of most activity based planning systems. The modularisation of this algorithm has led to an explosion of further development, empirical and theoretical study of this component. The provision of powerful constraint management languages and tools could lead to a rapid expansion of the benefits to be gained by identifying more standard components that can be combined and re-used in planning and scheduling systems.
O-Plan2 is a command, planning and control architecture which has an open modular structure intended to allow experimentation on or replacement of various components. This paper describes the modular structure of the system along with the internal and external interface languages and protocols which are being developed on the O-Plan2 project. The research is seeking to isolate functionality that may be generally required in a number of applications and across a number of different planning, scheduling and control systems.
This paper is intended to further discussions on the identification of suitable “standard” re-usable components in planning and scheduling systems.

1. Introduction

Three decades of planning and scheduling research in artificial intelligence has led to a number of "standard" approaches and components which are at the heart of the majority of systems. As realistic systems are built and commercial toolkits are created for planning and scheduling applications, it becomes increasingly important that modularisation of these "standard" components is attempted. This can lead to re-usability, embeddability and improved implementation provision.

One early example of a standardised component is the Truth Criterion condition establishment algorithm. Such a Truth Criterion is now found at the heart of most activity based planning systems. The modularisation of the capability to establish the truth of a given statement at a given point in a partially ordered network of time points in a partial plan has led to an explosion of further development, empirical and theoretical study of this component.

In order to benefit from advances in various technologies and to allow improved implementations of components to be used, we need to be able to recognise separable functions and capabilities within our planners and schedulers. By separating the processing capabilities at the *architecture* level of a planner or scheduler from the *plan or schedule representation* we can begin to address modularity issues of this kind.

Moves to provide powerful constraint management languages and tools could lead to a rapid expansion of the benefits to be gained by identifying more standard components that can be combined and re-used in planning and scheduling systems. This can allow time network management, management of the persistence of facts over time, resource management and other such constraint management by separate components provided by someone other than the original developer or integrator.

O-Plan [9] and O-Plan2 [20] are successors to Nonlin. O-Plan2 is a command, planning and control architecture being developed at the Artificial Intelligence Applications Institute of the University of Edinburgh. It has an open modular structure intended to allow experimentation on or replacement of various components without the need to change the majority of the overall system. This paper describes the modular structure of the system along with the internal and external interface languages which are being developed on the O-Plan2 project. In a number of cases, only very simple versions of the interfaces are supported in the current O-Plan2 system. However, even the early versions of such interfaces are proving useful to isolate functionality that may be generally required in a number of applications and across a number of different planning, scheduling and control systems.

This paper is intended to further discussions on the identification of suitable "standard" re-usable components in planning and scheduling systems.

2. A Successful "Standardisation" - The Truth Criterion in Planners

One early example of a standardised component is the Truth Criterion condition establishment algorithm. This was first described for the Nonlin planner in 1976 where it was called the QA algorithm [17]). Such a Truth Criterion is now found at the heart of most activity based planning systems. The modularisation of the capability to establish the truth of a given statement at a given point in a partially ordered network of time points in a partial plan has led to an explosion of further development, empirical and theoretical study of this component.

Nonlin used the partially ordered plan representation of NOAH [14], but brought this together with a teleological approach to defining the search space and searching through the space of all legal ways to resolve goal interactions. Thus it combined the protection interval maintenance approach in HACKER [16] with proper planning for the first time. Nonlin was the first planner to have an explicit criterion/algorithm to establish the value of a statement at a point in a partially ordered plan.

Nonlin used an algorithm we call Question Answering in a partially ordered network of nodes to establish the truth value of a statement at a particular point in the partial ordered plan. This took the form of a question:

Does P=V at N (using tactics)?

This asked "does the proposition P have a required value V at the indicated point N in a partially ordered network of time points". The answer is in the form of one or more possible "contributor" time points. A set of allowable "tactics" to use to compute the answer could be given in terms of legitimate changes that could be proposed to the plan (say in terms of variable bindings or temporal orderings) to establish the required value for the proposition. The QA algorithm came back with "yes" the proposition has the required value, "no" it does not and cannot in the current plan, or "maybe" if one of the indicated conjuncts of plan constraints (in terms of variable bindings and time point orderings) is applied.

This QA algorithm became a basis for work by Chapman on the formalisation of the concept in his Modal Truth Criterion (MTC) [6]. This led to boom in formal analysis of planners from Chapman onwards. It has been a valuable *packaging* attempt since it has led to practical and theoretical advances. If the algorithm had been buried in a planner implementation and not drawn out as a separate module, this would have been more difficult. As will be seen in a later section of this paper, the interface adopted for the Condition Question Answerer should be of general utility in the packaging of other modules in planning systems.

3. A Framework for Discussing "Standard" Components

In order to benefit from advances in various technologies and to allow improved implementations of components to be used, we need to be able to recognise separable functions and capabilities within our planners and schedulers. By separating the processing capabilities at the *architecture* level of a planner or scheduler from the *plan or schedule representation* we can begin to address modularity issues of this kind.

3.1. Plan Representation

There have been a number of research and development efforts directed at defining planning and scheduling system *products* in a way which is independent of the planner or scheduler that produces or uses them. This allows results of planning to be passed between various different systems or components. Ontologies of plans and schedules have been created to underpin such representations. An example is the KRSL plan representation language defined for the ARPA/Rome Laboratory Planning Initiative programme in the USA [2].

Query languages have been defined such that one part of a planning system can query other parts or can use information in repositories. An example is the use of KQML [5] with embedded KRSL plan representations on the ARPA/Rome Laboratory Planning Initiative. The definition of a general purpose PQL (Plan Query Language), a SQL for the planning world, has been attempted in a project intended to allow interfacing between natural language front ends and various back end planning systems [7],[8].

3.2. A Planner Architecture Abstraction

It is useful to present a simple abstraction of how a planner or scheduler operates. Figure 1 shows such an abstraction that will be useful in this paper.

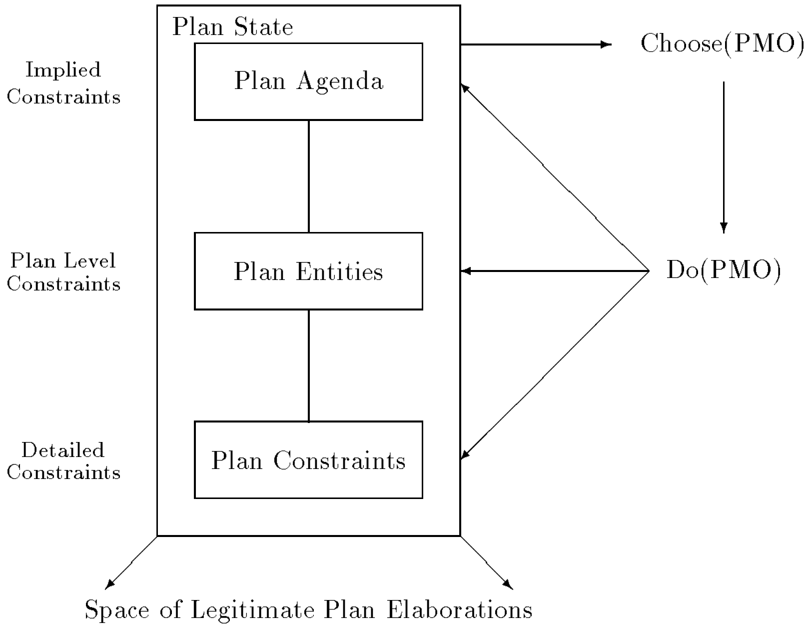

Figure 1: A Framework of Components in a Planning/Scheduling System

Many planners and schedulers work by refining a "current" plan (shown in figure 1 as the *Plan State*). They maintain one or more *partial plans* in this Plan State in which the previous decisions taken during the planning process restrict the space of plan elaborations which can be reached from that point.[1] The planner or scheduler needs to know what outstanding processing requirements exist in the plan (shown in figure 1 as the *Agenda*). These represent the implied constraints on valid plan solutions. One (normally) of these outstanding processing requirements is chosen to be worked upon next. This calls up processing capabilities within the planner which can make decisions and modify the Plan State - these are sometimes called *Plan Modification Operators*. The modifications can be in terms of definite plan structure in the Plan State or by noting further processing requirements (as a result of Plan State critiquing, etc).

[1]Plan constraint relaxation is also possible to increase the space of plan elaborations in some systems.

We have found it to be useful to separate the plan entities representing the decisions already made during planning into a high level representing the main plan entities shared across all planning system components and known to various parts of the systems, and more detailed specialised plan entities which form a specialised area of the representation of the plan. These lower level more compartmentalised parts can represent specialised constraints within the plan such as time, resource, spatial and other constraints. This separation can assist in the identification of modularity within planning and scheduling systems.

O-Plan2 [20], for example, has an *Associated Data Structure* (ADS) level of representation [12] which holds the main plan entities (such as activities). The lower level constraints then separately handle constraints on ordering and time points in the plan, resource constraints, etc. The lower level constraints are tied to the higher ADS level entities via associations. The TOSCA manufacturing scheduling system [4] which was based on the O-Plan2 architecture makes use of quite a different ADS level based on resource reservations, but shares the same time point constraint management code at the lower level.

4. "Standardising" Constraint Management in Planners

Moves to provide powerful constraint management languages and tools could lead to a rapid expansion of the benefits to be gained by identifying more standard components that can be combined and re-used in planning and scheduling systems. This can allow time network management, management of the persistence of facts across time, resource management, spatial constraint management and other such constraints to be managed by separate components provided by someone other than the original developer or integrator.

As one example, consider the provision of the management of temporal relationships in a planner. All modern planners embed some degree of time management for temporal relationships between time points or across time intervals and may provide support for metric (definite) time "stamps" on time points. Many planners also relate their time management to the management of the persistence of facts or propositions across time. This allows planners to reason about whether some required condition is true at a given time. The Time Map Management concepts, clearly described in [10] and used in the FORBIN planner [11], are a good example of the approach. The management of effect and condition (Goal Structure) tables in Nonlin [17] uses a similar approach.

This type of packaging has led to separate study of the support for time management and fact persistence management in planners at various research centres. O-Plan2 has a Time Point Network Manager [12]. A commercial Time Map Manager (TMM) is available from Honeywell based on the concepts described in [10]. More powerful temporal relationships are managed by the General Electric TACHYON temporal system [15]. In some cases, it has already proved possible to replace some simpler level of time constraint management in a planner with a better packaged and more powerful capability. One example of this has been the combining of the SRI SIPE-2 planner with the GE TACHYON temporal system. Other studies have indicated that the O-Plan2 TPNM can be replaced quite straightforwardly with the Honeywell TMM.

Studies at Edinburgh [13] relating to Resource Management have shown how progressively more capable resource management systems can be incorporated into O-Plan2 to replace the simple consumable resource handler in the system at present. These studies have developed a *Resource Criterion* interface to a Resource Utilisation Manager for the O-Plan2 planner which has many similarities to the interface used for the Truth Criterion/QA algorithm. This mechanism could allow resource handling by mechanisms as powerful as those based on the Habographs [4] constraint management mechanism incorporated in the Edinburgh TOSCA manufacturing scheduler.

Spatial constraint management which is not currently provided inside O-Plan2 has also been explored. We believe that clear modular interfaces can allow even such a "foreign" type of constraint management not understood by the core system at all to be be added reasonably straightforwardly to O-Plan2.

We will return in a later section to a proposal for a "standard" constraint management interface now being considered for O-Plan2. First we will introduce the O-Plan (Open Planning Architecture) work at Edinburgh and examine the ways in which modularity and interfaces are being explored in this research.

5. O-Plan – the Open Planning Architecture

The O-Plan2 Project at the Artificial Intelligence Applications Institute of the University of Edinburgh is exploring a practical computer based environment to provide for specification, generation, interaction with, and execution of activity plans. O-Plan2 is intended to be a domain-independent general planning and control framework with the ability to embed detailed knowledge of the domain. See [1] for background reading on planning systems and a chart showing how O-Plan2 relates to other planning systems. See [9] for details of O-Plan (now referred to as O-Plan1), the planning system that was a forerunner to the O-Plan2 agent architecture.

The O-Plan2 system combines a number of techniques:

- A hierarchical planning system which can produce plans as partial orders on actions.
- A control architecture in which each control cycle can post further processing steps on an agenda which are then picked out and processed by appropriate handlers (Knowledge Sources).
- 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.
- Constraint posting and least commitment on object variables.
- Temporal and resource constraint handling using incremental algorithms which are sensitively applied only when constraints can alter.
- O-Plan2 is derived from the earlier Nonlin planner [17] from which it takes and extends the ideas of Goal Structure, Question Answering (Truth Criterion) and typed conditions.
- We have extended Nonlin's style of task description language Task Formalism (TF).

O-Plan2 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 and satellites such as VOYAGER, ERS-1, etc.

6. A Sample O-Plan2 Scenario

- 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 *execution system* seeks to carry out the detailed activities specified by the planner while working with a more detailed model of the execution environment.

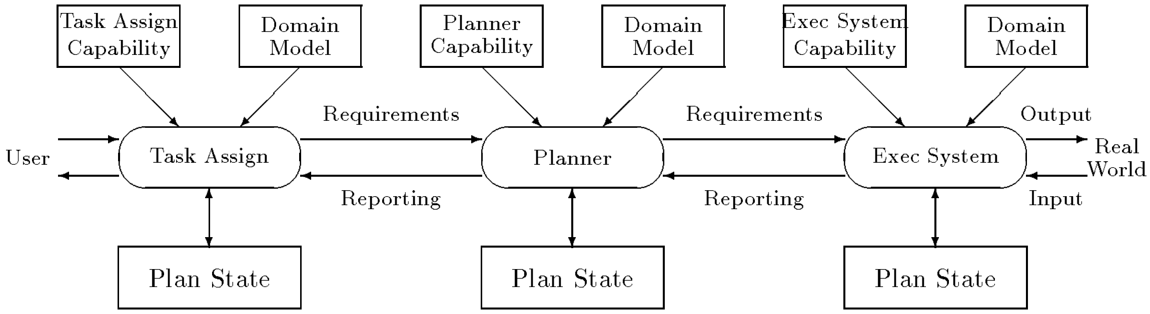

Figure 2: Communication between Strategic, Tactical and Operational Levels

The current O-Plan2 system is able to operate both as a planner and a simple execution agent. The task assignment function is provided by a separate process which has a simple menu interface. 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. However, we believe that the ideas are relevant to the more general case of a *cooperative, hierarchical and distributed command, planning and control* environment.

A common representation is sought to include knowledge about the capabilities of the task assigner, the planner and the execution agent, and the information used to represent the requirements of the plan and the plan itself either with or without flaws (see Figure 2).

The planner components described in outline form in Figure 3 can be mapped to the system and process architecture detailed in Figure 4. Communication between the various processes and support modules in the system is shown in the latter figure. More detail of the internal structure of O-Plan2 can be found in [20].

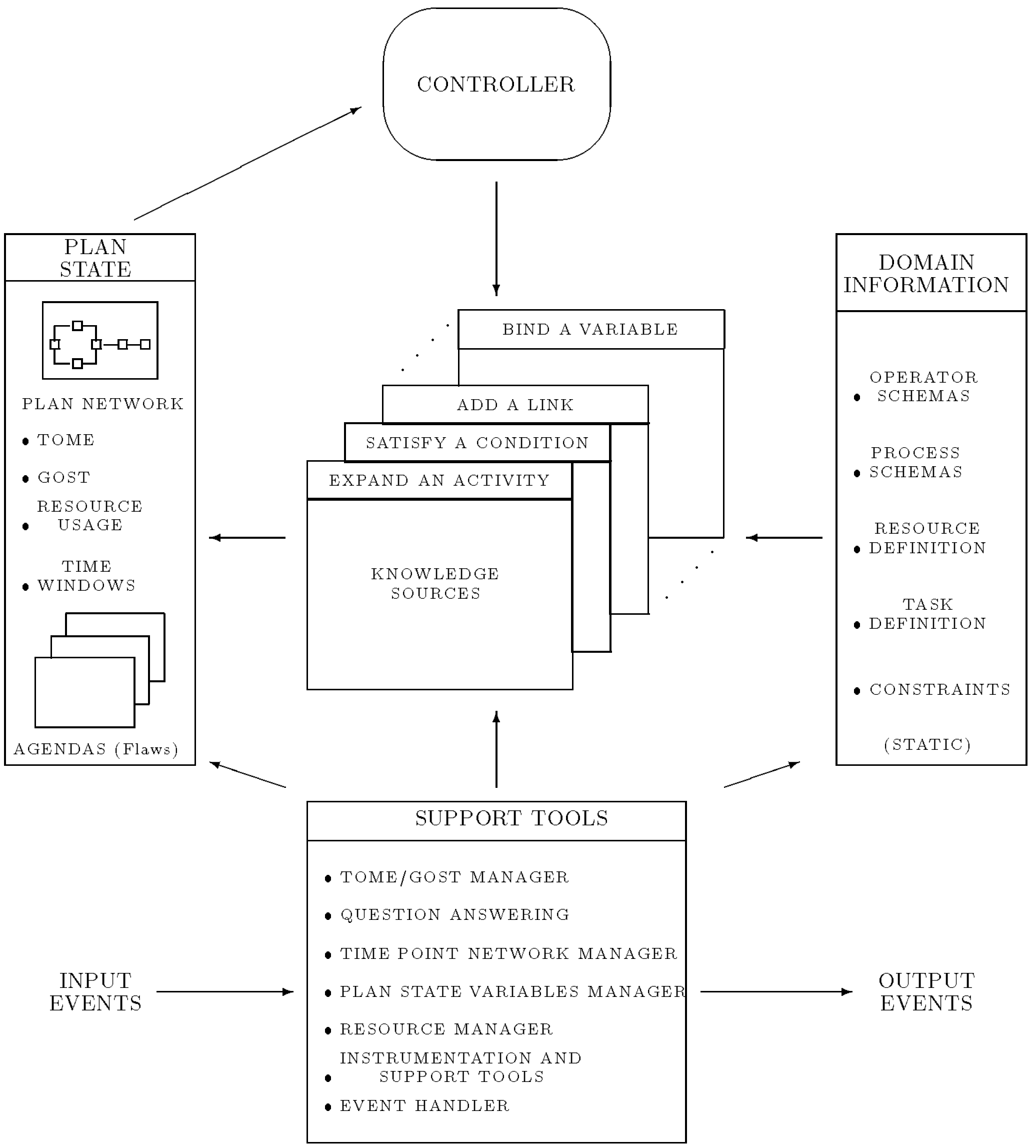

Figure 3: O-Plan2 Architecture

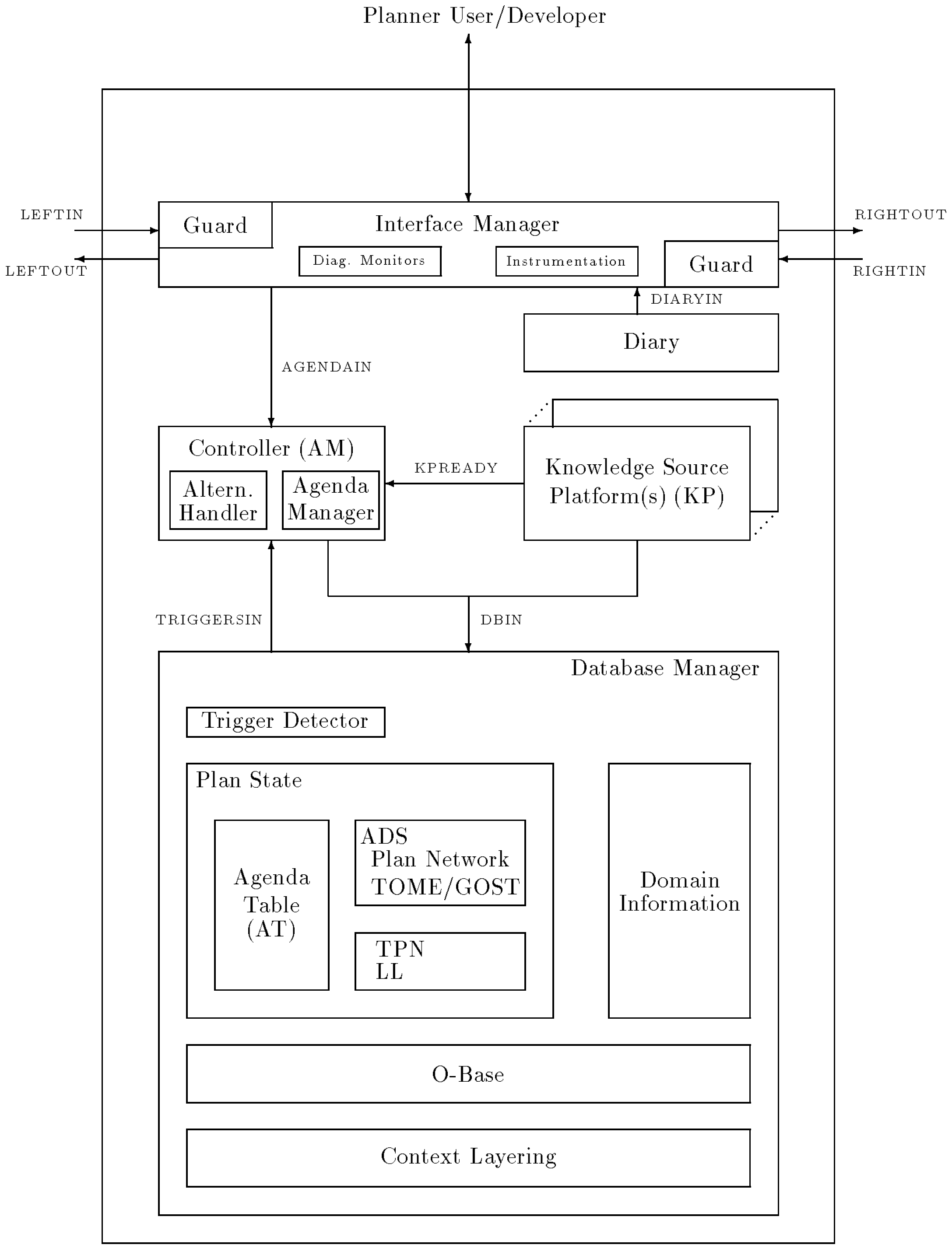

Figure 4: Internal Structure of the Current O-Plan2 Planner

7. Developer Interface

O-Plan2 is implemented in Common Lisp on Unix Workstations with an X-Windows interface. It is designed to be able to exploit distributed and multi-processor delivery systems in future. It therefore has a clear separation of agent roles (as shown in figure 2) and the various components (as shown in Figure 3). Each of these may be run on a separate processor and multiple *platforms* may be provided to allow for parallelism in knowledge source processing. Lower level constraint management and support routines are intended to allow for the exploitation of massively parallel computational and data base architectures and special purpose hardware.

A sample screen image as seen by the O-Plan2 developer or an interested technical user is shown in Figure 5.

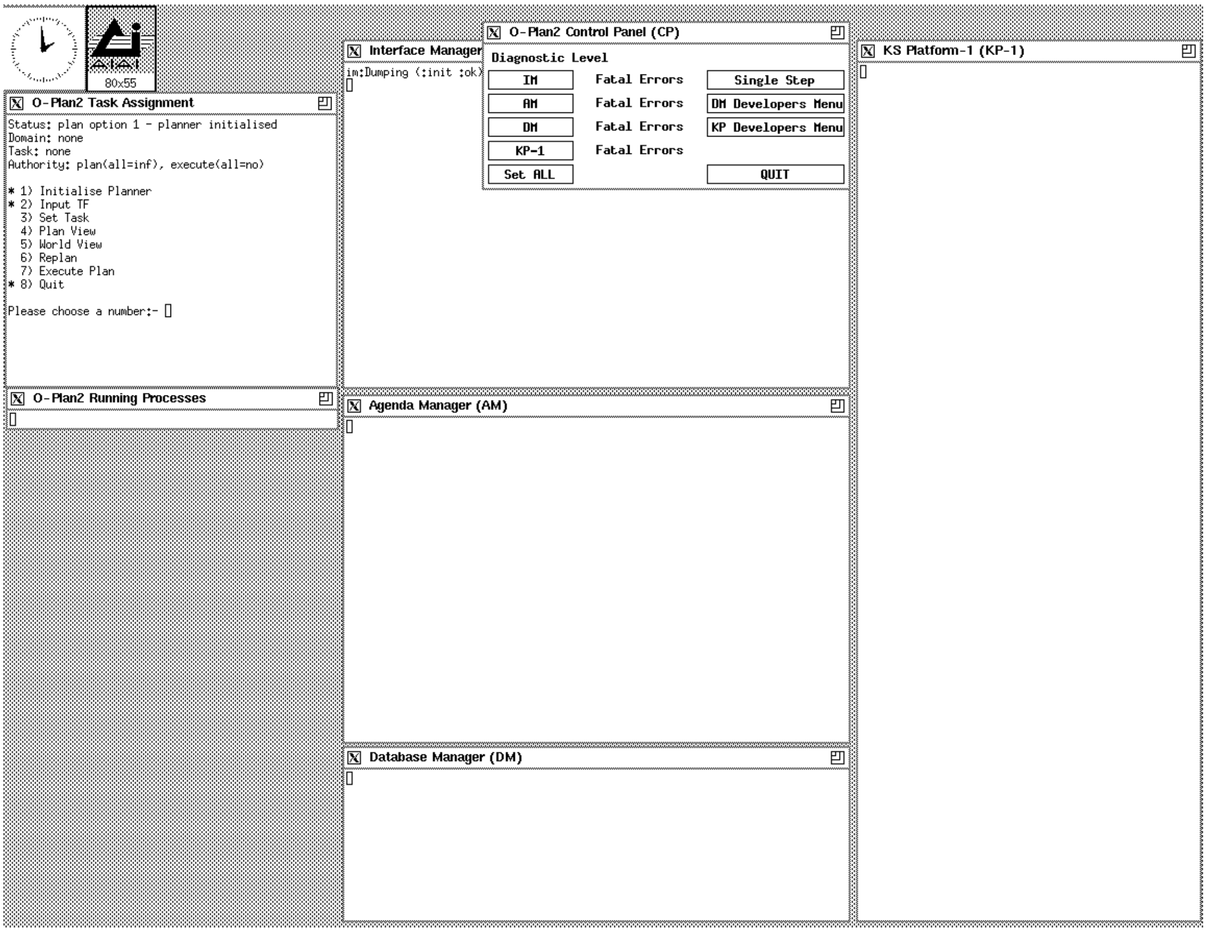

Figure 5: Example Developer Interface for the O-Plan2 Planning Agent

8. 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 they are being aided with. An interface to AutoCAD [3] has been built to show the type of User Interface we envisage (see Figure 6). The window in the top left corner shows the Task Assignment menu and supports the management of authority to plan and execute plans for a given task. The lower window shows a *Plan View* (such as showing the plan as a graph), and the upper right window shows a *World View* for simulations of the state of the world at points in the plan. The particular plan viewer and world viewer provided are declared to the system and the interfaces between these and the planner uses a defined interface to which various implementations can conform. Most of the developer aspects of the planner interface are not shown to the normal user. In figure 6, the developer windows are shown in iconic form along the top edge of the screen.

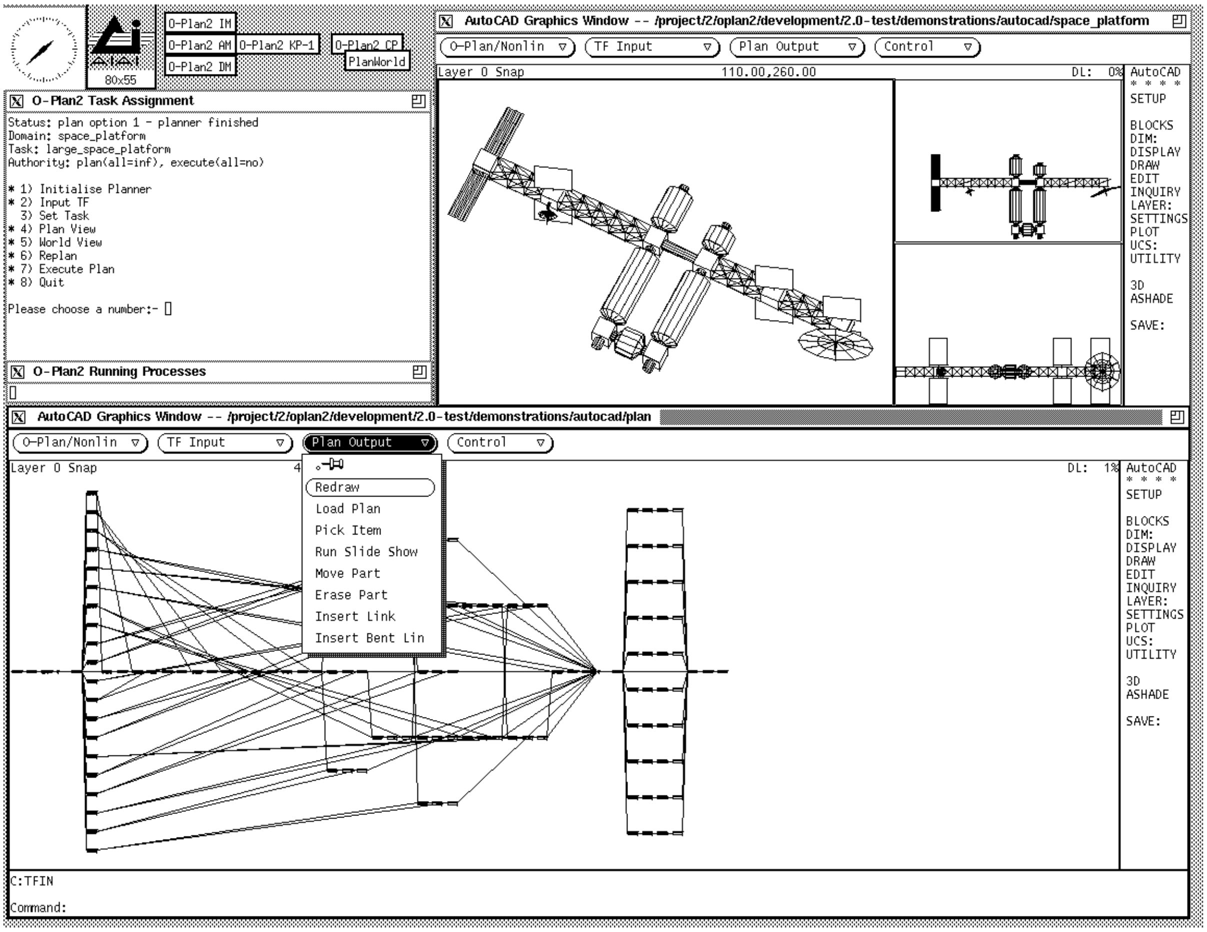

Figure 6: Example Output of the AutoCAD-based User Interface

9. O-Plan2 Modularity, Interfaces and Protocols

9.1. O-Plan2 Components

The O-Plan2 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* – PMOs).

4. Constraint Managers and Support Modules – functions which support the processing capabilities of the planner and its components.

5. Controller – the decision maker on the *order* in which processing is done.

9.2. Constraint Managers and Support Modules

Constraint Managers and 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 questions asked of them to the decision making level itself. They normally act to manage information and constraints in the Plan State. Examples of Constraint Managers and Support Modules in O-Plan2 include:

- Effect/Condition (TOME/GOST) Manager and Question Answering (QA) [19] – TGM.

- Resource Utilisation Manager – RUM.

- Time Point Network Manager [12] – TPNM.

- Object Instantiation (Plan State Variables) Manager – PSVM.

- Alternatives Manager.

- Interface and Event Manager.

- Instrumentation.

- Monitors for Output Messages, etc.

9.3. Protocols

In addition, a number of external interface specification and protocols for inter-module use have been established. Only first versions of these interfaces have been established at present in O-Plan2, 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 elsewhere into the planning framework too.

The protocols for regulating the processing conducted by a component of O-Plan2 are:

1. *Knowledge Source Protocol* – describing 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 Data Base Manager.

2. *KS-USER Protocol* – describing the ways in which the user (in the role of *Planner User*[2]) can assist the planning system via a specially provided Knowledge Source.

3. *Inter-agent Communications Protocol* – controls the way in which the Knowledge Sources operate and may use the Interface Manager's support routines which control the agent's input and output event channels.

9.4. Internal Support Facilities

The internal support provided within the planner to assist a System Developer and 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-Plan2 currently provides a limited number of monitorable triggers of this kind, but we anticipate this being expanded significantly in future.

[2]The O-Plan2 design identifies a number of distinct *roles* or ways in which a user may interact with the system.

3. *Controller Priority Language* – currently, the O-Plan2 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-Plan2 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 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.

9.5. 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 Viewer User Interface* – which allows for domain specific plan drawing and interaction to be provided.

3. *World Viewer User Interface* – which allows for domain specific world state input and simulation facilities to be provided.

4. *External System Interface* – provided by TF **compute conditions** [18] for ways in which external data bases, modelling systems, CAD packages, 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 if appropriate.

10. Constraint Managers in the O-Plan2 Architecture

O-Plan2 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 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.

It is intended that some of these Constraint Managers could be replaced by more efficient or more capable systems in future. This section considers the interfaces between the O-Plan2 architecture components and Constraint Managers to help others consider packaging and integration issues.

Our experience with earlier AI planners such as Nonlin and O-Plan1 was that a large proportion of the processing 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 in O-Plan2 as Constraint Managers (such as a Time Point Network Manager, an Effect/Condition Manager and a Resource Utilisation Manager), and Support Routines (such as a Graph Operations Processor) to allow for future improvements and replacement by more efficient versions.

Constraint Managers are intended to provide efficient support to a higher level of the planner 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. Examples of Constraint Managers in O-Plan2 include:

- Time Point Network Manager (TPNM).
- Effect/Condition (TOME/GOST) Manager (TGM) and the related Question Answerer (QA).
- Resource Utilisation Manager (RUM).
- Object Instantiation (Plan State Variables) Manager (PSVM).

A guideline for the provision of a good Constraint Manager in O-Plan2 is the ability to specify the calling requirements for the module in a precise way (i.e. the *sensitivity rules* under which the Constraint Manager should be called by a knowledge source or from another component of the architecture).

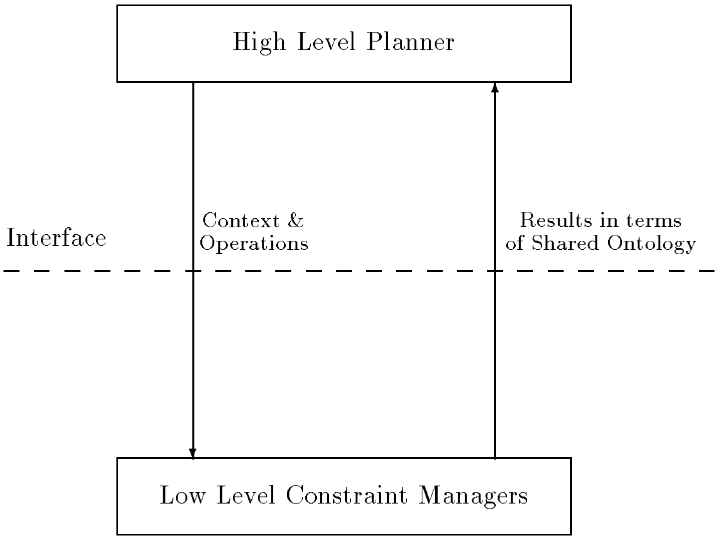

Figure 7: The Interface to Constraint Managers

The following sections explore the definition of an interface between the higher level decision making part of a planning or scheduling ssystem and a lower level constraint manager. Figure 7 shows an overview of the interface.

10.1. Constraint Manager Procedural Interface

A Constraint Manager is a part of the Database Manager (DM) component in O-Plan2 which looks after the Plan State and all of its alternatives (if any). A Constraint Manager may look after a specialised aspect of the Plan State on behalf of the DM.

The O-Plan2 design is being rationalised so that a Constraint Manager has the following generic procedural interface:

1. initialise Constraint Manager and name base context with given <tag>[3].
2. terminate Constraint Manager
3. push context and name new context with given <tag>
4. pop context to parent of current context
5. restore a previously created context which has the <tag> specified
6. open update transaction, and within this allow:
   - allow changes to managed entities[4].
   - queries can be made inside an open transaction. Any query reflects the changes made within the transaction to date.
   - nested open update transactions are not allowed (in O-Plan2 at present).
7. commit changes made within the update transaction
8. abort changes made within the update transaction

Some of the above routines may be inoperative or null for specific managers. In particular, context management as specified above is not needed for any Constraint Manager which chooses to make use of the O-Plan2/O-Base context managed structures – since the Associated Data Structure (ADS [12]) layer in O-Plan2 guarantees that Constraint Managers will only ever be called when the contexts being referred to are preset within the O-Plan2 planner.

10.2. Shared Plan Ontology between O-Plan2 and Constraint Managers

There are specialised update and query routines supported by each constraint Manager. These share a common plan entity model within the planner and its Associated Data Structure (ADS) layer. The design intention has been to keep this minimal, including only those elements that allow relevant communication between higher level planning decisions and lower level constraint management. This model includes *only*:

- a directed acyclic graph of time points.
- ability to map a plan activity node end to a unique time point and a time point to all associated node ends.
- time points as plan entities.

[3]Contexts specify alternative views of a Plan State. A tree of such contexts is manipulated by O-Plan2.

[4]An extra standard update routine is needed in our implementation to handle O-Plan2 TF **other_constraints** statements (constraints not directly understood by the planner) relating to this particular constraint manager.

- an ordering relation on two time points – before(tp1,tp2).
- context <tag>s to represent alternative Plan States.
- An understanding of the meaning of a Plan State Variable[5].

These entities allow for information to be communicated about constraints and options for correcting constraint violations in terms of the shared model. All other more specific entities may be unique to a specific Constraint Manager or shared only between pairs of caller and manager.

10.3. An Emerging "Standard" General Interface for Constraint Managers

The aim in O-Plan2 is to provide a standardised interface between each Constraint Manager and the rest of the planner. For this we are seeking to employ a very similar interface to that used by the Nonlin or O-Plan style Condition Question Answerer (QA) or Truth Criterion.

A Constraint Manager cannot take any decisions and cannot change parts of the Plan State not under its immediate management. It must return all legitimate answers for the query it is given or must undertake reliably the task it is given. One focus of the O-Plan2 research has been to build a *planning ontology* which describes those concepts which are shared between constraint managers and those parts of the Plan State which are private to the relevant manager.

A Constraint Manager's primary function is to manage the current set of constraints relevant to that manager (time, resource, spatial, objects, etc) which are part of the Plan State. It must signal to the caller when there is an inconsistent set of such constraints.

The interface allows for a constraint entry to be tested against existing managed constraints to see what the impact of making the entry would be, and then a commit or abort can be done to add it or not (either the commit or the abort could be active – the caller not being able to tell).

All Constraint Manager update routines return one of three results:

- **yes** – constraint is now under management (to be confirmed later by a caller using a commit update transaction).
- **no** – constraint cannot be added within the capabilities of the Constraint Manager and its communications capability to the caller (in terms of the shared ontology of entities).
- **maybe** – constraint can be added if plan entities are altered as specified in terms of the shared entity model. This normally means returning a standard O-Plan2 "or-tree"[6] of *all* (for search space completeness) the legal ways in which the Plan State can be altered (sets of Plan State Variable restrictions and ordering constraints between time points) to maintain consistency.

[5]The exact nature of what needs to be understood in the shared ontology needs to be considered further.

[6]a data structure representing the alternative ways in which the Plan State may be altered in terms of the shared plan ontology.

The constraint is *not* added after this maybe response. However, an "actually add constraint" routine may be provided to more cheaply add the constraint immediately following a query which returned "maybe". This would follow action by the caller to ensure at least one of the relevant binding constraints and/or time point orderings options were either dealt with or noted as necessary in the Plan State - thus the caller takes responsibility for resolving inconsistencies (*not* the Constraint Manager).

It is hoped to be able to take the result or-trees generated by the various Constraint Managers in O-Plan2 (TGM, RUM, PSVM and the TPNM) and merge them into a consistent or-tree which would represent an efficiently ordered set of possibilities – thus reducing the size of the search space.

We believe that this style of interface between the higher level decision making level of the planner and the various Constraint Managers could improve modularity in planning systems.

11. Summary

This paper was intended to further discussions on the identification of suitable "standard" re-usable components in planning and scheduling systems.

This paper has presented an overview of the O-Plan2 system under development at the Artificial Intelligence Applications Institute of the University of Edinburgh. Aspects of the system concerned with separation of functionality within the system, internal and external interfaces have been addressed. The O-Plan2 system is starting to address the issue of what support is required to build an evolving and flexible architecture to support command, planning and control tasks.

One particular area highlighted has been the interface between planning systems and Constraint Managers able to look after certain specialised aspects of parts of a plan on behalf of the overall planning system. An interface to such Constraint Managers has been developed to show how improved packaging can be beneficial to re-use of components. The value of the type of interface developed for the Condition Question Answering procedure in planners (the Truth Criterion) to act as a general interface to a number of different Constraint Managers has been explored.

Acknowledgements

O-Plan2 is an on-going project at Edinburgh. I am grateful to Brian Drabble and Jeff Dalton for their input to the work described here. Current O-Plan2 work is supported by the US Advanced Research Projects Agency (ARPA) and the US Air Force Rome Laboratory acting through the Air Force Office of Scientific Research (AFSC) under contract F49620-92-C-0042. The United States Government is authorised to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

Parts of section 9 of this paper were previously presented at the European Space Agency's Round Table on Planning and Scheduling Tools, 6-7th April 1992, ESTEC, Noordwijk, The Netherlands.

References

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

[2] ARPA/Rome Laboratory Planning and Scheduling Initiative, Knowledge Representation Specification Language (KRSL) Reference Manual (eds, Allen, J. and Lehrer, N.), Version 2.0, September 1991. ISX internal technical report, ISX, 4353 Park Terrace Drive, Westlake Village CA 91361, USA.

[3] AutoDesk, AutoCAD Reference Manual, 1989.

[4] Beck, H., TOSCA: A Novel Approach to the Management of Job-shop Scheduling Constraints, Realising CIM's Industrial Potential: Proceedings of the Ninth CIM-Europe Annual Conference, pages 138-149, (eds. Kooij, C., MacConaill, P.A., and Bastos, J.), 1993. Also available as AIAI Technical Report AIAI-TR-121.

[5] Chalupsky, H., Finin, T., Fritzson, R., McKay, D. Shapiro, S. and Wiederhold, G., An Overview of KQML: A Knowledge Query and Manipulation Language, Proceedings of Concurrent Engineering and Computer Aided Logistics Conference, Washington, USA, June 1992.

[6] Chapman, D. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333-377, 1991.

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

[8] Crouch, R.S. and Pulman, S.G., Time and Modality in a Natural Language Interface to a Planning System, *Artificial Intelligence* 63, pages 265-304, 1993.

[9] Currie, K.W. & Tate, A., O-Plan: the Open Planning Architecture, *Artificial Intelligence* 51(1), Autumn 1991, North-Holland.

[10] Dean, T. and McDermott, D., Temporal Database Management, *Artificial Intelligence* 32(1):1-56, 1987.

[11] Dean, T., Firby, J. and McDermott, D., Hierarchical Planning Involving Deadlines, Travel Time and Resources, *Computational Intelligence*, 6(1), 1990.

[12] Drabble, B. and Kirby, R.B., Associating A.I. Planner Entities with an Underlying Time Point Network, European Workshop on Planning (EWSP) 1991, Springer-Verlag Lecture Notes in Artificial Intelligence. Also available as AIAI Technical Report AIAI-TR-94.

[13] Drabble, B. and Tate, A., Resource Representation and Reasoning in O-Plan2, AIAI Technical Report ARPA-RL/O-Plan/TR/6, April 1993.

[14] Sacerdoti, E., *A Structure for Plans and Behaviours*, Artificial Intelligence Series, North Holland, 1977.

[15] Stillman, J., Arthur, R. and Deitsch, A., Tachyon: A Constraint-based Temporal Reasoning Model and its Implementation, *SIGART Bulletin*, 4:3, July 1993.

[16] Sussman, G.J., A Computational Model of Skill Acquisition, MIT AI Laboratory Technical Report TR-297, 1973.

[17] Tate, A., Generating Project Networks, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-77), Cambridge, Mass., USA, 1977.

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

[19] Tate, A., 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.) Morgan Kaufmann, 1986.

[20] Tate, A., Drabble, B. and R.B.Kirby, R.B., O-Plan2: an Open Architecture for Command, Planning and Control, in *Knowledge Based Scheduling* (eds. M.Fox and M.Zweben), Morgan Kaufmann.

[21] Wilkins, D., *Practical Planning*, Morgan Kaufmann, 1988.