Coordinating the activities of a Planner and an Execution Agent

Austin Tate

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

Abstract

We have defined a research programme that will explore the link between planning and execution systems. A simple scenario has been defined in which we have a very capable off-line planning system interacting with the user and a smaller, less capable, on-line real-time system executing plans and reacting to faults. However, the on-line execution system may have a more flexible representation of the plans it is executing. This imbalance in the capabilities of the two ‘‘agents’’ involved should clarify some of the research objectives and give us an experimental framework for our work. Our task is to investigate the knowledge representations and communication protocols needed to link a user stating some requirements for a task to be carried out through a planning system to the (remote) execution agent that can carry out the user’s wishes.

We are starting from the notion that a single representation can encapsulate the expression of the user’s requirements, the capabilities for action, the communication to the execution agent, the successful or faulty response from the execution agent and the means of keeping the user informed. This is based on our work on *Goal Structure*, which captures the intent of plan steps; *Task Formalism*, a declarative input language for the planners we have built at Edinburgh; and on the definition of a *Plan State*.

Methods of creating *Plan Patches* to update the plans separately held by each of the parties involved to keep them in step as they each react to changing circumstances in real-time will be investigated. This will involve the specification of plan patch *Attachment Points* that can be understood by the recipient. We will also investigate transaction based methods for coordinating the activities of the planner with those of the execution agent and user.

The trial application area for the research is in the command and control of an advanced Earth Observation Space Platform.

1. Introduction

We have just embarked on a new phase in our research concerned with the application of Artificial Intelligence (AI) planning techniques to the area of spacecraft command and control. In particular, the research addresses the issue of closing the loop between plan generation and execution monitoring in the face of simple plan failures.

1.1. Planning and Executing: Closing the Loop

In this project, we intend to make use of our experiences in dealing with applications of AI planning techniques to practical projects (with O-Plan, Currie and Tate, 1985) and 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 (Fikes, Hart and Nilsson (1972), Wilkins (1985), Malcolm and Smithers (1988), Drabble (1988)), 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 (Georgeff and Lansky (1986), Rosenschein and Kaelbling (1987), Nilsson (1988)) and those that can be generated by today's AI planners.

However, 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.

1.2. Planning with semi-autonomous agents

Most planners to date have constructed their plans with full knowledge of the capabilities of the devices under their control. Thus, executing such plans involves the direct application of the operators within the plan by an execution agent which has no planning capability. Invariably, 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.

The scenario we intend to investigate is as follows. A central planner plans to achieve a task described at a high-level of abstraction. The central planner has knowledge of the general capabilities of a semi-autonomous execution agent but does not need to know about the actual operators that execute the actions required to carry out the desired task. The execution agent executes the plan by choosing the appropriate operators to achieve the various sub-tasks within the plan, using its knowledge about the particular devices 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 operators within the plan and responding to failures fed back from the real world. Such failures may be due to the inappropriateness of a particular operator, or because the desired effect of an operator was not achieved due to an unforeseen event. The reason for the failure dictates whether the same operator should be re-applied, replaced with other operators or whether re-planning should take place.

1.3. The Role of Dependencies in Plans

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

Early work on Decision Graphs by Hayes (1975) 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 invalidate the success of the current plan. Some work to link these ideas with a non-linear AI planner was undertaken during the mid 1970s by Daniel (1983).

The notion of the teleology of a plan, which we call the Goal Structure (Tate, 1977), refers to the dependencies between the preconditions and postconditions of operators 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 (Fikes, Hart and Nilsson (1972), Tate (1984)), there has been no comprehensive discussion of their use in all aspects of plan generation, execution monitoring and plan repair.

2. Planning and Execution Architecture

Recently, we have been promoting 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.

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 the diagram above. 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 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. Plan States

The O-Plan (Currie and Tate, 1985) Plan State holds a complete description of a plan at some level of abstraction. The Plan State 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.

The O-Plan *Task Formalism (TF)* is a declarative language for expressing action schemata, for describing task requests and for representing the final plan. Our design intention for O-Plan is that a Plan State can be created from a TF description and vice versa. This has not quite been achieved in the existing O-Plan prototype (Currie and Tate, 1989), but this remains our goal. The aim is that the AI 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.

In practice, 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.

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 (Currie and Tate, 1985). An agenda of outstanding flaws is maintained in a Plan State and appropriate Knowledge Sources are scheduled on the basis of this.

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. We will explore the new properties that we must seek from our basic notions in the following sections.

4. 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 already 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 propose to define a *Plan Patch* as a modified version of the type of Plan State used in O-Plan. It would have some similarity to an operator or action schema given to an AI planning system in that it would be 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 would provide 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 could be handled by using this abstracted view (e.g., Georgeff and Lansky (1986), Nilsson (1988)). 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 (Malcolm and Smithers, 1988).

Outstanding flaws in the Plan Patch would be 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 user and the planner in an O-Plan type AI planner. Requiring rather more effort will be 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 user, but we believe that this is a viable objective.

5. 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 intend to endow all the components with a real-time clock that can be assumed to be fully synchronised. We will 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 will be the ‘‘back-stop’’ as a means of attaching a Plan Patch into the internal Plan State of the receiver. Metric time will also be 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. Initial work will centre on descriptions relative to the expected Goal Structure (Tate, 1977) of the receiver.

6. Incremental Plan States

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

1. a plan patch,
2. plan patch flaws as an agenda of pending tasks,
3. plan patch attachment points.

Such Incremental Plan States will be used for two way communication between the user and the planner and between the planner and the execution agent. Our current Plan State structures and flaw repertoire will be 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 Tate, 1984). Later in the research programme, 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.

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

7. 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:

1. plan patch description is clear,
2. plan patch flaws can be handled by receiver’s Knowledge Sources,
3.

attachment points understood. It is important that the sender and receiver (whether they are the user 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 their own Plan State. We propose to use *transaction processing* methods to ensure that such coordination is achieved.

We expect to create some specific flaw types and Knowledge Sources in the various components (user interface, 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 will 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.

8. Application to Spacecraft Command and Control

Spacecraft command and control provides a realistic target domain for looking at planning and execution, in particular, for planning with semi-autonomous agents. The desire to improve the autonomous capabilities of a spacecraft is apparent, especially for spacecraft in communication with an earth-based segment, such as for a deep space probe in communication with its mission control or for a satellite with its ground station, or, indeed, for a space station that is controlling various on-board devices such as robot arms and orbital manoeuvring vehicles.

The NASA Jet Propulsion Laboratory Deviser planner which was applied to the task of generating command sequences for the Voyager spacecraft (Vere, 1983) was based on our earlier work on the Nonlin planner (Tate, 1977). Recently, AIAI has had a team of people who have worked on the application of Knowledge-Based Planning and other Knowledge-Based Systems techniques to the area of spacecraft command and control. This has been funded by the European Space Agency for ERS-1 scheduling (Fuchs et al., 1988) and by the UK Science and Engineering Research Council for work on a technology proving satellite T-SAT (Drummond, Currie and Tate (1987, 1988), Fraser et al (1988)).

The investigation of planning and execution using the approach described above will be carried out in the context of an application to spacecraft command and control using data from a system such as ERS-1 or the Polar Platform segment of the International Space Station. We are alert to the possibility of viewing our techniques as being relevant to the process of *Task Amplification* whereby a user's commands can be interpreted via a planner and an intelligent execution monitor in a teleoperations environment.

9. Next Steps

The description in this paper of our approach to the integration of planning and execution is based on ideas and techniques that have been developed over a significant period of time. A number of important building blocks are now seen to be in place for a concerted effort to construct such an integrated command and control system able to operate in a realistic application domain.

We believe that the simplifications we have made and the differentiation of the nature of the experimental planning and execution environments we will explore will assist in clarifying our research approach. However, we believe that the architecture should be quite general if we are successful. We expect that one advantage of the line of attack we propose will be the ability to deal with large scale realistic execution environments of the type now being developed by other researchers.

10. Acknowledgements

The work of the Knowledge Based Planning Group at AIAI in Edinburgh is supported by the Science and Engineering Research Council (Alvey Directorate grant GR/D/5898-7 An Architecture for Knowledge Based Planning and Control and SERC grant GR/E/98058 Knowledge Based Command and Control of a Spacecraft). Members of the Knowledge Based Planning

Group at AIAI (especially Ken Currie and Roberto Desimone) helped formulate the approach described in this paper and assisted with its preparation. The support of SD-Scicon and the United States Air Force for the work of the group is gratefully acknowledged.

11. References

Fuchs, J., Guldberg, J., Olalainty, B. and Currie, K.W. (1988) "Expert Planning System for a Space Application" Workshop on AI Applications for Space Projects, ESTEC, European Space Agency.

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.

Currie, K.W. and Tate, A. (1989) "O-Plan: the Open Planning Architecture", to appear.

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

Drabble, B. (1988) "Intelligent Execution Monitoring and Error Analysis in Planning involving Processes", Ph.D. Thesis, University of Aston in Birmingham.

Drummond, M.E., Currie, K.W. and Tate, A. (1987) "Contingent Plan Structures for Spacecraft" Proceedings of the NASA Workshop on Space Station Telerobotics, JPL Publication 87-13 Vol III, pp. 17-25 (ed.) G. Rodriguez. Jet Propulsion Laboratory, California, USA, January 1987.

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.

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

Fraser, J., Conway, S.M., Currie, K.W., Drummond, M.E., Lockwood, R.M., Macintosh, A.L. and Tate, A. (1988) "Using on-board AI to increase spacecraft autonomy", International Conference on Human-Machine Interaction and Artificial Intelligence in Aeronautics and Space, Toulouse, France, September 1988.

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.

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

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.

Nilsson, N.J. (1988) "Action Networks", Proceedings of the Rochester Planning Workshop, October 1988.

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

Tate, A. (1977) "Generating Project Networks", IJCAI-77, Proceedings of the International Joint Conference on Artificial Intelligence, pp 888-893, Cambridge, Mass., USA.

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.

Vere, S.A. (1983) Planning in Time: Windows and Durations for Activities and Goals. IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-5, No. 3, May 1983, pp 246-267.

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