Plan Execution Using Human Agents

Carol A. Broverman
W. Bruce Croft

COINS Technical Report 89–83
March 1989

Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01003

Most planning systems have been applied to simple domains. In complex domains, the autonomy of human agents and the dynamic nature of realistic settings give rise to frequent exceptional occurrences (exceptions). Rather than using a traditional error recovery approach, we advocate the use of plan recognition techniques to identify the purposeful behavior underlying an exception and its contribution to an ongoing plan. This paper discusses a model of plan execution and exception handling, and describes SPANDEX, an implementation of this approach. The SPANDEX system produces explanations consisting of *rationales* and *amendments* to incorporate exceptions into the current plan, allowing planning and execution to continue.

This work is supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, the Air Force Office of Scientific Research, Bolling Air Force Base, District of Columbia 20332, under contract F30602-85-C-0008, and by a contract with Ing. C. Olivetti & C.

Contents

1 Motivation

Planners can potentially be used to automate or support a variety of complex tasks [8,12,14]. Most planning research, however, has been done in very simple domains (e.g. the blocks world). The dynamic and unpredictable nature of many real world domains suggests that sophisticated monitoring of plan execution is vital and systems should have the capability to respond to unexpected change. Recovery measures such as those of [1,6,14] have been proposed to effectively replan around an unanticipated domain state change, allowing resumption of the task while preserving as much of the plan as possible.

In our work, we are concerned with the special requirements of domains where a planner is used to support the cooperative work of one or more human agents [8]. In such environments, human input is required to guide the development of a plan for a task. In addition, execution of plan steps will be performed by human agents as well as by the planning system. The natural intelligence and familiarity of humans with the application domain means that their actions, even when inconsistent with system expectations, are generally purposeful. That is, human-generated plan exceptions should be incorporated

into the developing plan, rather than "undone" using replanning techniques.

A planner employs a set of axioms that defines the planning process and a predefined set of domain activities and objects to generate valid plans that accomplish a particular goal. While restricting a potentially explosive search space, the plans that are produced are stereotypical and may not be adequate predictors of subsequent execution behavior. Our approach is to make a conventionally produced plan "elastic" in response to exceptions and to thus allow the continuation of planning and execution [2,3,5]. The domain knowledge base is used in an attempt to transform the current plan into a valid alternative, or, put another way, to recognize an alternate plan. During this process, additional domain knowledge may be acquired. The overall goal of our approach is to allow the system to continue planning and execution while incorporating the exceptions that occur in real domains.

In this paper, we give a detailed formulation of plan execution and the exception problem. We then describe iterative and interactive algorithms which provide explanations for exceptions by establishing plausible rationales and proposing corrective measures. In the final section, we describe the implementation of these ideas in the SPANDEX exception handling system using an example from the software development domain.

2 Planning

In this section, we give a formalization of the planning problem and define terminology which is relevant to plan execution and exception handling. Our definition of the planning task of a hierarchical nonlinear planner is similar to that proposed in [9]:

Given:

- w: an initial world state
- A: a set of activities, some of which are primitive (A_p) and others which are complex (A_c);

- g: a desired final goal state;
- E: a set of available agents;

Determine: a partial ordering P of primitive activities A_p in A which, when executed in an initial state w by agents in E, will produce a new state containing the final goal state g.

Activities are considered complex if they can be elaborated by the planner into sub-activities; primitive activities are associated with executable actions. The partial ordering P which represents the final plan for a task is the result of a series of transformations of the initial goal specification g. The result of each of these manipulations is represented by a *plan network*, which represents the current version of an evolving plan. A plan network is a strict partial order and consists of the following elements[1]:

- N: a set of nodes, where each node represents a *goal* or *activity*;
- L: a set of temporal links which establish a partial ordering among the nodes;
- W: a set of world states, which are snapshots of the dynamic domain knowledge base. Two of these world states are attached to each node in N to describe the world states believed by the planner to hold before *(before-world)* and after *(after-world)* the execution of that node;
- I: a set of *protection intervals*, where each interval specification designates a partial world state and the temporal range during which it must be maintained.

Plan networks can also be described in terms of the stage of their execution. Since the domains we are concerned with generally interleave planning with execution, plan networks are often partially executed. Associated with each plan network is a set of *expected action*

[1]More detailed descriptions of each of these elements can be found in [8].

nodes which are the nodes which can be executed next. A node is in this set if and only if[2]:

- it is a primitive activity node;
- all of the conditions specified in the node's before-world are satisfied;
- all of the node's necessary predecessors are complete and awaiting successors.

The process of *planning* is viewed as iterative *transformations* on plan networks. A complete *plan* is a plan network which has been fully ordered, and every node is either a *phantom*[3] or it is a primitive activity node that has already been executed. Thus, a plan network represents a class of complete plans; there are multiple possible complete plans that may result depending on the choices of elaborations and operations that are subsequently applied. As a plan network is further elaborated and executed by the plan network maintenance system (PNMS[8]), a *plan history* is built up since a new plan network results each time a PNMS operation is performed. PNMS operations include node expansions, the imposition of temporal orderings and protection intervals, etc. The complete *plan history* is the set of all intermediate plan networks created by the planner. Thus, the plan history is a partial order of plan networks ordered within planning time, where the distinguished upper bound is the eventual complete plan. The relation is a partial order since backtracking may be allowed. The plan history maintains a record of all planning actions performed in the production of the final plan.

We define the concept of a *plan wedge*[4] in order to be able to refer to the portion of plan history that represents the abstractions and subsequent refinements which introduced a

[2]A more complete definition of "ready nodes" which defines "conditions" and "necessary predecessors" in detail can be found in [8].

[3]A phantom node [13] is a goal node which has been determined to be true at its position in the plan without further expansion and execution.

[4]Our definition is similar to the definition of a wedge used by Wilkins [14] and produces the semantic equivalent.

given node n into the plan. The concept of a wedge is important both for general replanning and in establishing a rationale for how an unanticipated event may be relevant to the plan history. A *plan wedge* for a node *n* is a set of nodes defined as follows:

> Given the following recursive functions:
>
> 1. **node-ancestors(n)** which returns set of nodes containing the parent node which produced the expansion containing **n** as well as all **node-ancestors(parent node)**, and
> 2. **node-descendants(n)** which returns all children nodes which form the expansion of n, as well as all **node-descendants(child)** for each of the children nodes,
>
> a *plan wedge* consists of a distinguished node in **node-ancestors(n)** which is chosen as the *apex* of the wedge, and the set of nodes in **node-descendants(apex)**.

3 Plan Execution and Exceptions

We can now describe how exceptions arise during the planning and execution process. We sketch the system loop in order to provide an overall context:

1. The planner completes an elaboration cycle of the current plan network.
2. An action is executed and incorporated into the plan network.
3. Inconsistencies in the plan network resulting from the executed action are calculated.
4. *Rationales* are generated as justifications for the inconsistencies, along with proposed *amendments* that will restore a consistent system state.
5. A *rationale* and an *amendment* are chosen through an interactive dialogue with the user. If no explanations are produced or considered acceptable, the exception may

represent a user error. SPANDEX is also capable of interpreting a limited set of common user errors, which are based on models of procedural error types or "slips" [7,10,11].

6. Any inconsistencies which might remain are handled by standard replanning techniques.

7. Planning and execution resume (new elaboration cycle).

In the remainder of this paper we describe in detail the detection and explanation of exceptions.

3.1 Detection of Exceptions

Given a plan network p with an identified set of expected action nodes, an action a may be executed with the resulting world state w. The execution event is denoted by (a, w). The activity descriptor a consists of an *operator* and *parameters*. The operator name is assumed to uniquely identify the activity and the parameters refer to domain knowledge base objects which are being manipulated by this activity. For example, a might be *compile-file(module-1)*. The operator in this case is *compile-file*, where the file being compiled by this activity is *module-1*.

If the specification of a unifies with one of the expected action nodes, that expected action node is processed accordingly to reflect that it has been executed, and no further changes are made at this time to the plan network. Otherwise, a node representing a is inserted into the network at the current point in execution time, so that it occurs after all executed nodes and prior to any expected action node, and resulting inconsistencies are calculated.

Therefore, the problem now posed to the SPANDEX exception handling system can be stated as follows:

Given:

- p: a partially executed plan network;
- (a,w): an event-result token;
- I: a set of calculated inconsistencies.

Produce a new successor plan-network $p\prime$ which meets the following criteria:

- The set of executed nodes in $p\prime$ include all executed nodes in p;
- $p\prime$ contains a node representing the exceptional action;
- $p\prime$ contains no inconsistencies.
- $p\prime$ has the same high-level goal as p.

Our general approach to this problem is to manipulate available domain knowledge to generate plausible *explanations* which indicate how the current network and domain knowledge base can be transformed to eliminate inconsistencies resulting from the occurrence of (a, w). Inconsistencies must be one or more of the following:

1. The action type of a doesn't match with the types of any of the *expected action nodes*.
2. The action type of a matches with the type of one of the *expected action nodes*, but the parameters of a and the parameters of the expected action node do not match.
3. As a result of changes reflected in the new world state w, the plan network may now be inconsistent. In other words, a violation may be detected of one or more of the *plan network consistency criteria* defined below:
   (a) All before-worlds and after-worlds in W are internally consistent with respect to domain constraints.
   (b) The preconditions of each plan step are satisfied in its before-world.

(c) The after-world of each plan step must be consistent with the goal of the plan step.

(d) All protection intervals in I must hold.

(e) The set of temporal ordering specifications L must be consistent.

The procedure followed up to this point (exception detection, insertion of a node representing the exception, and subsequent problem computation) is very similar to that followed by SIPE's exception handling component [14]. SIPE and other systems [1,6] have also categorized potential plan "flaws"' that may be introduced as a result of an unexpected state change. These flaws can be shown to be a subset of the above categorization of inconsistencies. In SIPE, all exceptions are treated as "mother nature" occurrences, handled by simple insertion into the plan network followed by generic recovery actions. Neither SIPE nor other replanning systems make any attempt to establish any correlation between an unexpected event and other elements of the ongoing plan, whereas the remainder of the SPANDEX task is to do exactly that. In the next section, we discuss how explanations are constructed to justify exceptions and eliminate inconsistencies.

3.2 Explanation generation

In the previous section we have enumerated the types of inconsistencies that can result from an unexpected user action. Explanations for these inconsistencies are generated by the controlled application of a set of *plausible inference rules* (PIs). Each PI maps from an inconsistent state specification S to an explanation E. S consists of a set of identified inconsistencies which are constrained by one or more specifications of relations between domain knowledge base objects. For example, the inconsistent state specification S of the PI which is used in the example in Section 4 is the following: "If the inconsistency is unexpected-action-type(a), and specialization-of(action-type(a), action-type(expected-action)), then ..." The explanation E also has two components: a *rationale*, and an *amendment*. The rationale gives a semantic basis for the exception, suggesting its contribution to the ongoing

plan. Examples of rationales are:

- This is an alternative way of performing an expected action.
- This is an alternative way of accomplishing an abstract goal or activity which is *in-progress*, which means that one or more of the subnodes of the abstract node has been executed.
- This is an alternative action which represents a shortcut in the plan (some steps may be skipped).
- Actions are being performed out of order and ordering may be relaxed.
- This is a new action which was not known to be part of the plan and should be added to the static task description.

An amendment prescribes the changes to be made in order to establish the rationale and restore system consistency. It consists of one or more of: a predefined set of *plan network alterations*, and primitive modifications on the domain knowledge base. The plan network alterations are composed of the primitive plan network operations *delete-node*, *insert-node*, *expand-node*, and *establish-ordering*. Examples of plan network alterations include: 1) replacing one of the expected actions with a node representing the unexpected action, 2) replacing a wedge containing one of the expected actions with a node representing the unexpected action, and 3) replacing a later activity node with a node representing the unexpected action and deleting the intervening nodes. The modifications that may be made to the domain knowledge base include the addition or deletion of values to a field of an object, adding or deleting a taxonomic link, or modifying a constraint.

The most likely explanations will be generated by the application of PI rules whose inconsistent state specification S holds completely in the current world model, and are referred to as *complete* explanations [3]. However, since we are interested in adding to an inherently incomplete domain model, we want also to consider rules whose inconsistent

state criteria are not entirely met; we attempt to establish the missing information through interactive dialogue, thus producing additional plausible explanations while adding to domain knowledge. In order to intelligently control the application of PI's, we use a set of heuristics similar to those applied to plan recognition problems [4]:

- **Completeness**: Prefer a plausible inference rule which has more components in its inconsistent state specification S that are true in the world model to a rule with fewer true components.
- **Locality**: Prefer a plausible inference rule that considers an expected action (or wedge) to one considering a later action (or wedge).
- **Cost**: Prefer a plausible inference rule that proposes fewer modifications in its amendment to one proposing more modifications.

A threshold is set to limit the number of explanations produced. These most likely explanations are presented to the user in an interactive fashion and a choice is requested. If none of these explanations are acceptable, the process is iterated and the next set of explanations are produced and presented, until an explanation is selected. If no explanation is selected, SPANDEX attempts to fit the exception into one of its known common error classes. If an explanation is selected, the amendments are applied, and SPANDEX must check the resulting network for consistency. Any remaining violations are handled through standard replanning techniques or through an interactive acquisition session with a human agent.

4 Example

In this section we present an example from the domain of software engineering, one of the domains which is currently implemented in SPANDEX.

The overall goal of the example task is to create a new version of a software system, incorporating desired changes and additions. A partial plan network is generated for this task, and is executed in conjunction with the relevant agents. Three ordered subgoals are generated for this task: (*decide-on-changes* (the programmer must decide which particular changes to make), *make-changes* (the editing must be performed on the appropriate modules)) are expanded and accomplished, and *have-consistent-system* (the entire software system must be updated so that changed modules are recompiled and the system is relinked).

After expanding and accomplishing the first two subgoals, the planner attempts to achieve the third subgoal *have-consistent-system* by selecting the activity *update-software-system*. Upon requesting verification from the user to perform the first primitive action in this activity expansion (*compile* the first changed file), the user denies verification and instead initiates a *unix-make* action. SPANDEX determines that an action mismatch has occurred, implying a possible attempt at an action substitution or an out-of-order action[5]. An *exception record* (see Figure 1) is created to summarize the exception. The exception analyst module of SPANDEX then uses a heuristic *rationale.selector* to choose a method to generate rationale records for the exception.

In this example, a single applicable plausible inference rule is retrieved, and SPANDEX constructs one rationale record, which represents a complete explanation (see Figure 2). In this particular case, the record states that since the activity *unix-make* is a known specialization of the activity *update-software-system*, the unexpected action may be a substitution for the more abstract activity node.

An amendment record is next constructed for the explanation which specifies the changes that must be made to the current plan network and domain knowledge in order to restore consistency to the system. The implementation of this rationale record involves replacing the wedge of the plan network subsumed by the more abstract parent node *(update-software-system-01)* with the unexpected action *(unix-make-01)*. As a side

[5]These implications are derived from relevant plausible inference rules, as described in section 3.2.

Unit-name: ACTION.TYPE.MISMATCH.01
Unit-comment: "The type of action performed did not match an expected action type."
Exceptional-action: unix-make-01
Exception-summary:
"The target action: *compile-file-01* did not occur;
unix-make-01 was performed."
Target.action:*compile-file-01*
Perceived.action *unix-make-01*
Rationales: alternative.action.rationale, out.of.order.action.rationale
Rationale.selector: rationale.selector.method
Rationale.records: alternative.action.rationale.01

Figure 1: Exception record (ACTION.TYPE.MISMATCH.01)

Unit-name: ALTERNATIVE.ACTION.RATIONALE.01
Unit-comment: "The unexpected action is an alternative
to an in.progress.parent.node of an expected action."
Rationale-summary: "The unexpected action *unix-make-01* is sufficient
since its activity type is a specialization of an in-progress activity node
update-software-system-01. "
Status: complete
Rationale.type: Alternative.action.rationale
Subrationale.type: Specialization.of.in.progress.node
In.progress.parent.activity: update-software-system-01
Pending.goal.achieved: updated(SPANDEX)
Unexpected.action.goal: consistent(SPANDEX)
Hierarchy.level.difference: 2
Amendments: Replace.plan.wedge.01

Figure 2: Rationale record for ACTION.TYPE.MISMATCH.01

Unit-name: REPLACE.PLAN.WEDGE.01
Unit-comment: "Replace a wedge of the plan subsumed by a single node by a new node."
Amendment-summary: "Replace the plan wedge subsumed by
update-software-system-01 with *unix-make-01*."
Implementation: (do
(replace-wedge update-software-system-01 unix-make-01)
(deactivate compile-file-01 compile-file-02 compile-file-03
link-system-01))

Figure 3: Amendment record for ALTERNATIVE.ACTION.RATIONALE.01

effect, the nodes in the expansion of *update-software-system-01* are deactivated from the planner's predictions (see Figure 3).

References

[1] Ambros-Ingerson, J.A., Steel, S. "Integrating Planning, Execution, and Monitoring," *Proceedings of AAAI-88*, Minneapolis-St. Paul, Minnesota, pp. 83-88.

[2] Broverman, C.A., Croft, W.B. "Exception Handling During Plan Execution Monitoring," *Proceedings of AAAI-87*, July 1987, Seattle, WA.

[3] Broverman, C., Croft, W.B. "Plausible Explanations to Cope with Unanticipated Behavior in Planning," COINS Technical Report 88-56, University of Massachusetts, Amherst, Ma. June 1988.

[4] Carver, Norman, Victor Lesser, and Daniel McCue, "Focusing in Plan Recognition," *Proceedings of AAAI-84*, 1984, 42–48.

[5] Broverman, C., Croft, W.B. "SPANDEX: An Approach Toward Exception Handling in an Interactive Planning System," COINS Technical Report 87-127, University of Massachusetts, Amherst, Ma. December 1987.

[6] Hayes, P.J. "A Representation for Robot Plans", *Proceedings IJCAI-75*, 181-188, 1975.

[7] Hollnagel, E. "Action Not as Planned: The Phenotype and Genotype of Erroneous Actions," draft, Computer Resources International, Copenhagen, Denmark, 1987.

[8] Lefkowitz, L.S. and Croft, W.B. "Planning and Execution of Tasks in Cooperative Work Environments," *Proceedings of the 5th IEEE conference on Artificial Intelligence Applications*. March 1989 (to appear).

[9] Pednault, E.P. "Formulating Multiagent, Dynamic-World Problems in the Classical Planning Framework," *Proceedings of the 1986 Workshop on Reasoning About Actions and Plans*. Timberline, Oregon, pp. 47-82.

[10] Rasmussen, J. "What Can Be Learned from Human Error Reports?" In K. Duncan, M. Gruneberg, and D. Wallis (Eds.), Changes in Working Life. John Wiley: London. 1980.

[11] Reason, J., Mycielska, K. Absent-Minded? The Psychology of Mental Lapses and Everyday Errors. Prentice-Hall, Inc., 1982.

[12] Sacerdoti, E.D. *A Structure for Plans and Behavior*, Elsevier North-Holland, Inc., New York, NY, 1977.

[13] Tate, A. "Generating Project Networks", *Proceedings IJCAI-77*, Boston, 888-893, 1977.

[14] Wilkins, D.E. Practical Planning: Extending the Classical AI Planning Paradigm. Morgan-Kauffman Publishers, San Mateo, CA. 1988.