Interpreting Unexpected Actions by Human Agents During Plan Execution Monitoring

Carol A. Broverman
W. Bruce Croft

COINS Technical Report 90–31
(updated from COINS TR 89–83)
April 1990

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). Instead of relying on a traditional error recovery approach, we advocate the use of a theory of human behavior, combined with 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 with particular regard to the special role of human agents. We describe SPANDEX, an implementation of our approach. The SPANDEX system uses a set of *plausible rationales* to produce explanations which dictate *amendments* to be made to the plan and domain model in order to incorporate the exception. Planning and execution can then continue, often with an improved domain model.

This research was supported in part by a contract with Triumph-Adler AG and Ing. C. Olivetti and by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, NY 13441-5700.

Contents

1 Motivation

Planners can be used to automate or support a variety of complex tasks [9, 14, 18]. 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 crucial and systems should have the capability to respond to unexpected change. Recovery measures such as those of [1, 7, 18] have been proposed to effectively replan around arbitrary unanticipated domain state changes, and other transformational approaches have been developed which hypothesize agent-less processes as responsible for unpredicted effects that are observed [16, 17, 11].

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 [9]. In such environments, human input is required to guide the development of a plan for a task as well as to execute plan steps. The natural intelligence and familiarity of humans with the application domain means that their actions, even when inconsistent with system expectations, are generally purposeful. In addition, models of realistic domains are inherently incomplete. Therefore, we believe that human-generated plan exceptions may be valid actions and introduce new domain information. Such exceptional actions should be incorporated into the developing plan rather than "undone" using replanning techniques.

The primary goal of our approach is to determine, where possible, the relationship of an exceptional action to the semantics of the current plan, and thus to provide an explanation which will allow the continuation of planning and execution [3, 4]. When an exception occurs, the domain model, along with a theory of human procedural behavior, is used to transform the current plan into a valid alternative, i.e., recognize an alternate plan. Subsequent replanning can thus be avoided or reduced, and additional domain knowledge may be acquired in the process.

In this paper, we describe a theory of human procedural behavior, from which we extract a set of *plausible rationales* that can underlie exceptional agent behavior during plan execution. We then give a formal model of plan execution and define the exception problem. A general architecture is presented which has been designed and implemented to handle the exception problem. We show how the plausible rationales identified by our theory are used as the basis for explanations of exceptions. Evidence is collected to establish these rationales as justification for the exceptional behavior. Corrective measures are then proposed in the form of *amendments* to the current plan representation and to the domain theory. 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 A theory of human procedural behavior

One of the major contributions of this work is to address the human component in plan execution and plan revision. To do this, we have adopted a theory of human procedural behavior that is closely tied to research analyzing deviations of human agents from typical plans of action. We base this theory on data made available through studies identifying aberrations that occur when humans perform procedural tasks [13, 10, 12, 8].

We adopt a unique perspective of what many have termed as "errors" when syntactically detecting unanticipated deviations in a plan monitoring context. This perspective is predicated on two key points: (1) A computer-generated plan is often based on an inherently incomplete model of the domain; and (2) We operate under the assumption that human agents behave intelligently and purposefully. In the domains in which human errors have been studied, the violated procedures are often fixed and the procedural knowledge is bounded. Therefore, a deviation from a plan of action in these settings is almost always regarded as an incorrect action. However, when an inherently incomplete domain model is used to generate the plan of action, the identical syntactic deviation detected during plan execution may in fact represent an intentional and valid step towards the pursuit of the overall plan goals. The recognition, through explanation, of this action as valid can have the side-effect of adding to the incomplete model of the domain.

Several researchers have analyzed the ways in which human agents deviate from plans during their execution. Four major taxonomies of "error" have emerged, which are closely related and exhibit a degree of overlap:

1. A detailed empirical study of "actions not as planned," resulting in several categories of error [13];

2. A categorization of both cognitive sources of error as well as subcategories of error types; along with a proposal for a set of possible system response strategies [10];

3. A layered mechanistic model suggesting the role of the underlying bases for error as

well as their external manifestations [12];

4. A taxonomy of error manifestations, distinguishing between the underlying cause of error and the manifestation [8].

Although these taxonomies generally refer to such deviations as "errors," we prefer to use the term *exceptions* to avoid an overriding negative connotation. As already suggested, some of these exceptions may be purposeful and valid, especially in the context of an underspecified domain model. Also, it is important to distinguish between the surface manifestation of an unexpected action during the execution of a procedure and the intent which may have generated the action. In general, the taxonomies we examined intermix these two aspects of exceptions, although Hollnagel does underscore the importance of distinguishing between the syntactic appearance of an exception (*phenotype*) from its underlying cause (*genotype*) [8].

Most of the exceptions identified by these taxonomies fall into one of four general categories: repetition, use of wrong object, intrusion, and ommission. Particular examples of these categories are: performing actions out-of-order, skipping steps, reversing steps (known as behavioral spoonerisms), exchanging resources with similar functions, exchanging resources with physical similarities, inserting extra actions, and repeating actions [13]. We hypothesize constructive intentions motivating these actions, and have derived a set of *plausible rationales* to represent these intentions. The plausible rationales are then used as the basis of explanation when considering the role of an unexpected event in the current plan context. The rationales can be grouped into the following categories:

1. **Knows alternative.** The agent is performing an action which is an alternative way to achieve an action anticipated by the planner.

2. **Knows shortcut.** The agent knows a way to skip some of the anticipated steps of the plan and still achieve the high-level goal. This may involve accomplishing an abstract plan goal in "one fell swoop"[1].

[1]This is the rationale contained in the plausible inference rule used in the example presented in Section

3. **Can reorder.** The agent is able to perform the anticipated plan steps but in an alternative order than that predicted by the planner. The original ordering may be overconstrained.

4. **Knows reordering shortcut.** The agent is able to perform an action which accomplishes a number of plan steps that are anticipated later in the plan.

5. **Introduces new action.** The agent is introducing a new action that is not anticipated by the planner but is part of achieving the high-level goal (and should be part of the static task description).

6. **Introduces redundant action.** The agent is introducing a new action that is not anticipated by the planner, nor is part of achieving the high-level goal, but does not harm the ongoing plan in any way.

7. **Use alternative resource.** The agent is performing an action that is using a resource similar to the resource we expected to use.

In Section 6, we discuss how these rationales are used to construct explanations of exceptions which restore the plan to a consistent state.

3 Planning and Execution

In this section, we give a formalization of planning and plan execution, and define terminology relevant to exception handling. A planning problem is specified as: (1) an initial world state s_i, (2) a desired final goal state s_f, and (3) a set of domain activities A which can be applied in some order to s_i to produce s_f. A planner iteratively elaborates the top-level goal representing the final goal specification s_f. The result of each of these manipulations is represented by a *plan network*, which is viewed as the current version of an evolving plan. A plan network is a strict partial order and consists of the following elements:

7.

- N: a set of nodes, where each node is either a *goal* node (representing a decomposable activity), or an *executable* node (representing a primitive action that can be executed by a tool or human agent);
- 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 model. 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 (for an executable node) or achievement (for a goal node) of that node. Each node also contains a set of conditions C that must hold in the before-world of that node before it can be processed by the planner;
- 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.

We further define a set of criteria which must hold in order for a plan network to be consistent:

- All before-worlds and after-worlds in W are internally consistent with respect to constraints in the domain model.
- All activity preconditions (a subset of C) specified by nodes in the current plan must be satisfied in the before-worlds of their respective nodes.
- Constraints imposed on resources used by an activity (a subset of C) must be satisfied in the before-world of each node in an expansion of that activity.
- The after-world of each plan node must be consistent with the goal of that node.
- All protection intervals in I must hold.
- The set of temporal ordering specifications L must be consistent.

We define the process of *planning* as iterative *transformations* on plan networks. Since planning is interleaved with execution, we adopt a broad definition of planning that subsumes execution. Therefore, we define a *final plan* as a plan network which has been fully ordered, and every node is either a *phantom*[2] or it is a primitive activity node that has been executed. Thus, a plan network represents a class of final plans; many different possible final plans can result depending on the subsequent choices of elaborations and operations. Conceptually, a new plan network results each time an operation is performed by the plan network maintenance system (PNMS [2]), such as node expansion, node execution, node ordering, or protection interval establishment. The complete *plan history* is a lattice containing all intermediate plan networks ordered within planning time, where the distinguished upper bound is the eventual final plan. The relation is a partial order since backtracking may be allowed. The plan history maintains a record of all planning actions performed.

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 nodes* which are eligible for execution. A node is in this set if and only if: (1) it is a primitive activity node, (2) all of the conditions C specified by the node are satisfied in the before-world of that node, and (3) all of the node's necessary predecessors are complete and awaiting successors[3].

4 Detection of Exceptions

Given a plan network p with an identified set of expected action nodes, a human agent or tool may execute an action a with the resulting world state w. The execution event is denoted by (a, w). If the specification of (a, w) unifies with one of the expected action nodes, that expected action node is processed accordingly to reflect that it has been executed, and no

[2]A phantom node is a goal node which has been determined to be true at its position in the plan without further expansion and execution [18].

[3]A complete definition can be found in [9].

further changes are made at this time to the plan network. Otherwise, a node representing the event 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.

Two classes of inconsistencies are detected by our system. The first class is detected by the execution monitor when performing a rudimentary comparision of what was executed by an agent with what was anticipated by the planner. There are two types of inconsistencies in this category: (a) The action a doesn't match with the action specified by the expected action node chosen for processing, and (b) The state w achieved does not match the goal state of the expected action node chosen for processing.

The second class of inconsistencies are those pertaining to the state of the current plan network. As a result of asserting the new world state w, the plan network may now be inconsistent. Specifically, one or more of the plan network consistency criteria defined in Section 3 may have been violated. The union of these two classes of inconsistencies results in a taxonomy which is broader than other established categorizations of plan flaws [1, 18].

The problem can now be posed to the SPANDEX exception handling system as follows:

- **Given:** (1) p: a partially executed plan network; (2) (a,w): an event-result token; and (3) X: a set of calculated inconsistencies;
- **Compute:** a new successor plan network p' which meets the following criteria: (1) The set of executed nodes in p' include all executed nodes in p; (2) p' contains a node representing the exceptional event; (3) p' is consistent, and (4) p' has the same top-level goal as p.

5 System architecture

Our general approach to this problem is to manipulate available domain knowledge to generate plausible explanations which indicate how the current network and domain model can

be transformed to eliminate inconsistencies resulting from the occurrence of *(a, w)*. The architecture of a system designed to handle this problem is shown in Figure 1.

When an exception is detected by the plan execution monitor, the *exception classifier* is invoked to compute the membership of the exception within a predetermined set of exception classes. These exception classes are based on the characteristics of the mismatch between the planner's expectations and the agent's action. The *plan critic* determines if achieved or protected goal states have been violated in the plan as a result of the exception. The *replanner* handles *unaccountable* exceptions (generated by unknown agents who are represented by *world* in Figure 1) using plan repair methods based on those of [18]. The procedure followed up to this point is similar to the handling prescribed by other systems (e.g., SIPE). However, SPANDEX goes beyond other replanning systems by attempting to establish correlations between an unexpected event and other elements of the ongoing plan.

In SPANDEX, the *exception analyst* applies domain knowledge according to a set of prespecified *plausible rationales* to construct explanations of *accountable* exceptions (generated by known agents and represented by *users* in Figure 1). The rationales are based on the model of human procedural behavior described in Section 2, and the mechanisms of the exception analyst which use these rationales to produce explanations are described in Section 6.

Since several agents can be affected by an exception, we propose the use of negotiation [6, 15] to establish a consensus among them regarding the explanation and proposed plan modifications. The *negotiator* identifies the affected agents and uses the information provided by the exception analyst to conduct a dialogue. The output of the negotiation phase is a selected explanation, along with approved changes or *amendments* to be made to either the plan network instantiation or to the permanent plan library.

The *explanation generalizer* produces a generalized form of the verified explanation, using taxonomic information in the domain model. This new knowledge about domain activities and the amendments resulting from negotiation are passed to the *domain model modifier* for implementation. Thus, a successful negotiation can result in a system which has "learned,"

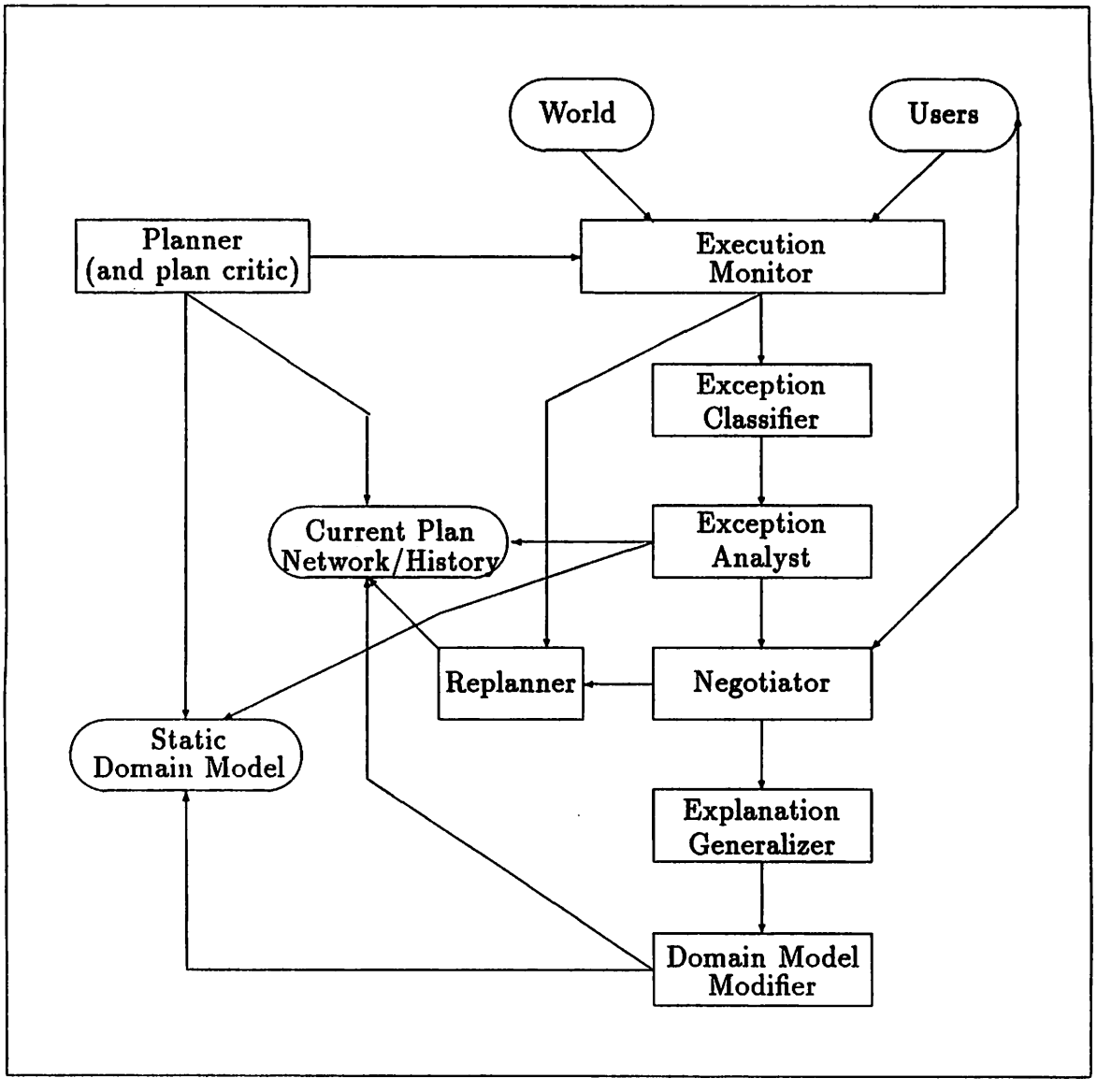

Figure 1: An architecture for planning and exception handling

that is, the static domain model may be augmented with knowledge about the exception and thus the system is able to handle future similar exceptions.

6 Explanation generation

In this section, we discuss the *exception analyst* component of our general architecture in more detail. Explanations of an exceptional action are generated by the controlled application of a set of *plausible inference rules* (PIs). Each PI maps from a state specification S to an explanation E. The specification of S consists of a set of predefined indicators and inconsistencies which are specified as: (1) relations between objects in the domain model, (2) status of execution monitor state variables, or (3) qualifications on the plan network state. For example, the state specification S of one PI (that used in the example in Section 7) is: "If *exception.type(exception)* = *action.mismatch*, and *specialization-of(action-type(a), action-type(expected-action))*..." 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. The set of rationales which are currently used by the system are described in Section 2.

An amendment prescribes the changes needed to establish the rationale and restore system consistency. It consists of one or more plan network alterations and primitive modifications of the domain model. Examples of plan network alterations include: (1) replacing one of the expected actions with a node representing the unexpected action, (2) replacing a wedge[4] containing one of the expected actions with a node representing the unexpected action, and (3) replacing a later node with a node representing the unexpected action and deleting the intervening nodes. The plan network alterations are composed of the primitive plan network operations *delete-node*, *insert-node*, *expand-node*, and *establish-ordering*. Possible domain model modifications are the addition or deletion of values to a field of an

[4]We define the concept of a *plan wedge* as a node in plan history along with its descendants (determined recursively) which have been introduced through one or more levels of plan expansion. This definition is based on that used by Wilkins [18] and Sacerdoti [14].

object, the insertion or removal of a taxonomic link, or the modification of a constraint.

Explanations can be either *complete* or *partial*. A complete explanation results from the application of a PI rule whose state specification S holds completely in the current world model, while a partial explanation results from a PI rule having one or more unconfirmed indicators in its state specification. Since we are interested in adding to an inherently incomplete domain model, we consider partial explanations. An attempt is made to confirm missing indicators through interactive dialogue, producing additional plausible explanations while adding to the domain model.

In order to intelligently control the application of PI's and the presentation and selection of the resulting explanations, we use a set of heuristics similar to those applied to plan recognition problems [5]:

- **Completeness**: Prefer a plausible inference rule with more confirmed components in its state specification S.
- **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 checks the resulting network for consistency. Any remaining violations are handled by the replanner or through an interactive acquisition session with a human agent.

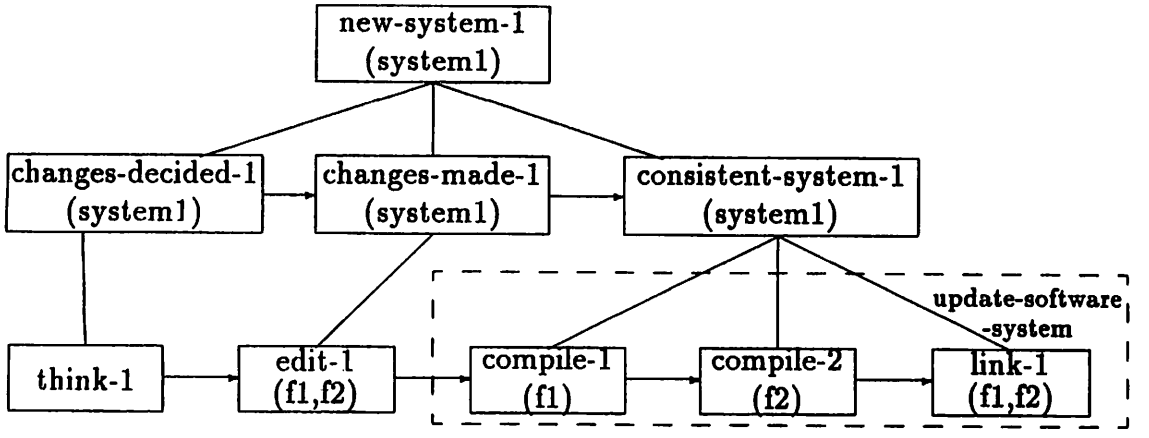

Figure 2: Plan Network for Example Software Task

7 Example

In this section we present an example from the domain of software engineering, one of the domains 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. The initial plan network generated for this task contains three ordered subgoals: (*changes-decided-1* (the programmer must decide which particular changes to make), *changes-made-1* (the editing must be performed on the appropriate modules), and *consistent-system-1* (the entire software system must be updated so that changed modules are recompiled and the system is relinked) (see Figure 2).

After expanding and accomplishing the first two subgoals, the planner attempts to achieve the third subgoal *consistent-system-1* 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 *f1*), the user vetoes the proposal and instead initiates a *unix-make* action. SPANDEX determines that an action mismatch has occurred, implying a possible attempt at an action substitution, an out-of-order action, or an extraneous action[5]. An *exception record* (see Figure 3) is created to summarize the exception. The exception

[5]These implications are derived from relevant plausible inference rules, as described in Section 6.

Unit.name: EXCEPTION.RECORD.1
Exception.type: ACTION.TYPE.MISMATCH
Unit.comment: "The type of action performed did not match an expected action type."
Exception.summary:
"The suggested.action: *compile-file-1* did not occur;
unix-make-1 was the performed.action."
Suggested.action:*compile-file-1*
Accomplished.action *unix-make-1*
Rationale.selector: rationale.selector.method
Complete.rationale.records: subsumes.current.step.1, additional.step.to.achieve.current.goal.1

Figure 3: EXCEPTION.RECORD.1

analyst module of SPANDEX then uses a heuristic *rationale.selector* to choose a method to generate explanations (represented as *rationale records*) for the exception.

Two complete explanations result from the application of retrieved PI rules; the one chosen interactively through negotiation with the user is shown in Figure 4. In this particular case, one of the key indicators in the plausible inference rule that applied involves activity specialization. An activity *activity-1*is defined to be a specialization of another activity *activity-2*if both activities accomplish the same goal, but *activity-1* specifies a more restrictive precondition or additional constraints on resources used[6].

The chosen explanation record states that since the activity *unix-make* has been verified to be a specialization of the activity *update-software-system*, this unexpected action may be a substitution for the abstract goal node which derived the *update-software-system* expansion (see Figure 2). In addition, this substitution subsumes the expected action.

An amendment record is next constructed for the explanation which specifies the changes that must be made to the current plan network and domain model in order to restore consistency to the system (see Figure 5). The implementation of this rationale record involves replacing the wedge of the plan network subsumed by the more abstract parent node *(consistent-system-1)* with the unexpected action *(unix-make-1)*. As a side effect, the nodes in the expansion of *consistent-system-1* are deactivated from the planner's predictions.

[6]Other definitions of activity specialization are possible.

Unit-name: SUBSUMES.CURRENT.STEP.1
Unit-comment: "The unexpected action is an alternative
to an in.progress.parent.node subsuming an expected action."
Rationale-summary: "The unexpected action *unix-make-1* is sufficient
since its activity type is a specialization.of *update-software-system*,
used to achieve in.progress.parent.node *consistent-system-1*. "
Status: complete
Comparison.node: *consistent-system-1*
Indicators: Specialization.of.in.progress.activity,
Equivalent.achieved.goal.and.comparison.goal
In.progress.parent.activity: update-software-system
Inconsistencies: Action.mismatch
Hierarchy.level.difference: 2
Amendments: Replace.plan.wedge.1

Figure 4: Rationale record for EXCEPTION.RECORD.1

Unit-name: REPLACE.PLAN.WEDGE.1
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
consistent-system-1 with *unix-make-1*."
Implementation: (do (replace-wedge consistent-system-1 unix-make-1)
(deactivate compile-file-1 compile-file-2 link-system-1))

Figure 5: Amendment record for SUBSUMES.CURRENT.STEP.1

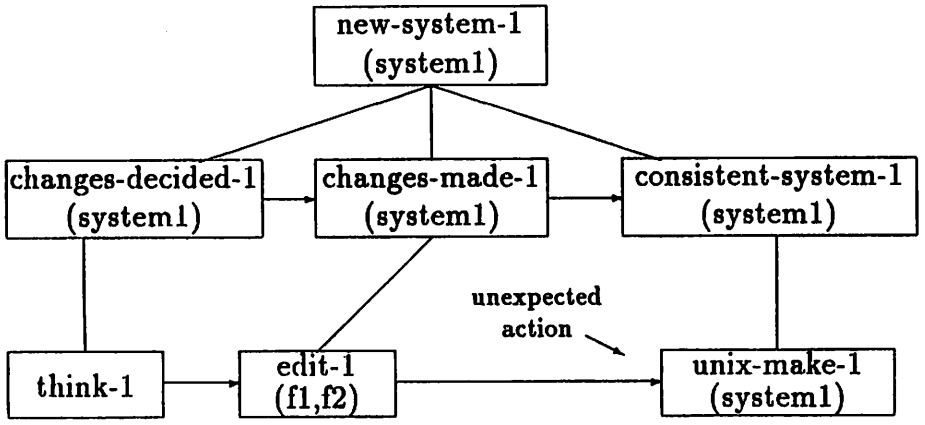

Figure 6: Plan Network for Resolved Exception

The new plan network which results from the implementation of the amendment just described is shown in Figure 6.

8 Status and Conclusions

Plans generated by a nonlinear hierarchical planner are often based on an inherently incomplete domain model. Run-time discrepancies are therefore inevitable, and we have presented an approach to handle such exceptions encountered during plan execution. Previous approaches have focused on general replanning tactics which pay no attention to the possibility of an intentional semantic basis for unanticipated actions or results. In contrast, we adopt the default assumption that the behavior of human agents is purposeful, even when it may be inconsistent with system expectations. We have presented a theory of *plausible rationales* that has been operationalized for use as an exception detection and correction mechanism. Using this theory, we construct explanations of exceptional events and amend the ongoing plan to restore consistency. Extensive replanning can be avoided, and new information can be acquired to refine the domain model. A Common-Lisp prototype demonstrating our approach is implemented on a TI-Explorer. In addition, user studies are underway to evaluate the adequacy and coverage of our extended exception taxonomy and the rationales used for explanation.

References

[1] Ambros-Ingerson, J.A. and S. Steel. "Integrating Planning, Execution, and Monitoring," *Proceedings of AAAI-88*, Minneapolis-St. Paul, Minnesota, 1988, pp. 83–88.

[2] Beetz, M. and L.S. Lefkowitz, "Reasoning about Justified Events: A Unified Treatment of Temporal Projection, Planning Rationale and Domain Constraints", Collaborative Systems Laboratory Technical Report CSL-89-6, University of Massachusetts, 1989.

[3] Broverman, C.A. and W.B. Croft, "Exception Handling During Plan Execution Monitoring," *Proceedings of AAAI-87*, July 1987, Seattle, WA, pp. 190–195.

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

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

[6] Fikes, R.E. "A Commitment-based Framework for Describing Informal Cooperative Work", *Cognitive Science*, 6: 331-347; 1982.

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

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

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

[10] Norman, Donald A., "Categorization of action slips," *Psychological Review*, 88(1):1–15, January 1981.

[11] Rajamoney, S., G. DeJong, B. Faltings, "Towards a model of conceptual knowledge acquisition through directed experimentation," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 688–690, Los Angeles, CA, 1985.

[12] 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.

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

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

[15] Sathi, A., T.E. Morton, S.F. Roth, "Callisto: An intelligent project management system," *AI Magazine*, 7(5):34–52, Winter 1986.

[16] Simmons, R. and R. Davis, "Generate, Test, and Debug: Combining Associational Rules and Causal Models," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* Milan, Italy, pp. 1071–1078, 1987.

[17] Simmons, R.,"A theory of debugging plans and interpretations," *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 94–99, 1988.

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