Plausible Explanations to Cope with Unanticipated Behavior in Planning

Carol A. Broverman
W. Bruce Croft

COINS Technical Report 88–56
June 1988

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

Complex tasks can be accomplished efficiently by human agents with the assistance of a hierarchical nonlinear planner. However, since such a paradigm implies an active role of the human agents in making decisions and providing information, an interactive interface must be prepared to encounter unusual behavior which deviates from system expectations. This paper describes a system (SPANDEX) which constructs *plausible explanations* as justifications for agent behavior. The verification of an explanation can involve a reorganization of existing knowledge or require the acquisition of additional knowledge. Selected explanations are implemented through proposed *amendments* to the knowledge base. Detailed examples of the operation of the system are presented.

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 Introduction

No matter how carefully a plan is conceived, things frequently go wrong during its execution. When human agents are responsible for executing plan steps this problem is of particular importance. People change their minds or opportunistically revise a plan mid-execution. In addition, they are prone to error and misjudgement. Consequently, a system designed to support the performance of human tasks in an interactive setting [4,5] must be prepared to cope with frequent unanticipated occurrences (*exceptions* [2]) during the planning and execution process.

In [2], we describe an interactive planning system and its requirements for a general exception handling mechanism. In this setting, the input of human agents is required for task completion and thus exceptions can be generated by the actions of known agents. In particular, an agent may perform an action which is inconsistent with system predictions. For example, an agent may leave out a step in a task as a deliberate short cut, or he may perform an unexpected action as an intentional substitution of an expected action. In other cases, the action of an agent may not be an intentional aberration, but may be viewed initially as an exception due to an incomplete or incorrect domain plan library.

Such exceptions are referred to as *accountable* since it is presumed that *agents behave purposefully* and there are *motivations*[1] for their behavior. Actions which initially appear to be "errors" can often be recognized and explained as actions consistent with the goals of the plan. We contrast this class of accountable exceptions with the more frequently addressed arbitrary changes in world state brought about by unknown agents. For example, a system that is planning a travel itinerary may have to contend with the effects of an earthquake which has forced the cancellation of a scheduled train. We refer to this latter type of exception as *unaccountable* and recognize replanning as an effective approach for plan recovery [11,15].

Accountable exceptions, however, should be justified rather than "counteracted" through replanning. Explanations of unanticipated agent behavior can result in improvements in both system understanding and performance. As an initial step towards achieving this aim, we have defined the categories of accountable exceptions that can arise in a cooperative planning framework. The types of exceptions defined by this behavioral perspective are: *action not in plan, out of order action, repeated action, user assertion*, and *expected action*

[1] What constitutes a relevant agent motivation may vary from one domain to the next, since the policies and constraints of the work setting are influential. For example, an environment with strong financial incentives may encourage individuals to act in ways to cut costs even at the expense of lengthy tasks, while in other settings financial expense may be considered as secondary to task simplification. Other possible motivations for deviating from expected procedure include: anticipation, partial achievement of an expected plan step, and special case handling.

with parameter causing constraint violation. A complete description of this taxonomy and additional groundwork for the approach described in this paper can be found in [3].

In this paper we investigate the construction of *plausible explanations* of accountable exceptions. Section 2 describes the implemented system SPANDEX (Support for Planning and EXception handling) and its interface to POLYMER, a hierarchical planning system similar to NOAH [12] and NONLIN [14]. SPANDEX allows a planning system to continue the planning and execution of a task after encountering an exception. The approach described here facilitates the extension of an existing knowledge base to accommodate alternative ways to complete task goals, as learned through the handling of previous exceptions. In section 3, we introduce the concept of *plausible explanations* as justifications of presumed agent motives and describe how they are constructed. Section 4 discusses how *amendments* to the knowledge base are proposed to restore consistency to the plan network upon acceptance of an explanation. An example of the operation of SPANDEX in the domain of journal editing is presented throughout for illustration, and an additional example from the software development domain is given in the Appendix.

2 An architecture to support exception handling

POLYMER [5,6] is an interactive planning system designed to assist in the management of tasks in a cooperative setting. It uses a hierarchical planner to construct a procedural net that specifies the sequences of actions required to achieve a goal. POLYMER constructs partial plans and executes them in cooperation with agents. The actual actions taken by these agents are compared to expected actions, and when differences are found (producing an exception) SPANDEX is invoked. This architecture is shown in Figure 1. In this section, we describe the SPANDEX architecture and introduce an example to illustrate the mechanisms described.

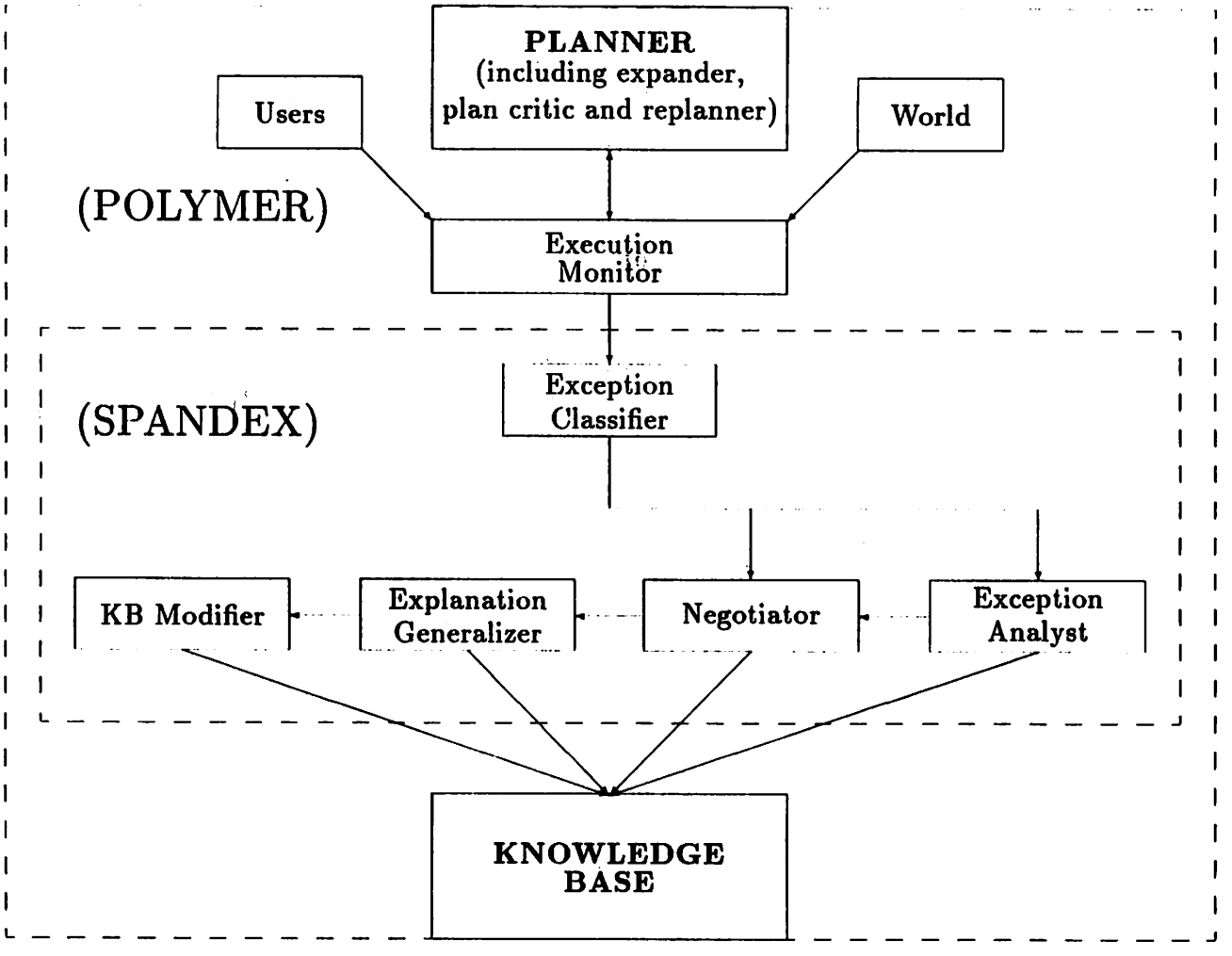

Figure 1: Architecture for a cooperative planning system

2.1 The SPANDEX architecture

When an exception is detected by the plan execution monitor, the *exception classifier* is invoked to determine the exception type. The *plan critic* determines if goal nodes have been violated in the plan as a result of the exception. The *replanner* handles unaccountable exceptions (generated by *world* in Figure 1).

The *exception analyst* applies domain knowledge to construct *plausible explanations* of accountable exceptions. The function of the exception analyst is described in more detail in [2,3] and plausible explanations are discussed further in the following section. Since several agents can be affected by an exception, we propose to use negotiation [10,13] 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 to be made to either the plan network for this particular instance or to the permanent plan library.

The *explanation generalizer* produces a generalized form of the verified explanation, using taxonomic information in the knowledge base. This new knowledge about domain activities, along with any suggested knowledge base changes resulting from negotiation, is passed to the *knowledge base modifier*. Thus, a successful negotiation can result in a system which has "learned," that is, the static domain plans may be augmented with knowledge about the exception and thus the system is able to handle future similar exceptions.

2.2 An example

To illustrate the mechanisms discussed in this paper, we will develop an example involving a journal editing task. The goal of the task is to decide whether or not to accept a paper for publication. The primary agent initiating the task is the editor, who has received a paper for review in the area of artificial intelligence. POLYMER generates a partial plan network for this task, and executes it in conjunction with the relevant agents. There are

three ordered subgoals generated for this task: *reviewers-selected* (the editor must select reviewers to judge the submission), *reviews-received* (the actual reviews must be obtained), and *decision-reached* (the editor must make a final decision based on the responses). A constraint on the *reviewers-selected* task subgoal specifies that each reviewer selected must have an area of expertise which matches the area of the paper to be reviewed.

The agent (editor) is instructed to perform the activity *select-reviewer* to achieve the first subgoal. The activity is performed and the reviewer selected is *Seymour.Wright*. When the constraint is evaluated, it is discovered that the area of expertise of *Seymour.Wright* is *computer.vision* rather than *artificial.intelligence*. SPANDEX is invoked by POLYMER upon detection of this exception, and the exception classifier determines that a constraint which was dynamically posted on a knowledge base object has been violated (a *dynamic object constraint violation* exception). The *exception record* pictured in Figure 2 is generated to summarize the exception. The exception classifier first records the action which generated the exception, and the type of exception which was detected. The actual constraint which was violated (*target.constraint*) is recorded, and SPANDEX also determines the actual value in the knowledge base object that triggered the constraint violation (contained in *perceived.constraint*). SPANDEX then invokes the exception analyst, which uses the *strategy.selector* to choose one of the *strategies* to generate *explanations* for the exception.

In the next section, we describe how explanations are generated for exceptions, and illustrate the behavior of the exception analyst on the example introduced above.

3 Plausible explanations

The term *explanation* is broadly used in current artificial intelligent research. In *explanation-based learning* [7,8,9], explanations are generated as *proof* that a sequence of steps achieves its goal. This type of explanation is logically sound, and is in effect a restructuring of existing knowledge. In POLYMER we also would like to construct explanations which justify how

Unit-name: DYNAMIC.OBJECT.CONSTRAINT.VIOLATION13
Unit-comment: "A constraint dynamically posted on a KB object has been violated. "
Exceptional-action: select-reviewer1
Exception-summary:
"The target constraint: *(area.of.expertise Seymour.Wright artificial.intelligence)*
was not met; it is the case that *(area.of.expertise Seymour.Wright computer.vision)*."
Target.constraint:*(area.of.expertise Seymour.Wright artificial.intelligence)*
Perceived.constraint *(area.of.expertise Seymour.Wright computer.vision)*
Strategies: dynamic.object.constraint.strategy
Strategy.selector: strategy.selector.method
Explanations: specialization.constraint.values.expl14, generalization.constraint.values.expl15

Figure 2: *Dynamic.object.constraint.violation13*

exceptions contribute toward current plan goals. Our approach, however, is not to supply a rigorous proof, but rather to suggest plausible roles for the exception in the context of the plan. Due to the interactive nature of our planner, we often have only a partial action sequence available for analysis, and assume a potentially incomplete knowledge base. Therefore, we are not always able to produce formal explanations of exceptions through the reorganization of existing knowledge, but must rely on the construction of potentially valid (*plausible*) explanations which may require user validation and additional knowledge for verification.

The task of the exception analyst is to apply a set of algorithms to the knowledge base in order to produce plausible explanations of exceptional behavior. The algorithms are based on *plausible inference rules* and conduct a controlled exploration of a rich and integrated representation of domain activities and objects [2]. In this section, we define what we mean by explanations, and show how they are constructed, using the example introduced in the previous section.

3.1 Plausible inference rules

For a given exception type, a set of plausible inference rules are retrieved. These rules are intended to reflect possible motivations of the responsible agents and are used to construct explanations for the exception. The general form of a plausible inference rule is a set of conditions (forming the premise of the rule) followed by the established rationale for allowing the exception (conclusion). For example, one plausible inference rule used during the resolution of an exceptional action resulting in a constraint violation is the following:

> **Plausible-inference-rule1**: If the violating value in a constraint predicate is a specialization of the target value of the violated constraint, then the violating value may suffice as a substitution for the target value.

In this case, we have a single condition in the premise (specifying a taxonomic relationship that must exist between actual and target values in a constraint), but in the general case there may be several conditions. Each condition specifies a semantic relationship which must hold among one or more entities in the knowledge base. These relationships include: direct specialization, direct generalization, sibling or cousin taxonomic relationships, and causal relationships. These and other semantic relationships are discussed in more detail in [3]. Parameters are associated with some of the condition types; for example, when a specialization relationship is established in an explanation, the number of taxonomic links between the two objects is recorded (a measure of closeness), or for an established generalization relationship, it may be the case that the more specific object has five additional attributes (a measure of similarity). The values of the condition parameters establish the *degree of plausibility* of each explanation to enable the ranking of multiple explanations.

3.2 Explanations

Each plausible inference rule retrieved for an exception gives rise to the instance of an *explanation*. Therefore, there may be many possible explanations for a given exception. An explanation is considered *complete* if all of the conditions in the premise can be substantiated by SPANDEX. If one or more of the conditions cannot be verified, the explanation is *incomplete*. Complete explanations are a result of analyzing the existing knowledge to infer information from existing facts. Incomplete explanations, on the other hand, represent lines of reasoning that may result in valid explanations of an exception if we are able to verify the missing conditions. In other words, the construction and acceptance of a complete explanation involves a *reorganization* of existing knowledge[2], while incomplete explanations require the *acquisition of new knowledge* in order to be substantiated and accepted. Complete explanations are preferred since they are already verified and can allow the successful completion of a plan in progress, but incomplete explanations can potentially lead to improved system performance through the acquisition of new knowledge.

When an explanation is considered to be incomplete, an attempt to complete it may be warranted for one of two reasons. First, the knowledge base may be incomplete; an unverified relationship or fact may be simply *unknown*, and might be acquired through a dialogue with a responsible agent. Secondly, the knowledge base may be incorrect; a condition in an explanation which is false could be established as true through a dialogue with an agent. Thus, the negotiation involving incomplete explanations plays a primary role in both the expansion and debugging of the initial knowledge base.

In the example introduced in section 2.2, SPANDEX constructs two plausible explanations, one which is complete, and one which is incomplete (see Figure 3). In this particular case, the complete explanation states that since the referee selected has an *area.of.expertise* which is a specialization of the required *area.of.expertise*, he is sufficiently qualified to review the paper of concern. Note that since the specialization relationship

[2]Note that complete explanations are similar to those explanations produced by the explanation-based learning paradigm discussed earlier.

Unit-name: SPECIALIZATION.CONSTRAINT.VALUES.EXPL15
Unit-comment: "Show that the actual value of the violated constraint predicate is a specialization of the target value in the constraint."
Explanation-summary: "The actual value of the *area.of.expertise* field of *Seymour.Wright* (*computer.vision*) is sufficient since it is a specialization of the desired value *artificial.intelligence*."
Status: complete
Reasoning.method: specialization.constraint.values
Hierarchy.level.difference: 1

Unit-name: GENERALIZATION.CONSTRAINT.VALUES.EXPL14
Unit-comment: "Show that the actual value of the violated constraint predicate is a generalization of the target value in the constraint."
Explanation-summary: "If the value of the *area.of.expertise* field of *Seymour.Wright* (*computer.vision*) had been a generalization of the target constraint value *artificial.intelligence*, this explanation would be valid."
Status: incomplete
Reasoning.method: generalization.constraint.values

Figure 3: Explanations for *Dynamic.object.constraint.violation13*

(*computer.vision* is a subclass of *artificial.intelligence*) exists in the knowledge base, *generalization.constraint.values.expl14* is not only incomplete but *invalid*, since an object cannot be both a generalization and a specialization of the same object. To illustrate the handling of incomplete explanations, suppose, however, that the taxonomic link between *computer.vision* and *artificial intelligence* was not initially specified in the knowledge base. If this had been the case, the two explanations in Figure 3 would have been constructed as *incomplete* (but potentially valid) explanations. Either one of them may have been chosen during the negotiation process and completed by verifying with a knowledgeable agent that the appropriate missing taxonomic link could indeed be added to the knowledge base.

4 Amendments

Once the candidate explanations for an exception have been generated by the exception analyzer, a selection must be made. Currently, summaries of the candidate explanations are

presented to the user, who makes a choice. Once the selection of an explanation has been made, it must be *implemented* to restore system consistency and enable the resumption of execution. The implementation of an explanation is specified by one or more *amendments* to be made to the static or dynamic state of the system. At this time, the choice of the amendment is also made by the user, although this choice could be automated by weighing the implementation costs of the candidate amendments.

Each plausible inference rule has one or more amendment types associated with it, based on the conditions involved in its premise. Amendments are generated only for complete explanations, unless an incomplete explanation is chosen and verified through agent interaction. In this paper, we will describe the concept of amendments by discussing the subset of exceptions which generate constraint violations, and illustrate with the amendments (see Figure 4) generated for the complete explanation *specialization.constraint.values.expl15*.

In general, when encountering a constraint violation, there are three fundamental approaches to resolving the problem[3]:

1. Relax the constraint. Possible ways to do this are:
   (a) Disjunct addition (add a disjunct to the specification);
   (b) Conjunct elimination (eliminate one or more conjuncts from the specification);
   (c) Taxonomic generalization (replace a class specification with a more general superclass);
   (d) Taxonomic expansion (replace a class specification by a set of classes which are subsumed by the original class specification);
   (e) Range extension (extend a numeric or other ordinal range);
   (f) Constraint elimination (eliminate the constraint);
2. Remove the source of constraint violation.

[3]The first approach is largely based on work described in [1].

Unit-name: CONSTRAINT.VALUE.DISJUNCT.ADDITION16
Unit-comment: "Modify a current.constraint by adding an additional.value
in a disjunct.clause to produce a new.constraint."
Amendment-summary: "Replace the current constraint
(area.of.expertise Seymour.Wright artificial.intelligence) with the new constraint
(or (area.of.expertise Seymour.Wright artificial.intelligence)
(area.of.expertise Seymour.Wright computer.vision))."
Implementation: (add-values-to-constraint (computer.vision)
(area.of.expertise Seymour.Wright artificial.intelligence))

Unit-name: CONSTRAINT.VALUE.TAXONOMIC.EXPANSION17
Unit-comment: "Modify a current.constraint by replacing the existing.value with
its taxonomic.expansion to produce a new.constraint."
Amendment-summary: "Replace the current constraint
(area.of.expertise Seymour.Wright artificial.intelligence) with the new constraint
(area.of.expertise Seymour.Wright
(or artificial.intelligence distributed.ai planning robotics computer.vision))."
Implementation: (replace-values-in-constraint
(artificial.intelligence distributed.ai planning robotics computer.vision)
(area.of.expertise Seymour.Wright artificial.intelligence))

Unit-name: TARGET.OBJECT.ATTRIBUTE.VALUE.ADDITION18
Unit-comment: "Modify a KB object by adding a new.value to
the existing.values in an object.attribute field of the object."
Amendment-summary: "Add the new value *artificial.intelligence* to the current values
(*computer.vision*) of the *area.of.expertise* field of the KB object *Seymour.Wright*."
Implementation: (add-attribute-values-to-kb-object (artificial.intelligence)
area.of.expertise Seymour.Wright)

Figure 4: Amendments for *Specialization.constraint.values.expl15*

(a) The constraint may have failed because of unknown information. Perhaps the missing knowledge can be acquired, resulting in a successful evaluation of the constraint.

(b) Information in the knowledge base may be incorrect. A change should be made to the knowledge base so that the subsequent evaluation of the constraint is successful.

3. Undo what was done, and do it differently (replan).

Conceptually, the first approach implies that a constraint was specified incorrectly; it was too strict. The second appoach implies that some knowledge is either not explicit or is incorrect in the knowledge base, and that additional information can either be inferred or acquired in order to nullify the violation. With these concepts in mind, the amendments shown in Figure 4 were produced for the example exception summarized in Figure 2. Note that the first two proposed amendments illustrate the first approach. They use the techniques of disjunct addition and taxonomic expansion to actually change the constraint specification to accommodate the exception. The third amendment is based on the second approach, and involves making an actual change to a knowledge base domain object. Thus, the actual source of the constraint violation is removed. The user selects one of these amendments to indicate how the explanation should be implemented. The knowledge base modifier performs the indicated changes specified in the *implementation* field of the amendment, and planning and execution by POLYMER is resumed.

5 Status

A prototype system implementing the SPANDEX architecture is integrated with the POLYMER planner and running on a Texas Instruments Explorer. The exception analyzer currently handles a subset of the exception types; algorithms for the remainder are being implemented. Generalization techniques such as those described in [1] are being examined as the basis for the explanation generalizer. The negotiator is not yet implemented.

6 Appendix

As an additional illustration of the mechanisms discussed in this paper, we present in this section an example from a second domain, that of software development. The overall goal of the example task is to create a new version of a software system, incorporating desired changes and additions. The primary agent initiating the task is the project leader, who is directing a programmer, *Dave.Hildum*, to effect the changes. POLYMER generates a partial plan network for this task, and executes it in conjunction with the relevant agents. There are three ordered subgoals 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) and *have-consistent-system* (the entire software system must be updated so that changed modules are recompiled and the system is relinked).

The agent (Dave.Hildum) is instructed to perform the actions *think* and *edit* which are selected by POLYMER to achieve the first two subgoals. The actions are performed as anticipated. The planner attempts to achieve the third subgoal *have-consistent-system* by selecting the activity *update-software-system* to achieve it. 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 is invoked by POLYMER upon detection of this exception, and the exception classifier determines that an action mismatch has occurred, implying a possible attempt at an action substitution or an out-of-order action[4]. An *exception record* (see Figure 5 below) is created to summarize the exception, recording the action which generated the exception, and the type of exception which was detected. SPANDEX then invokes the exception analyst, which uses a heuristic *strategy.selector* to choose a strategy to generate an *explanation* for the exception.

In this example, a single applicable plausible inference rule is retrieved, and SPANDEX

[4]These implications are derived from relevant plausible inference rules, as described in section 3.1.

Unit-name: ACTION.MISMATCH.01
Unit-comment: "The action taken by an agent did not match the action expected by the planner. "
Exceptional-action: unix-make-1
Exception-summary:
"The target action: *compile-file-01* did not occur,
it is the case that *unix-make-01* was performed."
Target.action:*compile-file-01*
Perceived.action *unix-make-01*
Strategies: action.substitution.strategy, out.of.order.action.strategy
Strategy.selector: strategy.selector.method
Explanations: substitute.for.higher.level.goal.01

Figure 5: ACTION.MISMATCH.01

constructs one plausible explanation, which is complete. (see Figure 6). In this particular case, the complete explanation states that since the goal of the unexpected action *(updated(*SPANDEX*))* unifies with the goal of a parent[5] of the expected action node (the goal of *update-software-system*, which is the parent node of the expected *compile* action, is also *updated(*SPANDEX*)*) the unexpected action may be a substitution for the more abstract parent node. Since there is only a single explanation in this example, there is no need for negotiation among affected agents to choose among potential explanations.

An amendment 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 explanation 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 effect, the nodes in the expansion of *update-software-system-01* are deactivated from the planner's predictions (see Figure 7).

The changes specified by the implementation field of the amendment shown are per-

[5]The term "parent" here is used to refer to the more abstract node from which an expansion now in place in the current network was derived.

Unit-name: SUBSTITUTE.FOR.HIGHER.LEVEL.GOAL.01
Unit-comment: "Show that the unexpected action is a substitute
for an in.progress.parent.node of the expected action."
Explanation-summary: "The unexpected action *unix-make-01* is sufficient since it
has a goal unifying with the goal of the parent node *update-software-system-01*. "
Status: complete
Reasoning.method: substitute.for.higher.level.goal
In.progress.parent.activity: update-software-system-01
Pending.goal.achieved: updated(SPANDEX)
Unexpected.action.goal: updated(SPANDEX)
Hierarchy.level.difference: 2

Figure 6: Explanation for ACTION.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 7: Amendment for SUBSTITUTE.FOR.HIGHER.LEVEL.GOAL.01

formed, and planning and execution by POLYMER is resumed.

References

[1] Borgida, A., and K.E. Williamson, " Accommodating Exceptions in Databases, and Refining the Schema by Learning from Them," *Proceedings of the Very Large Data Base Conference*, 1985, pp. 72-81.

[2] Broverman, C.A., Croft, W.B. "Exception Handling During Plan Execution Monitoring," *Proceedings of the Sixth National Conference on Artificial Intelligence*, July 1986, Seattle, WA., pp.190-195.

[3] Broverman, C.A., Croft, W.B. "SPANDEX: An Approach to Exception Handling in an Interactive Planning System," COINS Technical Report No. 87-127, University of Massachusetts, Amherst, Mass. December 1987.

[4] Croft, W.B., Lefkowitz, L.S. "Task Support in an Office System," *ACM Transactions on Office Information Systems*, 2: 197-212; 1984.

[5] Croft, W.B., Lefkowitz, L.S. "Knowledge-Based Support of Cooperative Activities," *Proceedings of the Hawaii International Conference on System Sciences*, January 1988, pp. 312-318.

[6] Croft, W.B., Lefkowitz, L.S. "A Goal-Based Representation of Office Work," IFIP Conference on Office Knowledge. North Holland, W. Lamersdorf, ed., 1988.

[7] DeJong, G.F. "Generalizations Based on Explanations," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, B.C., Canada, August 1981, pp. 67-70.

[8] DeJong, G., Mooney, R. "Explanation-based Learning: An Alternative View," *Machine Learning*, 1, 1986.

[9] DeJong, G. "An Approach to Learning From Observation," Machine Learning, Vol. II. Michalski, Carbonell, Mitchell, eds., Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986, pp. 571-590.

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

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

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

[13] Sathi, A., Morton, T.E., Roth, S.F. "Callisto: An Intelligent Project Management System," *AI Magazine*, 7:5, Winter, 1986, pp. 34-52.

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

[15] Wilkins, D.E. "Recovering from Execution Errors in SIPE", SRI International Technical Report 346, 1985.