

**Constructive Interpretation of Human-Generated
Exceptions During Plan Execution**

Carol A. Broverman
Ph.D. Thesis

COINS Technical Report 91-9
February 1991

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

© Copyright by Carol A. Broverman, 1990
All Rights Reserved

IN LOVING MEMORY OF MY MOTHER,

RITA WOLFE BROVERMAN.

ACKNOWLEDGMENTS

I owe much to my advisor, Bruce Croft, without whom I would never have made it through this process. He was always available to discuss ideas, and to provide moral and intellectual support whenever I needed it most. I would also like to thank Vic Lesser for his input, and particularly for reminding me of the importance of keeping the ink flowing steadily (or in today's vocabulary, the fingers typing...). I am also indebted to the sharp and broadening perspective that Edwina Rissland provided in her contributions to the final product. Finally, I would like to thank Graham Gal for his friendly support and a helpful outside perspective.

I would like to express my appreciation and thanks to all my COINS friends who have helped keep me from losing sight of the end of the tunnel and for reassuring me that manic depression was a normal state during the dissertation years. I owe special thanks to Al Hough (the original spokesman) and Larry Lefkowitz (the constant reiterator) for reminding me that "Coding (no matter how much) can never be a substitute for thinking!" — which encouraged me not to stay in the tunnel. I would especially like to thank Larry for all his support, his argumentative spirit (believe it or not), and most of all his friendship during these seven years. Finally, I owe much to my non-COINS friends, and especially my family, for their constant and unflagging love and support.

ABSTRACT

CONSTRUCTIVE INTERPRETATION OF HUMAN-GENERATED EXCEPTIONS
DURING PLAN EXECUTION

FEBRUARY 1991

CAROL A. BROVERMAN, B.S. PSYCHOLOGY, TUFTS UNIVERSITY

M.S., COMPUTER SCIENCE, UNIVERSITY OF MASSACHUSETTS

PH.D., COMPUTER SCIENCE, UNIVERSITY OF MASSACHUSETTS

Directed by: Professor W. Bruce Croft

In realistic settings, several factors limit the utility of planners based on the classical nonlinear hierarchical paradigm. The need for human supervision, the inherent incompleteness and incorrectness of models of complex domains, the dynamic nature of the real world, and the frequency of unexpected occurrences while carrying out a calculated plan all imply the inevitability of run-time discrepancies. Previous approaches have viewed deviations from expectations as destructive events, devoid of any valid semantic basis. In contrast, this thesis promotes a constructive approach. We adopt the assumption that the behavior of human agents who interact with the planning system is purposeful, even when it may be inconsistent with system expectations. We claim that the actions of human agents are often misinterpreted as failures, due to gaps in the domain model.

This thesis focuses on an interactive planning framework, and describes

an approach that intelligently resolves detected inconsistencies resulting from the actions of human agents. Through a process of controlled explanation, unusual occurrences are incorporated as valid contributory events, thus avoiding extensive replanning. Explanations are based on a theory of *rationales*, encoded as plausible inference rules and operationalized for use as a problem detection and correction mechanism. Initial points of failure are categorized according to a taxonomy of *exceptions*, and the relevance of an exception to the current plan context and domain model is established. *Amendments* are then made to the partial plan and/or the domain model to restore consistency.

A robust planning architecture results that resolves unanticipated contingencies in a constructive fashion whenever possible. In addition, exception handling is used as an opportunistic entry point through which additional knowledge acquisition and refinement takes place. New knowledge acquired during the handling of exceptions produces an augmented knowledge base and improved subsequent system performance. A Common-Lisp prototype demonstrating the approach described in this thesis is implemented on a TI-Explorer. User studies support the adequacy and coverage of our exception taxonomy and rationales.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	v
ABSTRACT	vi
LIST OF FIGURES	xiii
CHAPTER	
1. OVERVIEW	1
1.1 Planning in a Cooperative Setting	2
1.1.1 Sources of Knowledge	3
1.1.2 Managing Complex Tasks Involving Human Agents	6
1.2 Exceptions	10
1.3 The Thesis	15
1.4 System Sketch	19
1.5 Example	22
1.6 Proof of Concept	27
1.7 Guide to the Thesis	28
2. RELATED RESEARCH	30
2.1 Planning and Nonlinear Hierarchical Planners	31
2.2 Plan Flaws and Replanning	35
2.3 The Adaptation and Transformation of Old Plans to Fit New Situations	48
2.4 Explanation-Based Learning Applied to Novel Phenomena	54
2.5 The Completion of Incomplete Theories or Explanations: Learning	57
3. PERSPECTIVES ON DEVIATIONS BY HUMAN AGENTS DURING PROCEDURAL TASKS	59
3.1 Errors as Exceptions	59
3.2 Related Work on Human Error	61
3.3 A New Theory of Human Procedural Deviations	76
4. THE SPANDEX APPROACH	79

4.1	Planning and Execution	79
4.2	Exceptions	87
	4.2.1 Detection of Exceptions at Run Time	88
	4.2.2 The Exception Problem	89
4.3	Explanation of Exceptions	92
4.4	A Review of the System Architecture	96
4.5	An Explanation-Based Learning Perspective	101
5.	AN INTERACTIVE PLANNER	105
5.1	POLYMER Planning Details	106
	5.1.1 Agents	111
	5.1.2 Composite versus Primitive Activities	113
	5.1.3 Objects	114
	5.1.4 Constraints	114
5.2	Modifications to POLYMER Representation to Handle Ex- ceptional Occurrences	116
	5.2.1 Offline Plan Networks	116
	5.2.2 User-assertion Activity	117
5.3	Activity Selection	118
5.4	Suggestions	123
	5.4.1 Constraints: Used as Parsers	124
	5.4.2 Constraints: Used as Generators	125
6.	THE SPANDEX IMPLEMENTATION	137
6.1	Exception Detection	137
	6.1.1 Exceptions During User Interaction	138
	6.1.2 Exceptions Resulting from Planner Operations	141
	6.1.3 Plan Flaws Revisited and Comparison With Other Systems	142
	6.1.4 Extensions	146
6.2	Exception Classification	147
	6.2.1 The Role of the Exception Classifier	148
	6.2.2 Context and Terminology	150
	6.2.3 Classifications	154
6.3	Explanation Construction	163
	6.3.1 Explanation Basis Types	164
	6.3.1.1 Rationales	164
	6.3.1.2 Imperfect Domain, the "Else"	166

6.3.2	Determining the Scope of an Exception	166
6.3.3	Establishing the Basis for Explanation	168
6.3.4	Strategy Establishment and the ISS Record	175
6.4	Amendments	179
6.4.1	Amendments for Unanticipated Actions	181
6.4.2	Explicit Knowledge Acquisition for no-activity-possible Exception	184
6.4.3	Explicit Knowledge Acquisition for no-known-resource	184
6.4.4	Discussion of Amendments that Involve Constraints	185
6.4.5	Inconsistent Resource Choices	187
6.4.5.1	Adding to the Knowledge Base in Response to Unsatisfied Constraints	189
6.4.5.2	Adding to the Knowledge Base in Response to Violated Constraints	201
6.4.5.3	Another Approach: Modifying the Constraints to Fit the KB	202
6.5	Explanation Presentation	203
6.6	Limitations of the Implementation	204
7.	DEMONSTRATION AND EVALUATION	212
7.1	SPANDEX Scenarios	214
7.1.1	Scenario 1	214
7.1.2	Scenario 2	232
7.1.3	Scenario 3	242
7.1.4	Scenario 4	248
7.1.5	Scenario 5	255
7.1.6	Scenario 6	259
7.2	Cost of explanation generation	266
7.3	Analysis of Survey Results	269
8.	SUMMARY AND FUTURE DIRECTIONS	278
8.1	Restatement of the Problem	278
8.2	Summary of the thesis	280
8.3	Contributions	282
8.3.1	Interactive Planning as a Total Man-machine System	282
8.3.2	Expanded Plan Flaw Taxonomy, and a Constructive Approach Towards Their Resolution	283

8.3.3 Graceful Degradation, and the Avoidance of Re- planning	284
8.3.4 Constraint Relaxation	285
8.3.5 Extending the Domain Model for Improved Per- formance	285
8.4 Directions for Future Research	286
8.5 Conclusion	291
APPENDICES	
A. POLYMER DOMAIN THEORY	293
A.1 POLYMER Domain Theory	293
A.2 Plan Network Definitions	295
B. ALGORITHMS FOR SUGGESTIONS COMPUTATION	297
B.1 The algorithm for calculation of suggestions for INTER- SECTION valueclass constraint	297
B.2 Calculating the Intersection of Two Value Sets	297
C. SPANDEX STRUCTURES	300
D. EXAMPLE CONSTRAINT PARTITIONS	308
E. CONFERENCE DOMAIN DESCRIPTION	315
F. SURVEY: EXPERIENCE IN ORGANIZING AND ATTENDING CON- FERENCES	344
BIBLIOGRAPHY	351

LIST OF FIGURES

1. The process-call-for-papers activity	4
2. The call-for-papers object	4
3. A cooperative planner	8
4. Knowledge distribution in cooperative planning	17
5. Architecture for a cooperative planning system	20
6. Partial plan outline for issuing a call for papers	23
7. Asking the user to perform a task	24
8. The SPANDEX execution record	25
9. Chosen explanation	26
10. Partial plan outline after exception is resolved	27
11. Wedges of a plan outline	84
12. An interactive hierarchical planning loop	91
13. A plan outline as a pyramid of wedges	96
14. An architecture for planning and exception handling	98
15. An interactive hierarchical planning loop (extended)	100
16. POLYMER plan network before expansion	106
17. The stage.a.symposium activity	108
18. POLYMER plan network after expansion	112

19. An example domain object description	115
20. A partial graph of entities in the CARPENTER domain	121
21. The exception types detected and handled by SPANDEX	138
22. A comparison of plan flaw taxonomies	145
23. Example of establishing bindings during goal matching	155
24. Terminology used during exception classification	156
25. An example match record stored in an explanation	162
26. An example plausible inference rule (PI)	164
27. The explanation.basis types used by SPANDEX	165
28. SPANDEX explanation establishment strategies	171
29. Inconsistent-state-specs for the accomplish-current-step explanation strategy	177
30. SPANDEX amendment types	180
31. The form of a simple constraint	190
32. The constraint partition structure	192
33. The constraint partitioning algorithm	193
34. A partial view of the SITES Knowledge base	194
35. The execution.record after handling example in Example 1	195
36. The selected constraint partition for example 1	196
37. The execution.record after handling exception in Example 2	198
38. The selected constraint partition for Example 2	199
39. Possible activities to achieve (a papers-chosen-for-presentation (?meeting, done))	217

40. Initial goal posed to planner in Scenario 1	218
41. Choosing submission.004 from among possible submissions	219
42. The decision slot of the submission and submission.004 units prior to constraint relaxation	219
43. Invoking the exception handler, with the current inconsistent plan network	220
44. Strategies and indicators for the Imperfect-domain explanation basis	222
45. The decision slot of the submission and submission.004 units after valueclass modification	224
46. Selected explanation to generalize constraint by adding a disjunct	224
47. The possible-decisions in the knowledge base and the decision slots of the submission and submission.004 units prior to valueclass modi- fication	225
48. The decision slot of the submission and submission.004 units after valueclass modification	226
49. Selected explanation to generalize constraint by replacing with a more general class object	227
50. The decision slot of the submission unit and possible-decisions prior to valueclass modification	228
51. The decision slot of the submission unit and possible-decisions after knowledge base has been extended to satisfy violated constraint .	229
52. Selected explanation to modify the knowledge base in order to sat- isfy a violated constraint	230
53. Contents of the SPANDEX execution.record after handling an excep- tion during planning	231
54. User selects from possible SPANDEX explanations to resolve the ex- ception	231

55. User selects from possible SPANDEX amendments to resolve the exception	232
56. Partial plan outline for issuing a call for papers	233
57. The POLYMER plan network for issuing a call for papers	234
58. Asking the user to perform a task	234
59. The user chooses from among alternative activities	235
60. The newly acquired activity CONTRACT-OUT-CALL-FOR-PAPERS	236
61. The SPANDEX execution record	237
62. Chosen explanation	238
63. Partial plan outline after exception is resolved	239
64. Partial view of POLYMER plan network after exception is resolved	240
65. Plan network before exception, showing interaction with user	243
66. Calling the exception handler	243
67. The plan network after wedge subsumed by symposium-is-planned is excised	244
68. The selected explanation record for exception in Scenario 2	244
69. Conditions on the selection of a reviewer, specified in executable node	246
70. The paper to review and available reviewers	246
71. Choosing among qualified reviewers	247
72. Specifying a new reviewer not in suggestions	247
73. The SPANDEX execution.record	248
74. The chosen explanation record	249

55. User selects from possible SPANDEX amendments to resolve the exception	232
56. Partial plan outline for issuing a call for papers	233
57. The POLYMER plan network for issuing a call for papers	234
58. Asking the user to perform a task	234
59. The user chooses from among alternative activities	235
60. The newly acquired activity CONTRACT-OUT-CALL-FOR-PAPERS	236
61. The SPANDEX execution record	237
62. Chosen explanation	238
63. Partial plan outline after exception is resolved	239
64. Partial view of POLYMER plan network after exception is resolved	240
65. Plan network before exception, showing interaction with user	243
66. Calling the exception handler	243
67. The plan network after wedge subsumed by symposium-is-planned is excised	244
68. The selected explanation record for exception in Scenario 2	244
69. Conditions on the selection of a reviewer, specified in executable node	246
70. The paper to review and available reviewers	246
71. Choosing among qualified reviewers	247
72. Specifying a new reviewer not in suggestions	247
73. The SPANDEX execution record	248
74. The chosen explanation record	249

75. Choosing among new facts to add to restore consistency	249
76. Scenario4: network before exception	250
77. Scenario4: network after exception	251
78. Scenario4: chosen explanation	252
79. Scenario4: the execution record	253
80. Scenario4: the network after removing intervening steps	254
81. Scenario5: the network before exception	257
82. Scenario5: network after exception	258
83. Scenario5: chosen explanation	258
84. Scenario5: the execution record	260
85. Scenario5: the network after regressing entire activity and expanding	261
86. Activity specialization	262
87. Scenario6: A complete explanation illustrating activity specialization	264
88. Scenario6: The SPANDEX execution record	266
89. Breakdown of identified exceptions by rationale	270
90. Breakdown of identified constraint relaxations	273
91. The work breakdown structure for staging a conference	350

CHAPTER 1

OVERVIEW

There is a trend in recent planning research to move the focus away from the controlled "blocks world" and address the requirements of more realistic settings. In many real-world situations, human intervention and supervision is often necessary, and unpredicted events can occur in the context of a pre-calculated long-term plan of action based on an incomplete domain theory. In applications such as *C³I* (command, control and communications intelligence) or other tactical situations, a dynamic change in the physical environment may alter plans, and adversarial actions must also be anticipated and counter-acted. Alternatively, in laying out a plan in an application such as intelligent tutoring, unexpected actions by the student should be taken into account and incorporated into the overall plan for teaching the task at hand. In fact, in any situation where unexpected events or actions may take place, a computer-based system must be able to deal gracefully with such occurrences, taking care to preserve as much of the original plan as possible.

The methods embodied by the classical nonlinear hierarchical planning paradigm can be of practical use in loosely constrained settings such as those just described, although making the required extensions to the very limited representation used by provably complete and correct planners results in the

loss of these theoretical properties [13]. The SIPE planner is one attempt to modify the classical planning paradigm to make it more suitable for solving practical problems [72, 73, 74]. This thesis takes yet another step in this direction, focusing on the role of human agents in cooperative plan development and execution.

We are concerned with using planning techniques to support the work of human agents in the performance of practical complex tasks. In particular, this thesis provides an approach towards the resolution of exceptional behavior during interactive plan generation and execution that will inevitably arise in realistic settings involving human agents. In this chapter, we first describe characteristics of the planning problem in a cooperative setting (Section 1.1). Section 1.2 then describes how exceptional events can occur within this interactive framework, and the primary aims of this thesis on exception handling are described in Section 1.3. Section 1.4 outlines the architecture that we have proposed and implemented to address the problem of handling exceptions within the framework we have identified, illustrating the capabilities of the system with an example from the domain of conference planning in Section 1.5. Finally, we briefly describe the implementation in Section 1.6, and a guide to the remainder of the thesis is found in Section 1.7.

1.1 Planning in a Cooperative Setting

Tasks that are performed by people in contemporary work environments are complex and detailed, usually involving multiple interrelated steps as well as communication and cooperation with other agents. For example, an individual

planning an academic conference is responsible for arranging the logistics of where the conference is to be held, where to house attendees, and who will be on the program committee, as well as for the delegation of subtasks to other agents, such as the refereeing of submissions and the catering of meals and coffee breaks.

1.1.1 Sources of Knowledge

Several sources of information must be available in order to successfully plan and execute a complex task. These are described below.

Taxonomic knowledge of domain objects and activities

The activities pertinent to the domain must be known in terms of the goals they are able to achieve, the effects that result from their performance, and conditions on their execution. For example, the program chair who is planning a conference must know what is involved in issuing a call for papers, such as establishing the date and requirements of the submission, and getting the call typeset, printed and distributed (see Figure 1). The domain objects that are manipulated by the activities of the domain should also be known, along with any interrelationships (an example object description is shown in Figure 2). Together, this knowledge constitutes the basic building blocks that determine how conferences are to be run.

```

{ACTIVITY process.call.for.papers
  *agents: ?agent
  *goal: status(?call, issued)
  *input-goal-vars: ?call
  *decomposition: call-for-papers-typeset = status(?call, typeset)
                  call-for-papers-printed = status(?call, printed)
                  call-for-papers-distributed = status(?call,
                                                         distributed)

  *constraints: ?call is of type call.for.papers
                ?agent is of type program-chair
  *control: (BEFORE call-for-papers-typeset call-for-papers-printed)
            (BEFORE call-for-papers-printed
              call-for-papers-distributed)
}

```

Figure 1: The process-call-for-papers activity

```

{OBJECT call.for.papers
  *subclass.of: written.material
  *meeting: VALUE-CLASS: technical.meeting
  *start.date: VALUE-CLASS: date
  *finish.date: VALUE-CLASS: date
  *location:
  *sponsors: VALUE-CLASS: (one.of professional.organization)
  *key.organizers:
  *meeting.scope:
  *meeting.purpose:
  *types.of.papers.desired:
  *due.date: VALUE-CLASS: date
  *acceptance.notification.date: VALUE-CLASS: date
  *where.to.send:
  *status: VALUE-CLASS: (one.of created typeset printed
                        distributed issued)
}

```

Figure 2: The call-for-papers object

Constraints

Many tasks are restricted in that they can only be performed in particular settings, are permitted to manipulate a restricted set of resources, or the substeps that make up a task must be accomplished in a particular order. For example, the control section of Figure 1 states that the call for papers must be typeset and then printed before the document can be distributed. Other examples of constraints in the conference domain are: a registration form cannot be sent out until it has been completed, participants cannot be issued name badges until they have registered, and an individual cannot be selected to be on the program committee for a conference if they are a student. Time limits also frequently apply with regard to the completion of task milestones.

In addition, constraints apply to the domain objects that are manipulated by domain activities. For example, Figure 2 illustrates that the status of a call for papers can be created, typeset, printed, distributed or issued, and the sponsor associated with a call for papers must be one of the known professional organizations.

Agent capabilities

A planner must have knowledge of the capabilities of other agents in order to delegate subtasks. Note that according to Figure 1, the agent responsible for issuing the call for papers should be the program chair of the technical meeting being planned. When the planner is working on the part of the plan involving issuing a call for papers this agent specification will be used to elicit the assistance of the appropriate individual(s).

Contingency tactics

In dynamic domains, it is important to have knowledge of how to overcome difficulties that might arise during the planning and execution of the task. For example, if the conference headquarters hotel is damaged by fire a month before the conference is to be held, arrangements must be made for an alternative site and everyone involved must be notified of the change. Alternatively, if the program committee decides to change the format of the original technical program schedule, arrangements must be made to eliminate or move the corresponding coffee breaks, fill in slack time or eliminate sessions if necessary, or reschedule rooms. In addition, the system may need to contend with a potential situation in which the program chair chooses an alternative route to issuing the call for papers, perhaps by contracting out the activity to an in-house service provided by the sponsor.

1.1.2 Managing Complex Tasks Involving Human Agents

The intricate makeup of tasks in most organizations implies a distinct need for task management facilities. The proliferation of project management tools [48] attests to the need for the planning and monitoring of complex tasks. Unfortunately, current project management techniques do not represent task components at the level of detail which is necessary to intelligently monitor task performance, and provide little help with plan generation. Usually, human agents must construct the entire task by hand, with the tool providing support for specialized aspects of the task, such as the detection of time-dependent conflicts and critical path computation.

An obvious next step is to adopt a more global view of complex tasks and provide more support for the task plan generation process. This view should incorporate a model for the coordination of the various special purpose tools that currently provide partial support for human activities in complex organizations, such as word processors, software compilers, and electronic mail facilities. An advance in this direction depends on an explicit and rich representation of task knowledge that is amenable to computation. In this regard, efforts have been made to apply artificial intelligence planning techniques to assist in the effective planning and completion of complex tasks. Computer-based planning techniques can be used to construct a detailed step-by-step procedure representing the typical way to achieve a particular task goal, automating much of the tool usage and coordination. The level of complexity to be managed by the human is thus significantly reduced. Specifically, the techniques of (1) plan refinement, (2) subgoal interaction detection and correction, and (3) the control of search through the management of constraints that are provided by the classical nonlinear hierarchical planning paradigm [55, 63, 71] can be brought to bear on the problem of intelligent task management. Resulting systems have been described as *intelligent assistants* for human agents in an organizational setting [11, 17, 33].

The application of "NOAH-style" [55, 63, 73] planning techniques has generally been restricted to settings in which the domain is relatively narrow and specialized, and the agents who execute the plan are neither rational¹ nor autonomous. For example, in robot planning, a robot who is executing a planned task is a "dumb executor" who follows commands and is incapable of making

¹*Rational* here is intended to mean "capable of reasoning and making intelligent decisions."

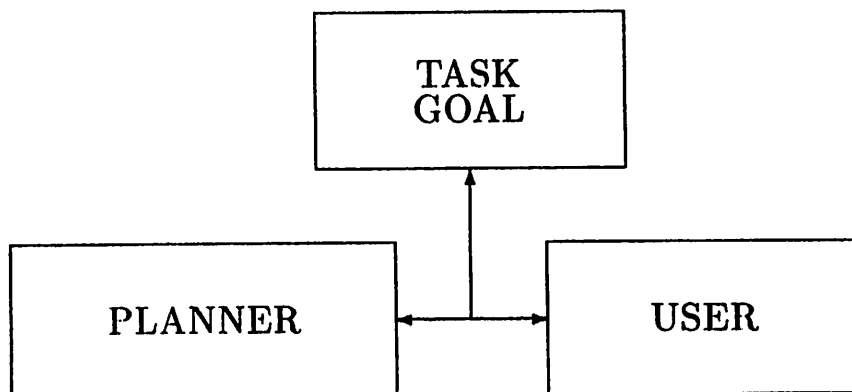


Figure 3: A cooperative planner

independent decisions that contradict the directives of the planner. In contrast, one of the most distinctive characteristics of the application of a planning paradigm to many real-world settings is the significant involvement of human agents who *do* exhibit rational and autonomous behavior. A problem-solving environment involving human agents thus has some important features which must be considered in the design of systems for use as intelligent assistants:

- **Human agents initiate planning.** The planner is invoked by human agents to assist in task management. The planner should be regarded as under the control of the human agent.
- **Planning is interactive.** The planner is not a stand-alone system; tasks often cannot be fully automated. In general, the human agents and planner cooperate in an attempt to achieve a common goal, namely the achievement of the goal of the task (see Figure 3). The user incrementally supplies salient information in the form of control decisions or parameter values to guide the planning process. This interaction im-

poses constraints on the ongoing development of the partial plan for a task goal.

- **Planning and execution are interleaved.** A common mode of control in a hierarchical planning system is typified by the *expand-criticize* loop used by NOAH [55]. In that and other similar planners, a complete level of the plan is expanded and criticized at a time, and all execution takes place only after the plan is completely expanded. This simple breadth-first approach is not appropriate for all environments. In some environments, the planner must wait for the user to execute an information-providing action before it can continue planning. For example, hotel selection for a conference cannot be made until it is known how large the conference is expected to be. Primitive actions by the user often constitute the execution of steps in the plan. In addition, the execution of actions by the planner may provide information necessary in order to predict what the user should do next towards accomplishing the task. Therefore, a depth-first processing strategy that involves interleaving execution with planning actions is more appropriate for a cooperative planner. This planning cycle is better characterized as a *pick-(execute or expand)* cycle [43].
- **The domain is underspecified.** A typical domain in an organizational setting does not have easily defined boundaries and can be difficult to specify. The successful construction and execution of a plan in such an environment requires the availability of common knowledge about task management, resources and contingency planning, as well as specific do-

main knowledge, and it is impossible to guarantee that an initial domain specification will be either correct or complete.

- **The setting is dynamic.** It is hard to prespecify all possible ways that task goals may successfully be achieved, as well as to anticipate the unexpected occurrences during execution that may alter characteristics of the initial setting upon which the plan was based. In addition, the general policies in the domain of concern that are in effect during the initial construction of the task descriptions may change over time.

1.2 Exceptions

The characteristics of a cooperative plan setting just described result in inherent limitations of an interactive planning system with respect to the real world. No matter how carefully a plan for a task may be conceived, something frequently may go wrong during its execution, or an unexpected contingency can cause the plan to go awry. We give a general definition of the term *exception* to refer to any detectable event or state that contradicts expectations of the current plan. In Section 4.2 we give a more concise definition of what we mean by an *exception* and in Section 6.1 we present a detailed taxonomy of exception types that are detected in our interactive planning framework.

Causes of exceptional behavior

In addition to the dynamic nature and inherently incomplete description of the domain, the algorithms guiding the planning process often opt for ef-

iciency over completeness. This trade-off can be explained as follows. In general, a planner employs a set of axioms that define the planning process and a predefined set of domain activities and objects to generate valid plans to accomplish a particular goal. The algorithms guiding plan expansion generally do not use all available knowledge, which if analyzed might provide definitions of a number of alternative methods for accomplishing a given task. The combinatorics of such a computation at every goal node expansion would be prohibitive. So, in order for a planner to operate efficiently and generally in a knowledge-rich domain, it may perform actions and decisions using only part of the potentially large amounts of relevant domain knowledge. While restricting a potentially explosive search space, the plans that are produced are stereotypical approximations and may not be adequate predictors of subsequent execution behavior.

Another source of inadequate planner predictive power emerges as a paradoxical consequence of the influx of technology used to automate tasks previously performed by humans. The introduction of greater amounts of automation does not necessarily avoid the problems of human error; rather, the source of inaccuracy is simply regressed back from the actions of the human operator to the decisions or knowledge encoded by the human designer. In other words, in an attempt to bypass the human element and avoid human operator error, systems are designed that are internally incorrect and that can generate inaccurate behavior. Furthermore, it has been found that human operators must still be involved in accomplishing the parts of the task that the designer does not know how to automate [6]. Exceptional behavior thus remains as a significant factor that must be contended with.

There are several other possible reasons that inconsistencies can arise during plan execution. For example, human planners do not always plan with a large degree of look-ahead. People change their minds, or opportunistically revise the plan mid-execution. They also opt for short-cuts while performing a task, or vary their usual methods. In addition, human beings are prone to error or misjudgement. Consequently, the class of exceptional occurrences that can arise when human agents are involved in plan execution is diverse and exceptions can be commonplace.

Anticipating exceptions

Therefore, the planning and execution of a complex task must be monitored for success. It is important to recognize changes to the assumed plan context that can affect the ultimate success of the plan. As described in a previous example, if the conference headquarters hotel is damaged by a fire a week before the start of the conference, the conference organizer will have to make new arrangements for the housing of the attendees. Alternatively, the conference organizer or one of the other domain agents may take deliberate actions that are not consistent with the system's view of the current task. For example, the program committee head may phone a contributor with an acceptance decision instead of mailing it out as expected. Techniques are required to efficiently handle such occurrences.

Dimensions of plan inconsistencies

Cognitive and empirical studies have suggested different ways to categorize inconsistencies that occur during the execution of a procedure by a human agent [32, 52, 53, 49, 40]. Upon further analysis, some of these apparently erroneous actions can be recognized as purposeful actions that can be regarded as consistent with the goals of the plan. For example, when the program chair made a phone call instead of using U.S. mail to notify an author of an acceptance decision, it may be that this agent has chosen an alternative route for communicating required information. In general, an unanticipated agent action is often relevant to the ongoing plan, and should be interpreted accordingly.

This discussion indicates that it is important to identify the impact that the unanticipated action has on the current plan. A distinction can be made between exceptions that represent *failures* and those that represent *surprises* [44]. A *failure* corresponds to the most typical type of planning exception in which new information introduces problems in the plan, preventing its success. A *surprise* is a more positive event, denoting the provision of information that may allow a shortening or simplification of the plan.

Beyond this simple classification of "harmful" versus "helpful" exceptions, we distinguish between *unaccountable* exceptions and *accountable* exceptions. Unaccountable exceptions correspond to unexplained changes in world state brought about by the actions of *unknown agents*². For example, a system that is planning a conference may have to contend with the effects of the

²An unknown agent is an agent for which the system has no model or knowledge. Therefore the system is unable to reason about that agent's behavior, or interact with that agent

fire that has damaged the headquarters hotel, but the fire may not be held accountable for its action. It is such unaccountable exceptions that constitute the class of exceptions typically addressed by existing replanning systems [72, 31]. *Accountable* exceptions, on the other hand, correspond to exceptions generated by rational agents who are interacting with the system. In essence, since in such cases the agent is accessible to the system, they can be considered accountable for their actions.

When interpreting actions that defy expectations, it is also useful to distinguish between the manner in which an accountable exception is manifested and the actual motivation behind the inconsistent action. Hollnagel refers to this dichotomy as the phenotype (the way an action appears and is detected) and the genotype (the cognitive basis or cause) of an action that was not planned for [32]. If the underlying basis for the unanticipated action can be determined, steps may be taken to demonstrate that what appears to be an error is in fact a valid action, and informed corrective measures may be applied.

In summary, we have discussed four dimensions along which exceptional occurrences during plan execution can be examined:

- causality (the origin or motivation behind the exception);
- manifestation (the appearance and detection of the exception);

in any way. This is in contrast to *known agents* who are agents (such as users) for which the system has some model.

- consequence (the effects that the exception has on the current plan context);
- accountability (who, if anyone, can be held responsible).

All four dimensions of exception analysis are important to consider when trying to interpret and recover from an unanticipated event during interactive planning.

1.3 The Thesis

In this thesis, we present an approach for dealing with all types of exceptions that can arise in an interactive planning framework involving human agents, with particular attention to accountable exceptions. This class of exceptions has largely been overlooked by other systems. Current approaches to exception handling in planning have focused on reactionary tactics whose primary purpose is to restore the plan to a consistent state upon the detection of an arbitrary problem. While general replanning techniques are helpful for recovering from a problem introduced into a plan by a non-rational agent, such methods do not attempt to determine if there are underlying reasons for the exception that may explain how it could be incorporated into the current plan. A general replanning approach is insufficient for an interactive environment since it does not consider the role of the user in generating an exceptional action that may turn out to be valid within the context of the overall plan goals.

An unanticipated user action may represent an action that is semantically valid in the current plan context. This claim is based on the assumption that *users behave purposefully* when performing an action; their behavior is not simply random. In addition, a user is assumed to be guided by the same overall goal as the planner, and therefore a cooperative relationship between the system and user exists; no such assumption could be made when analyzing the action of an unknown agent. For example, upon detection of the hotel fire, the procedure that would revise the arrangements for accommodations during the conference would not assume a contributory goal on the part of the acting agent (the fire in this case). On the other hand, if the technical program were modified by the conference chair, procedures for revision should assume that there was a conscious reason underlying the decision that contributes towards the goals of the conference.

The natural intelligence and familiarity of humans with the application domain means that their actions, even when inconsistent with system expectations, are generally goal-directed. Thus, an important assumption in this work is that human agents in an interactive planning system are regarded as knowledgeable experts. The internal domain knowledge bases that human agents possess are generally much less bounded than the initial system domain knowledge provided by a knowledge engineer, as indicated in Figure 4. Much as knowledge engineers rely on the advice of human experts during the acquisition of knowledge to be used by an expert system, we assume that the actions of human agents executing actions in their domain of reference are positive examples of correct task performance and thus can give our automated assistant reliable guidance. Thus, we believe that many accountable exceptions

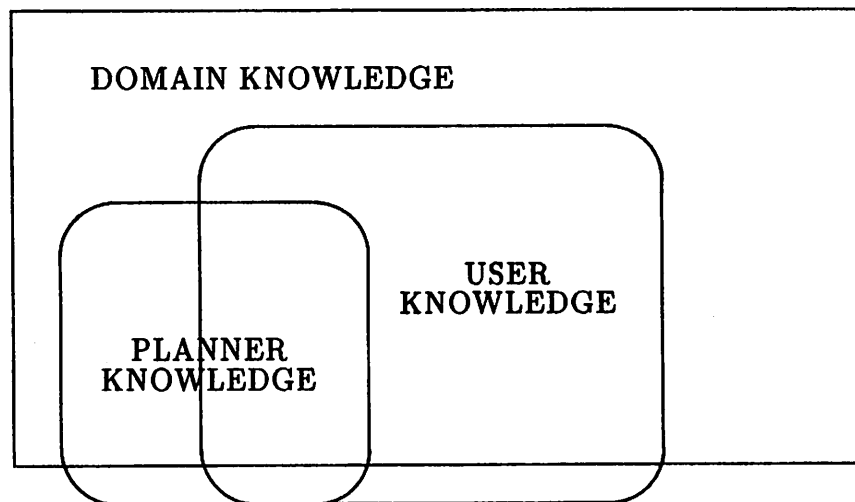


Figure 4: Knowledge distribution in cooperative planning

are representative of that portion of domain knowledge that is available to the user, but is unavailable (at least initially) to the planner (see Figure 4).

The user also may know about general planning principles that the planner does not consider. For example, the user may be motivated to take a shortcut while accomplishing a task in order to save time. Perhaps following an alternative route towards the accomplishment of a plan goal would save the organization some money, another general policy that implicitly guides the behavior of an agent. In general, we advocate the acquisition and modeling of policies of the domain and motivations on the part of the various known agents. These additional sources of knowledge can be used to aid the interpretation of exceptions that arise during plan execution monitoring.

Finally, it may be the case that a simple tuning of an inherently incomplete and incorrect knowledge base may legitimize what appears to be an excep-

tional action or inconsistent plan state. This tuning can take the form of the relaxation of violated constraints or incremental refinement of the knowledge base, through the addition of missing knowledge or modification of incorrect knowledge.

Of course, there is always the possibility that the unexpected action initiated by the user is simply an *error*, but our initial stance is that all plausible rationalizations and accommodations of the action should be considered before we assume that an error has occurred.

The overall goal of this thesis is to demonstrate an approach that intelligently handles exceptions generated by rational agents in a robust fashion, resulting in the continuation and completion of a task in spite of unanticipated events. Our approach is to augment a traditional AI planning system with mechanisms to cope with problems that can arise during execution due to an imperfect domain and incomplete planning algorithms, rather than to abandon the traditional AI planning paradigm for a purely reactive approach such as [10, 14]. The static and dynamic representations of domain knowledge are used in an attempt to transform the current plan into a valid alternative, or, put another way, to recognize an alternate plan.

In addition to minimizing the need for replanning by constructively accommodating exceptional user actions, a secondary goal of this work is to show how the handling of exceptions can help close the gap between planner and user knowledge. We show how techniques of constraint relaxation and knowledge base refinement are invoked in response to exceptions and help accomplish this aim. Focusing on user-generated exceptions is a valuable approach to-

wards addressing the inherent incompleteness and incorrectness of prespecified real-world domain knowledge bases.

The approach developed in this thesis can be used to effectively support applications of project management, such as the conference planning task used for illustration throughout this thesis. The ideas embodied in the approach presented here are also relevant in a wide range of other settings, i.e., wherever plan recognition is a factor in a problem-solving situation (e.g. *C³I*), or when the actions of cooperating human agents cannot always be anticipated correctly (e.g. intelligent tutoring).

1.4 System Sketch

In the remainder of this thesis, we develop and present the SPANDEX system for exception handling, whose general goal is to make a conventionally produced plan "elastic" in response to exceptions. This is done by modifying the current view of the ongoing plan and/or static domain knowledge in order to explain the exceptional behavior and allow continuation of planning and execution.

SPANDEX is part of the POLYMER system, a conventional hierarchical non-linear planner designed to assist human agents in the management of tasks in a cooperative setting [39, 40]. POLYMER constructs partial plans for user-specified goals and executes them in cooperation with agents. The actual actions taken by these agents are compared to expected actions, and when differences are found, the discrepancy is identified as an exception and SPANDEX

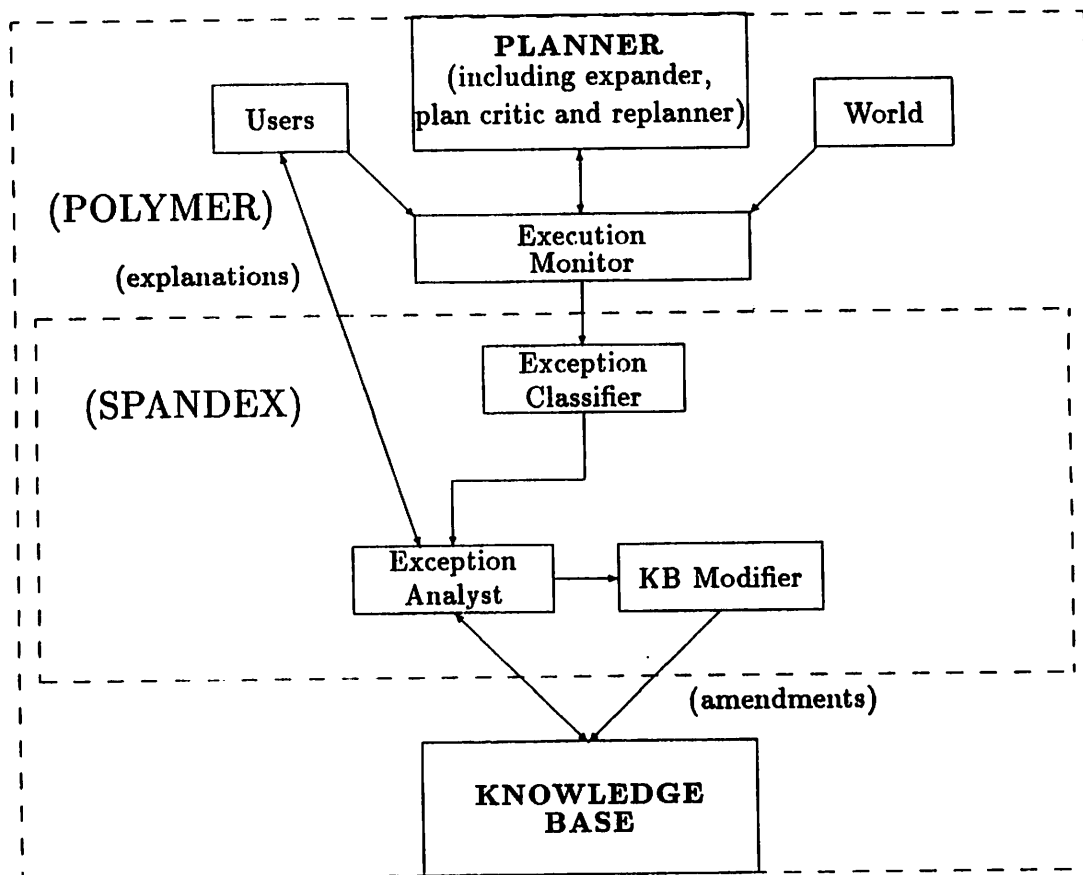


Figure 5: Architecture for a cooperative planning system

is invoked. In addition, the plan context is monitored for potential inconsistencies resulting either from planner actions or from the incorporation of a seemingly valid agent action. The detection of these inconsistencies also triggers the invocation of the SPANDEX system. This architecture is shown in Figure 5. A brief description is given here of the components of the SPANDEX architecture.

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 goal nodes 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 5).

The *exception analyst* is the core module of the SPANDEX system for exception handling, and produces *plausible explanations* of accountable exceptions. These explanations are constructed by a directed examination of domain knowledge, dictated by strategies attached to a set of prespecified *explanation basis* types. The explanation basis types include a set of purposeful *rationales* that are based on a model of human procedural behavior, including intentional reorderings of parts of the plan, shortcuts, substitutions, etc. A second type of explanation basis (in addition to the rationales) is based on the supposition of an inherently incomplete or incorrect initial knowledge base. The investigation of the search space for explanations is controlled by a set of heuristics that are generally applicable to plan recognition problems, specifically pertaining to locality, cost, and degree of evidential support³.

The exception analyst looks for evidence to substantiate one or more of the explanation bases during explanation construction. The knowledge examined during this search includes both static and dynamic knowledge. An exception is represented in terms of the action performed, the state achieved, and the resources being used, and is compared against: (a) pending and future goals and actions in the dynamic representation of the plan, and (b) expected re-

³These heuristics are described in Section 6.5.

source types. Possible matches are determined through unification and other techniques for determining partial matches, including the exploration of taxonomic knowledge about activities and domain objects. In addition, the known constraints on sub-activities in the plan and the objects being manipulated by these activities are accessed to determine possible relaxations of violated constraints that might alleviate the detected inconsistencies.

The output of the exception analyst is a set of ranked explanations, and the system conducts a dialogue with the user to select one of the candidate explanations. The result of this dialogue is a single explanation, along with approved changes or *amendments* to be implemented by the *knowledge base modifier* via modifications to the plan network for this particular plan instance or to the static plan library or object knowledge base. Thus a successful handling of an exception by SPANDEX can result in improved performance of the system, 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.

1.5 Example

In this section, we give an example of the operation of the components of the SPANDEX system just described, based on the domain activity of issuing a call for papers for a conference (briefly described and illustrated in Section 1.1). This scenario illustrates how an agent might perform a different activity than that anticipated by the planner, and then shows how the exception analyst is able to successfully construct an explanation showing that the accomplished

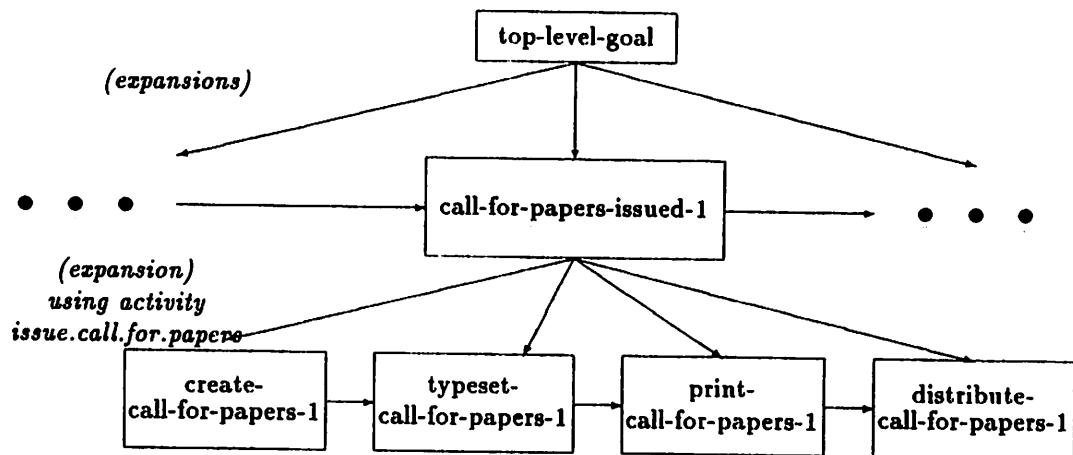


Figure 6: Partial plan outline for issuing a call for papers

action has a valid role in the plan since it subsumes a portion of the original plan⁴.

Suppose a conference organizer is setting up an academic conference. This task involves many interrelated steps, such as getting approval to hold the conference, soliciting and refereeing submissions, and arranging logistics such as housing and meals. As part of planning for the conference, a call.for.papers is usually issued by the program chair. The way this is commonly done is to prepare the actual content of the call for papers, have it typeset, then printed, and finally, distributed. The planner therefore selects the activity *issue.call.for.papers* to achieve the goal *call-for-papers-issued-1*. Figure 6 depicts part of the plan outline after the activity selection.

⁴An expanded version of this scenario is described in Section 7.1.2, illustrating the acquisition of a new activity description and including a listing of messages generated by the planner during exception handling.

```
JOES.TYPESETTERS: Please perform TYPESET.DOCUMENT  
To achieve (A STATUS OF CALL.FOR.PAPERS<001> IS TYPESET)  
OK  
I would like to do something different.
```

Figure 7: Asking the user to perform a task

The program chair next creates the content of the actual call.for.papers as anticipated. He is then asked by the system to request that the typesetter execute the typeset.call.for.papers.1 action (Figure 7). However, the organization which is sponsoring the conference has recently begun to offer a contract service to handle the various stages of producing and distributing a call.for.papers, and the program chair would like to make use of this new service. Since this was not the activity suggested by the planner, the agent responds that he would like to perform another activity (contract.out.call.for.papers) instead.

The system recognizes that an exception has occurred (agent-response-inconsistent-with-executable-expectation). The task for the exception analyst is to now determine how this alternative activity specified by the agent relates to the current dynamic model of the task. The SPANDEX execution.record summarizes the exception (see Figure 8).

Figure 9 shows the explanation constructed by the exception analyst. The explanation record states that since the activity contract.out.call.for.papers has been verified to achieve the same goal as the activity issue.call.for.papers⁵, this unexpected action may be a substitution for the abstract goal node that

⁵As based on an analysis of the domain knowledge base and the result of the unification of the goals which the activities are known to achieve. See Appendix E.


```

||| (Output) The *EXECUTION.RECORD* Unit in SPANDEX Knowledge Base
Unit: *EXECUTION.RECORD* in Knowledge base SPANDEX
Created by BROVERMAN on 3-6-89 13:07:56
Modified by BROVERMAN on 6-27-90 23:16:42
Member Of: ENTITIES in GENERICSUNITS

Maintains information about the current execution cycle.

Own slot: ACCOMPLISHED.ACTION from *EXECUTION.RECORD*
Values: P::ACM.CONTRACT.OUT.CALL.FOR.PAPERS

Own slot: ACCOMPLISHED.STATE from *EXECUTION.RECORD*
Values: #[Wff A P::STATUS OF P::CALL.FOR.PAPERS<002> IS P::ISSUED ]

Own slot: COMPLETE.EXPLANATIONS from *EXECUTION.RECORD*
Values: SUBSUMES.CURRENT.STEP.RECORD341

Own slot: EXCEPTION.TYPE from *EXECUTION.RECORD*
Values: S::AGENT-RESPONSE-INCONSISTENT-WITH-AGENT-EXECUTABLE-EXPECTATION

Own slot: MATCH.RESULT from *EXECUTION.RECORD*
Values: #[S(E::MATCH.RECORD:WORLD-STATES 6 :ACTIVITIES 4 :ACTIVITY-GOALS 5 :ACHIEVE
D-STATE-AND-PERFORMED-ACTIVITY-GOAL 1)

Own slot: OVERALL.EXECUTION.RESULT from *EXECUTION.RECORD*
Values: P::RESOLVED-EXCEPTION

Own slot: SELECTED.AMENDMENT from *EXECUTION.RECORD*
Values: S::REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT

Own slot: SELECTED.EXPLANATION from *EXECUTION.RECORD*
Values: SUBSUMES.CURRENT.STEP.RECORD341

Own slot: SUGGESTED.ACTION from *EXECUTION.RECORD*
Values: P::TYPESET.DOCUMENT

Own slot: SUGGESTED.NODE from *EXECUTION.RECORD*
Values: <TYPESET.DOCUMENT>FOR.AGENT<001> in CONFERENCE

Own slot: SUGGESTED.STATE from *EXECUTION.RECORD*
Values: #[Wff A P::STATUS OF P::CALL.FOR.PAPERS<002> IS P::TYPESET ]

```

Figure 8: The SPANDEX execution record

```

[[[ (Output) The SUBSUMES.CURRENT.STEP RECORD341 Unit in SPANDEX Knowledge Base
Unit: SUBSUMES.CURRENT.STEP.RECORD341 in knowledge base SPANDEX
Created by BROVERMAN on 6-27-90 23:15:22
Modified by BROVERMAN on 6-27-90 23:15:35
Member Of: SUBSUMES.CURRENT.STEP

Own slot: AMENDMENTS from SUBSUMES.CURRENT.STEP
Values: REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT

Own slot: COMPARISON.NODE from SUBSUMES.CURRENT.STEP.RECORD341
Values: <ISSUE.CALL.FOR.PAPERS>CALL-FOR-PAPERS-PROCESSED-AND-ISSUED<001> in CONFERENCE

Own slot: ENGLISH.EXPLANATION from SUBSUMES.CURRENT.STEP.RECORD341
Values: "Possible Rationale: SHORTCUT. The evidence is :
(((The performed action ACM.CONTRACT.OUT.CALL.FOR.PAPERS
achieved a goal which subsumes the expected action TYPESET.DOCUMENT.)))"

Own slot: EXPLANATION.BASIS from SUBSUMES.CURRENT.STEP
Values: S:SHORTCUT

Own slot: INCONSISTENT.STATE.SPEC.IN.FOCUS from SUBSUMES.CURRENT.STEP.RECORD341
Values: ISS22

Own slot: MATCH.RESULT from SUBSUMES.CURRENT.STEP.RECORD341
Values: #S(S:MATCH-RECORD:WORLD-STATES 1:ACTIVITY-GOALS 1:ACHIEVED-STA
TE-AND-PERFORMED-ACTIVITY-GOAL 1)

```

Figure 9: Chosen explanation

derived the issue.call.for.papers expansion (refer back to Figure 6). In addition, this substitution subsumes the action which we were expecting to take place (typeset.call.for.papers.1).

An amendment is then proposed for the explanation which specifies the changes that must be made to the current representation of the plan in order to restore consistency. The implementation of this explanation involves replacing the section of the plan network subsumed by the more abstract parent node (call-for-papers-issued-1) with the unexpected action (contract.out.call.for.papers.1). As a side effect, the unexecuted nodes in the expansion of call-for-papers-issued-1 are deactivated from the planner's predictions. A portion of the plan outline which now represents the hierarchical decomposition of the task is depicted in Figure 10. Normal planning and execution can now be resumed.

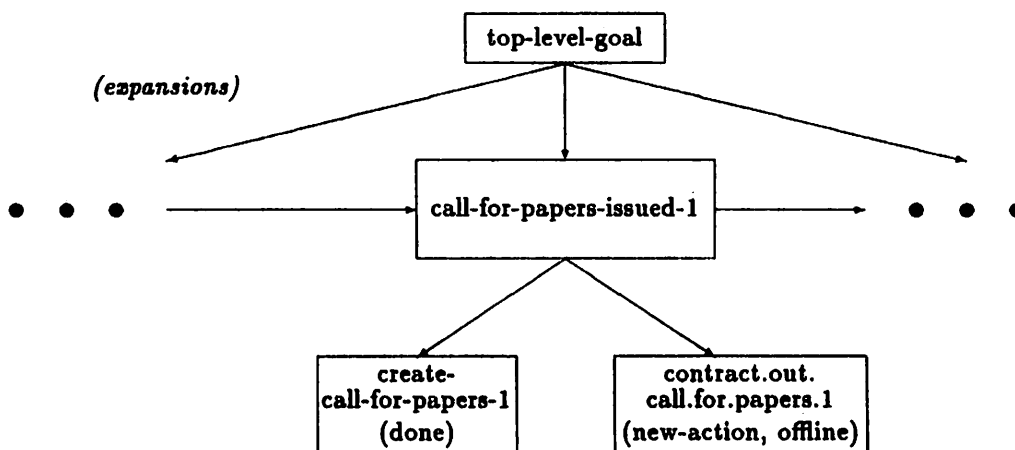


Figure 10: Partial plan outline after exception is resolved

1.6 Proof of Concept

The SPANDEX system has been implemented as a proof of concept for the ideas that are described in this thesis. SPANDEX runs on TI Explorers as a detachable module of the POLYMER planning system, and consists of approximately 6000 lines of Common Lisp code. The implemented system can successfully resolve exceptions in the categories that have been identified in this thesis, using a constructive approach based on explanation, constraint relaxation, and incremental knowledge refinement. The added capabilities of the POLYMER planning testbed when the SPANDEX component is activated demonstrates improved performance over existing planners when events do not proceed as planned. This improved performance, as well as the capability of the system to extend its domain knowledge through exception handling, is illustrated through the description of several scenarios that have been implemented and tested in the domain of conference planning. In addition, we conducted a survey that encouraged experienced conference organizers to iden-

tify potential deviations from an established plan for conference planning. The results of this survey lend credibility to the classes of exceptions and resolution strategies used by SPANDEX, and also reveal some limitations of the current approach and implementation.

1.7 Guide to the Thesis

In the remainder of this thesis, we present details of the design and implementation of the SPANDEX exception handling system as well as an illustration of the system's capabilities. First, to provide a comprehensive context in which to view this work, other work related to the topics addressed by this thesis is summarized in the next chapter. Common themes and comparisons to the approach presented by this thesis are discussed. In Chapter 3, literature on human deviations from procedural behavior is presented, in addition to a statement of our unique perspective of human error as *exceptions*.

Chapter 4 outlines the SPANDEX approach in more detail; presenting a formal model of planning and execution and giving a precise definition of what is meant by an *exception*. In Chapter 5, we discuss the features of our interactive planner that are important for exception handling. In particular, we describe: (1) the activity selection process used during goal expansion, (2) modifications that were made to support exception handling, and (3) a facility that was added to exploit constraints in order to intelligently guide resource selection by human agents. The implementation is then described in Chapter 6, with each module of the system described at length. This is the main body of the thesis. Limitations that are related to the implementation

are also discussed. Chapter 7 presents an evaluation of our experimental approach. Detailed scenarios from the current system are included to illustrate how SPANDEX successfully resolves several exceptions. In addition, an analysis is given of the results of a questionnaire that was distributed to test the adequacy of our exception taxonomy and explanation strategies. In Chapter 8, the contributions and limitations of this work are discussed, concluding with the suggestion of a number of directions in which this work could be continued.

CHAPTER 2

RELATED RESEARCH

In this chapter, we review other work that is related to our problem and approach. Other research has focused on various aspects of the execution monitoring problem ranging from issues in initial detection (such as how to introduce verification strategies or "sensors" to perceive execution results [22]) to strategies for learning via generalization of unusual observed behavior [21]. Prior to the description of particular systems, we first give an overview of the field of AI planning, with specific attention to hierarchical nonlinear planning. Since our approach can be viewed as an extension of the classical planning paradigm, we summarize some of the techniques and terminology used. Categorizations of plan flaws that can arise within the context of a nonlinear hierarchical planner are then discussed, along with the achievements and limitations of general replanning approaches that have been developed to handle such flaws.

Research has also proceeded in a number of other directions that are tangential to the problem of encountering and handling unusual events during planning and plan execution. One such body of work attempts to adapt and transform plans to fit newly encountered situations. Much of this work follows the "case-based" paradigm for reasoning about and interpreting new situa-

tions [36, 27]. Another sub-area of AI that is concerned with the explanation of novel phenomena is explanation-based learning (EBL) [21], and we discuss particular related efforts. In addition, other research is concerned with the completion of incomplete theories and learning that can result.

One important thing to note in preface to a review of these systems is that the techniques they provide are primarily relevant to the class of exceptions designated as *unaccountable*, since the problem states of concern are not produced by the volitional action of a rational agent known to the system. Thus, to provide a context for the important role of the human agent in settings that can benefit from non-autonomous "intelligent assistance," the following chapter provides an exposition of published studies that analyze how humans perform procedural tasks and taxonomies of human error that have emerged.

2.1 Planning and Nonlinear Hierarchical Planners

The field of artificial intelligence has produced many efforts over the last twenty years toward the construction and use of planners for the management of complex tasks. This work has adopted different stances, depending on the level of predictability assumed about the domain and the degree of advance knowledge and planning that is available or desired. A continuum seems to have emerged along which to classify planning research. At one end are the systems that adopt a view of the domain as highly unpredictable and thus place little value on extensive advance planning. These are the planners that have been termed "reactive," "reflexive," "tactical," or "situated," and are

exemplified by work described in [10, 61, 1]. Work along these lines has drawn a great deal of attention in recent years and is the topic of much debate.

At the other end of the reactivity continuum [18] lies what is considered to be the classical AI planning paradigm that has been incrementally refined over the last two decades. The general premise of this paradigm is a knowledge-based view; the world model can be represented in significant detail so that detailed plans can be constructed as predictive devices and can guide decision-making. In accordance with this outlook, such planners have been termed as strategic in nature [18]. The classical AI planning paradigm is still widely accepted, regardless of the recent challenge of its adequacy by situated action enthusiasts [1, 61]. In many domains, the frequent stereotypical performance of knowledge-intensive procedural tasks justifies the use of an approach that plans in advance. This thesis describes an attempt to extend classical AI planning to address some of its shortcomings rather than to abandon it in favor of a purely reactive approach; thus we focus on the development of systems at this end of the continuum.

The first classical planners were *linear* in that they produced a solution for a specified goal in terms of a completely ordered set of primitive actions [23, 62]. Two major problems were recognized with the linear planning paradigm: first, commitments were often made prematurely to arbitrary orderings of primitive actions that could result in an unsuccessful plan, and second, it was not possible to talk about plans and actions at different levels of abstraction. Plan repair strategies used by HACKER [62] could address the former problem after overly-rigid orderings were detected, but these techniques were specific and did not circumvent the original cause of the problem. Work by Sacerdoti [55]

first introduced the idea of *hierarchical nonlinear planning*, in which ordering commitments are postponed until absolutely necessary. In addition, partial plans in Sacerdoti's NOAH planner contained abstract actions that were iteratively elaborated and refined by planning actions. The development of other nonlinear hierarchical planners subsequently followed [63, 60, 68, 71], all of which built upon and refined the original idea put forth by Sacerdoti.

Some of the fundamental features of a typical nonlinear hierarchical planner are summarized below¹:

1. **State based representation.** The representation of the world is state-based, usually using some limited form of first-order predicate calculus. A state is expressed as a partial specification of the world model.
2. **Library of operators.** The planner has accessible to it a set of *operators*. Each operator can be applied to a given state if its *precondition* is satisfied in the partial world model that is represented by that state. An operator defines a state transition from the state to which it is applied to a new state in which the *goal* and *effects* described by the operator are valid².
3. **Different abstraction levels for planning.** Operators can be *abstract* or *primitive*. An *abstract* operator can be decomposed into a partially ordered sequence of primitive operators and/or other abstract operators that can be applied to accomplish the overall goal. Planning is accomplished through an iterative refinement of abstract operators.

¹This list is based on an excellent survey of classical planning systems found in [73] where the interested reader will find a more thorough exposition.

²The goal and effects of an operator are also represented as states.

4. **Solution as a partially ordered sequence of operators.** A problem for the planner is posed as (1) an initial world state, and (2) the goal state that is to be achieved. The output of the planner is a network of partially ordered operators and/or subgoals (referred to as a *plan network*), that when applied to the initial state will result in the goal state.
5. **Nonlinearity and least-commitment approach to action ordering.** The suboperators to be used in a plan may be partially ordered to allow concurrency. Ordering is only introduced when necessary, for example if a potential interaction is detected.
6. **Constrained plan variables.** Plan variables with attached constraints are specified within operator definitions to denote the resources to be used. The use of such variables avoids premature commitment to instantiated resources while at the same time narrowing the set of possibilities via the attached constraints.
7. **The STRIPS assumption.** Only the state changes explicitly stated by an operator in its add and delete lists (or equivalent statement of effects) are assumed to have taken place after its application. No other changes will have occurred.
8. **Planning and execution can be interleaved.** Execution of primitive actions may take place before the plan is fully elaborated into primitive operators. Information made available as a result of execution may be necessary for further planning.

2.2 Plan Flaws and Replanning

Now that the basic features of the types of planners under consideration have been described, we can turn our attention to problems that can occur during interleaved planning and execution in a volatile world. Some planning systems have anticipated the need to compensate for such contingencies [32, 73, 4], and develop various categorizations of plan flaws that must be handled. In this section, we review the types of flaws that may be encountered by each of these systems.

Hayes

Philip Hayes' early work on replanning in hierarchically structured planning systems [32] takes a modified backtracking approach to the problem of unexpected events. The basic idea is to recognize actions that have become unexecutable as a result of an unexpected event and remove them from the plan. In addition to the hierarchical plan structure, a hierarchical decision structure is maintained to represent the planning decisions responsible for the plan structure. When sections of the plan are discarded, the relevant links into the decision tree are followed, and the decision nodes responsible for generating the affected part of the plan are also discarded, along with any other plan nodes that resulted from these decision nodes. Alternative subplans are then generated at these identified decision points to produce a valid plan. The basic idea of this system is to selectively undo affected parts of a disturbed plan and redo the corresponding decisions and subsequent execution. No attempt is made to specifically identify the types of problems caused by unexpected

events or to relate the effects of an unanticipated event to expected effects or events remaining in the plan. Unexpected events in this system are represented by arbitrary state changes (e.g. train cancellations) and assumed to be detrimental.

NOAH

In contrast to most other classical planning systems that are designed with a robot executor in mind, Sacerdoti's NOAH planner was oriented towards a human agent who was to execute the constructed plans. Sacerdoti raised the possibility of exceptional states being introduced by an internal agent, such as a system apprentice using a planning system. NOAH monitored the human agent's actions and responses by querying the user in regard to the completion of dictated plan actions. The user could respond in such a way as to elicit further explanation and advice from the system. In addition, upon the detection of a discrepancy in the world model, NOAH would engage the user in a dialogue to verify the parts of the plan leading up to the error in order to pinpoint the source of problem. Once identified, NOAH would then plan for new actions to recover from the problem state while maintaining as much of the original plan as possible. While the design and implementation of NOAH did not provide a comprehensive approach to address the issue of user-generated exceptions, it was one of the earliest pieces of work in planning to recognize the need for a more sophisticated handling of this issue.

SIPE

Wilkins implemented the SIPE planner [71] significantly extending the capabilities of its predecessors such as NOAH and NONLIN [63]. A sophisticated domain-independent replanning component [72] makes use of a rich plan representation (including preconditions, conditional effects, constraints, plan rationale, etc.) to categorize the possible problem types and to propose "fixes" for the areas of the plan adversely affected by an unexpected world state change. Such unexpected occurrences are represented as "mother nature" nodes that are created and inserted into the plan network at the current point of execution. Problems resulting from a "mother nature" node are then computed in a general fashion and addressed. As a consequence of the underlying representation in SIPE, a fixed set of six types of problems were found to occur and are detected by the problem recognizer component of the system. They are listed below, along with the responses taken by the system:

1. **Purpose not achieved; redo.** The main effects of the step just executed (prior to the "mother nature" node, that contains the unexpected effects and represents the unexpected event) are not achieved at this point. The SIPE response is to "redo" the action by inserting a goal node representing the unachieved purpose after the "mother nature" node.
2. **Previous phantom not maintained; reinstantiate or retry.** A phantom node³ occurring before the current execution point protected until sometime after the current execution point is no longer a phantom.

³A phantom node is a goal node in a plan network that has been determined to be true in the world state at that point in the plan as a consequence of previous actions. No further expansion or execution is needed to accomplish the goal represented by that node.

The SIPE response is to try to reassign (restantiate) bindings of the phantom goal to make it become true again; if unsuccessful, the phantom is converted to a goal node and reached (retry).

3. **Unknown process variable; restantiate.** If a node in the plan represents an action, and is operating on a resource that is specified as a variable or a non-existent object, this variable must be bound to an existing object instantiation. SIPE instantiates a binding for the unknown argument.
4. **Future precondition no longer true; restantiate or pop-redo.** A precondition has become false that occurs after the current execution point and is the justification for an expansion choice. SIPE first tries to rebind the precondition with another variable binding (restantiate). If unsuccessful, SIPE backtracks and splices out the entire *plan wedge*⁴ resulting from the higher level goal that first specified the violated precondition. An expansion of that goal node is regenerated.
5. **Future phantoms no longer true; restantiate or retry.** A phantom node that occurs after the current execution point is no longer true. If the attempt to restantiate bindings in the phantom goal proves unsuccessful, the phantom is converted back to a goal.
6. **Parallel postcondition not true; insert parallel redo.** One of the postconditions (effects) in a parallel set is untrue; the system responds

⁴The definition of a *plan wedge* in SIPE is based on Sacerdoti's original definition [55]. A *plan wedge* is made up of those nodes in the plan network that are the result of one or more levels of elaboration of some abstract goal node that was previously in the plan network.

by inserting a parallel set of goals (one for each untrue postcondition) after the join node.

An additional replanning action (**pop-remove**) is available to take advantage of serendipitous effects that may have resulted from an unanticipated occurrence.

Both SIPE and the Hayes replanning system are capable of recovering from unexpected effects that are introduced into the environment of the plan. Because of the differences in the underlying representations, different methods are used for determining the affected portions of the plan and for recovery. Both systems assume that actions that are actually executed take place as expected and that some unknown process brings about the unexpected effects. These systems thus intentionally "avoid having to model the way in which the unexpected effects might interact with their expected counterparts ⁵." As we will discuss in later chapters, this interaction of unexpected effects with anticipated effects and actions in the plan is precisely what we do address in the SPANDEX system for exception handling.

IPEM

IPEM [4] is a recent system that integrates the processes of planning, execution, execution monitoring, and replanning using the plan representation of TWEAK [13]. IPEM applies a production system to uniformly implement planning, execution, and replanning actions, and uses a set of heuristics to exert control and resolve conflicts.

⁵From [73], p.150.

IPEM uses rules in the production system to identify "flaws" in a plan network (left hand side of a rule) and to apply "fixes" (right hand side of a rule). The flaw/fix rules apply uniformly during the process of plan elaboration as well as to plan execution and monitoring. The flaws identified in IPEM are listed below with their corresponding fixes:

1. **Unsupported precondition; reduce.** If there is no record of a node establishing the state required for a later precondition node in the plan, then establish or introduce a (possibly) prior node as the producer of this state.
2. **Unresolved conflict; linearize.** When a proposition and its negation are both determined to be asserted sometime within the same plan range, a *clobbering* may result, and a linearization of the two nodes (that assert the proposition and its negation) must be performed. This is a standard conflict resolution technique used during planning.
3. **Unexpanded action; expand.** This is not really a plan inconsistency, but considered to be a flaw to maintain uniformity. If a node in the plan network can still be expanded, then expand it.
4. **Unsupported range; excise range.** When a proposition is being protected during a subrange in the plan and that proposition is no longer true, the protected range declaration should be excised from the plan specification. Note: this automatically creates an unsupported precondition flaw on any subsequent nodes that are relying on the proposition, and these newly introduced flaws are dealt with accordingly.

5. **Unexecuted action; execute.** Again, this is not really an inconsistency, but unexecuted actions are treated as flaws for uniformity to invoke their execution.
6. **Timed-out action; excise action.** This flaw is particular to the IPDM representation of actions and the time range in which the effects of an action can be delayed. When the range expired and the effects have not been established, the action and its declared protection ranges are removed from the plan.
7. **Unextended range; extend range.** If a protection range can be extended so that the beginning of the range may be established on an earlier node than the current range start node, this is considered a flaw. The action taken is to extend the range to the earliest node that establishes the condition that is to be protected.
8. **Redundant action; excise action.** If an action has no protection range that denotes it as establishing a state required for a subsequent node, it is considered to be redundant and is removed from the plan⁶.

These rules fire when the specified flaws occur, and are put on a task agenda. Conflict resolution is accomplished using domain supplied heuristics, which prioritize the criticality of the flaws, and thus determines the order in which they are addressed. Preference is given to fixes that introduce fewer

⁶Note that the implicit definition of redundancy here implies that a strong causal model is required. If an action is not an explicit "provider" for a later action, it is considered unnecessary. This is an overly strict requirement for many settings. In semi-structured complex domains, it is not always possible or desirable to state all causal connections between among the steps of of a typical procedural task.

constraints in the plan. The flaw preference order established by IPEM is the following: unsupported range, unextended range, timed-out-action, unresolved conflict, unsupported precondition, unexpanded action, unexecuted action.

IPEM makes the following assumptions:

1. STRIPS assumption.
2. The current world description may not be complete. In other words, the "closed-world assumption" is not maintained.
3. The current world description is dynamic and can change anytime.
4. Actions can fail to achieve their intended effects, and partial success may be subsequently exploited.
5. The effects of an action can be delayed, thus allowing for potential time-outs.
6. Problems may be unsolvable given an initial world description.

In summary, IPEM provides for four "true" flaws: unsupported precondition, unresolved conflict, unsupported range, and timed-out action. Two additional flaws (unexecuted action, unexpanded action) are introduced as special cases to maintain uniformity of representation, and the remaining two flaws (unextended range and redundant action) are detected in order to introduce optimizations into the plan to take advantage of serendipitous effects.

CHEF

A case-based approach to planning that pays particular attention to failures is exemplified by Hammond's CHEF system [29, 28, 27, 30]⁷. Past plans are indexed by the goals they achieve as well as by failures encountered. In CHEF, a failure is noticed when comparing the results of a plan simulation with expected results. If an expected goal is not among the simulation results, or if an undesirable state is reached, a failure has occurred, and a causal explanation of the failure is extracted from the simulation trace. Elements of this *causal explanation* are then used as indices for the retrieval of appropriate plan repair strategies to patch the plan.

Plan failure in CHEF is handled by applying a set of domain-independent repair strategies to address the identified violations. In general, faults in CHEF involve the violation of one of the types of conditions that are represented in CHEF⁸, or the violation of an overall plan goal. These failures are indexed according to what actually caused the violation, such as a *side-effect*, a *desired-effect*, or a *feature* of an object. There are 16 such failure types, and one or more of 17 available repair strategies are assigned to the type of failure as potential remedies.

The repair strategies dictate measures such as the reordering of steps in the plan, the insertion of new steps to counteract bad effects, or the substitution of steps or tools that have similar goals but whose conditions are satisfiable or

⁷Case-based approaches will be discussed more generally in Section 2.3.

⁸Hammond identifies these conditions as *preconditions*, *satisfaction* conditions, *maintenance* conditions, and also addresses more domain-specific conditions such as *balance* conditions (e.g. specifically regarding the ingredients being used in Szechuan cooking).

whose features or effects do not produce a violation. Specific fixes also exist to extend or shorten the time duration of a step, or to alter the balance of a violated balance relationship⁹.

The CHEF system is described further and contrasted to the approach described in this thesis in Section 2.3.

GORDIUS

The GORDIUS system of Simmons [59, 58] uses a paradigm called *generate-test-debug* (GTD) to construct plans (called *hypotheses*) that could be responsible for observed geological formations. The task addressed is primarily one of interpretation. Since the plans generated are only approximations, they are handed off to a simulator within the system, and any inconsistencies between results of the simulation and the observed state, or inconsistencies encountered during the simulation, are explained by the debugger within GORDIUS. Thus, the definition of what constitutes a plan flaw is a very general one. While Simmons' does not explicitly present a taxonomy of what he refers to as *bugs*, he implicitly describes the bugs that may be detected by GORDIUS as including the types of flaws we have mentioned (inconsistent variable value, untrue condition, untrue precondition, violation of ordering, etc.). In addition, as in IPEM, Simmons states that normal unachieved goals in an unexpanded plan can be treated as bugs, in which the response of the debugger would be to elaborate them [59].

⁹This is the only real domain-specific repair.

Plans in GORDIUS are constructed by selecting a set of *scenarios* that apply to and can explain the observed situation. These scenarios are essentially associational rules that state patterns to match against observed features, precondition patterns (that are used to backchain), and a set of partially ordered and constrained events that make up the definition of a process that could produce the observed pattern. Multiple scenarios may apply and be selected. The constructed plan(s) are then simulated.

Bugs arise when results (intermediate and final) generated by the simulator are either inconsistent with required conditions in the plan to be simulated or in the final observed state. These inconsistencies are possible for two reasons. First, there is not always a known process that could be responsible for every observed and predicted state. Second, interactions may occur when combining multiple explanatory processes, resulting in inconsistencies. In other words, a number of different contributory solutions that are locally consistent may produce a globally inconsistent solution.

The justification for Simmons' approach is predicated on the crucial assumption that the set of scenarios have a high degree of independence, while not being completely independent. Without a high degree of independence, the plan generator would be extremely inefficient and would produce an abundance of extremely flawed plans. On the other hand, since the bugs usually result from combining locally consistent plans, a total independence of processes would result in generated plans that were always correct, removing the need for simulation and debugging.

Since more than one valid explanatory plan may result from this process, rating strategies rely on two major heuristics: (1) prefer plans that reduce

the number of outstanding unsolved goals, and (2) prefer plans that dictate a shorter sequence of responsible processes. Thus, the repairs proposed for bugs are evaluated for their goodness by seeing whether they introduce new bugs or serendipitously repair others.

Simmons treats all bugs in a uniform fashion. An analysis is performed to identify an assumption that may be responsible for the bug; this assumption is hypothesized to be faulty and is retracted in order to restore consistency to the plan. Thus, a fix is essentially a credit (actually "blame") assignment action followed by a correction to the plan. Simmons claims there are six general fixes that cover most of the bugs encountered. Several of the fixes in Simmons' system involve relaxing the closed-world assumption. These faulty assumptions constitute a taxonomy of potential *explanations* of flaws rather than a categorization of flaws themselves, and are listed below with possible ways to implement retraction:

1. Faulty assumption: A particular event occurred. Fixes: delete the event, or replace it with a similar event that differs in effects or preconditions.
2. Faulty assumption: The value of formal parameter ?x of ?event is ?val. Fixes: change the value of the formal parameter.
3. Faulty assumption: A specified temporal ordering exists between two events. Fixes: reorder the events.
4. Faulty assumption: The value of a variable persists over a specified interval. Fixes: introduce events between event1 and event2 that could have caused the observed change. If existing events could have caused the

desired value then a new ordering is established instead of introducing a new event.

5. Faulty assumption: An object exists at a specified point in the plan. Fixes: introduce event that could have destroyed the object.
6. Faulty assumption: Only objects of type x are in set-a. Fixes: introduce a new member of set-a by introducing a new event that creates it.

Simmons' work has explicitly been referred to as a *transformational* approach to planning, since in addition to adding detail as plans are being refined, decisions may be changed, nodes may be deleted, etc. The other approaches reviewed here may also be regarded as transformational. In Section 6.1.3 we compare the categorizations of flaws identified by some these systems with each other and with those flaws identified in the POLYMER system and handled by SPANDEX.

Limitations of existing replanning approaches

The systems just reviewed (Hayes' planner, SIPE, IPEM, and Simmon's transformational planner) refer to actions of some uncooperative unknown agent who interferes with the plan, creating problems. In SIPE, this agent is referred to as "Mother Nature," and represented by the "baby" in IPEM¹⁰. In our view, a "cooperative" agent can intentionally introduce an unexpected event or effect that should be accepted and incorporated into the plan and not simply

¹⁰Originally introduced by Schoppers in [57].

recovered from. The systems just discussed are unable to handle contingencies of this type.

Thus the approach we take in SPANDEX is conceptually different than the approaches outlined here. Rather than simply inserting an unexpected action and reacting to the introduced problems, we hypothesize what that action represents in terms of the rest of the plan. The system attempts to make a perfect or close fit of this action into the plan, and then computes the remaining problems and recovers/reacts as necessary. Replanning is still a necessary function to provide in a realistic planning system, but should not always be invoked blindly. In addition, replanning may not always be necessary when cooperative human agents are generating exceptional actions for which they may be held accountable.

2.3 The Adaptation and Transformation of Old Plans to Fit New Situations

A focus that has emerged recently in planning research is the reuse of old plans during new plan construction. Previously constructed plans are adapted or transformed to fit a newly encountered situation. Although our work is directly aimed towards the handling of exceptions as they may arise when dealing with an incomplete knowledge base in an interactive system, we share some of the goals and the techniques of these approaches that make use of past plans.

Several of these efforts [3, 2, 27, 66] can be regarded as a type of *case-based planning* that is a special application of the more general *case-based reasoning* paradigm in which old cases are retrieved and modified to reflect newly encountered cases [35, 26]. In general, case-based approaches place a heavy emphasis on maintaining a well-organized memory of previous problem cases in the domain that can be accessed in order to interpret or handle a new situation. In other words, past experience is relied upon to create new plans or debug problematic ones.

CHEF

Hammond's CHEF system has already been described in Section 2.2 and is a case-based planner that pays particular attention to plan failure. Memories of failures are recorded by marking the actual features of the initial situations that are associated with the failures. The major thrust of this work is to show that potential failures in new planning situations can be avoided through anticipation. The anticipation of failures is triggered by the activation of memories of past failures.

Hammond relies on a strong causal model of the domain to debug plans that are imperfect and to assign credit to the features at fault. Since organizational procedures have causal information associated with the temporal ordering restrictions, we can employ a similar approach in categorizing problems in the plan network that are caused by the occurrence of an exception. The maintenance of causal information is a crucial element needed to understand the relevance of a user-generated exception to the goals of the ongoing plan.

There are several important differences between the approach taken in CHEF and our focus in SPANDEX. First, CHEF is a very specific type of planner, that differs significantly from our planning paradigm. Planning is accomplished by simulating the cooking process specified by a predetermined recipe, and the results of the trace are compared with the goals of the recipe to determine plan failure. The entire plan is generated and simulated before any analysis is performed. Thus the planner in CHEF is neither hierarchical nor interactive. As a result there is no concept in CHEF of planning history or levels of plan abstraction. In addition, since planning is not interleaved with execution, and there is no user input, the only discrepancies that are noticed are those that result at the end of a fully simulated plan. Thus failures or discrepancies are not detected "mid-execution," and no attempt is made towards incremental plan repair. The mode of *execution* in CHEF is really simulation, and is not realistic for many domains. Much of the focus of our work is to explore the hierarchical derivation of partial ongoing plans, in settings where execution must be interleaved with planning. Our goal is to identify serendipitous effects of unexpected actions *as they happen*, thus making dynamic modifications to the plan if possible that can allow the remainder of the plan to be significantly shortened or reconfigured.

In addition, CHEF does not do any analysis of the actual operators used in a plan, and does not include plan operators in the abstraction hierarchy used to determine plan similarity. Similarity in CHEF is determined from the comparison of features of plan situations alone, that are abstracted from the goal inputs. We are concerned with understanding relationships between observed and expected *actions* as well as the actual objects involved in the

plan. CHEF pays little attention to the procedural structure of a plan. We have taken a more integrated view of plan operators, objects, and agents, and our activity descriptions are quite complex.

Tenenberg

Tenenberg's [66] work on the use of abstraction in planning can also be considered to be a case-based approach. This research develops an algorithm to constrain search for a new plan by making use of condensed old plans, in the form of plan graphs. This work demonstrates a recognition of the utility of a generalization hierarchy of both plan operators and domain objects when constructing a new plan. Anderson and Farley extended Tenenberg's work with an implementation [5]. A generalization hierarchy of operators is constructed by combining operators in two ways: (1) those that share one or more (but not all) preconditions or effects clauses, or (2) those that share all preconditions and effects but the object types being manipulated differ. In addition, an object hierarchy is constructed based on object function that is determined by operators that get categorized together during the operator hierarchy construction. For example, if `((holds(robot ?x) or holds(robot, ?y)) and block(?x) and hammer(?y))`, then `block(?x)` and `hammer(?y)` become instances of a new generalized object `holdable-object`. Our work follows similar lines in making use of taxonomic links as a primary semantic resource to explore when encountering an exceptional action or object. Like Tenenberg, we also see this approach as one that provides savings in search complexity. Whereas Tenenberg constrains the search using an abstract plan graph condensed from an earlier plan, we constrain our search for a consistent plan by

constraining our search to the knowledge closely related to either the expected action or the exceptional action.

PLEXUS

Alterman's PLEXUS planner [3] implements an approach that emphasizes the structure of background knowledge as an important means for altering old plans to fit new situations. Alterman's *adaptive planner* treats failing steps of a plan as representative of the category of activity that should be performed, and thus traverses generalization and specialization links in order to find a step that fits in the new situation. We make similar use of the static library of domain activities and objects, by exploring the relationships between unexpected actions and object parameters and those used in a stereotypical plan for the task. Alterman concentrates on resolving any of four identified types of *situation differences* that can arise in trying to refit an old plan to a new situation: *failing preconditions*, *failing outcome*, *differing goals*, and *step-out-of-order*.

Whereas the similarities between our work and Alterman's are numerous, our goals differ. Alterman is attempting to construct a new plan for a known novel situation, while we are in a sense trying to *recognize* an ongoing modification of an existing plan. Also, while we assume a great deal of initial knowledge about the domain we are operating under the premise that our knowledge may be incomplete. Alterman, on the other hand, seems to assume a knowledge-intensive environment that is complete. An additional difference is that in Alterman's work, the planner is responding to new constraints imposed by the situation, whereas in our environment, the planner must respond

to unusual actions by the user and is inherently performing some recognition. In our approach, we would like to take advantage of having intelligent agent(s) "in the loop." Alterman hints at the fact that his system is an interactive one, yet the user seems to be a passive agent and is not engaged in a dialogue to provide guidance during the construction of the new plan.

Collins

The work of Collins [15, 16] constitutes another case-based approach to plan reformulation. This work applies a model to monitor the execution of general plans in potentially novel domains, and new plans are created in response to specific problems. General strategies are used as "blueprints" for constructing new plans. These strategies incorporate plan transformation rules that actually alter the problem solving strategy underlying the plan. This method goes beyond the more fine-grained approach of replacing subparts of plans with alternatives but may introduce tradeoffs regarding the optimality of the plan.

Huff

Another transformational approach is that of Huff [35], in which special meta-plan operators are applied to a plan according to a set of prevailing conditions to handle special cases and failure recovery. Specified parts of a plan are dynamically modified according to the transformations stated in the applicable meta-plan operators, such as the addition and deletion of subgoals, or the changing of bindings of operator variables.

In addition, Huff [34] gives an algorithm for *reconciliation* that changes the world model in order to accommodate an interpretation of an ongoing plan which is inconsistent with current beliefs. This approach is a variation on the standard dependency directed backtracking approach used by truth maintenance systems (TMS) [22] to resolve contradictions. The analog to this process in the SPANDEX system can be found in the algorithms for knowledge base extension through constraint partitioning and the relaxation of violated constraints (see sections 6.4.5.1 and 6.4.5.3). All three techniques involve the revision of the current world model in order to resolve detected inconsistencies. However, in SPANDEX, rather than manipulate a TMS belief structure, changes are made to the structure of the underlying domain model and may persist between dynamic invocations of the planning system. The focus of the two systems is also different: the GRAPPLE system of Huff is a plan recognition system whereas the SPANDEX system performs incremental knowledge acquisition through exception handling.

2.4 Explanation-Based Learning Applied to Novel Phenomena

Much work has been published recently under the rubric of *explanation-based learning*¹¹. The goal of the explanation-based learning approach is to construct an *explanation* for a novel concept, and then to generalize the explanation of the specific instance into a more general and operational concept

¹¹I am using the term *explanation-based learning* to subsume other approaches designated as *explanation-based generalization*, *analytic learning* and other similar paradigms.

definition. More formally, explanation-based approaches apply a *domain theory* and a *goal concept* to a *single training example* to construct an *explanation* as to how the training example is an instance of the goal concept. Techniques are then applied to transform the specific explanation into a more general specification of the learned concept.

DeJong

The work being done by DeJong et. al. [20, 21, 19] is particularly relevant to our work in handling exceptions during plan execution. DeJong's *explanatory schema acquisition* (ESA) is couched in problem-solving terminology rather than being presented from the more typical theorem-proving viewpoint. This orientation is more compatible with our planning perspective, incorporating preconditions, effects, goals, and motivations. The entities used by their systems are actions, states, and objects; that are similar to the entities used in a knowledge-based hierarchical planner. The general goal of ESA is to construct and generalize an explanation for an observed novel sequence of problem-solving operators. DeJong often refers to his approach as *learning by observation* [19], a term that could also be applied to our approach that detects an exceptional action during plan execution monitoring, and attempts to construct an explanation for the role of that action in the plan. In addition, DeJong addresses the need to infer missing steps. This is similar to the determination of activities that potentially have occurred "off-line," that is a technique used in SPANDEX when an action in a plan is detected as occurring out of order (see Section 6.3.3). However, our approach differs from this work in the following ways:

1. Our planning paradigm is an interactive one, so plans are developed and executed in an incremental fashion. Therefore, we do not have a full-fledged detailed sequence of actual plan operators invoked from which to construct an explanation¹².
2. We are often unable to construct a complete explanation for an exceptional problem-solving strategy taken by the user and must resort to negotiation in order to acquire truly new information to complete the explanation. In other words, we are not assuming a complete knowledge base to start, and in addition to the use of explanation as a method for *restructuring* the knowledge base, we also provide for the acquisition of completely new knowledge about the domain.

Rajamoney

Rajamoney et. al describe an explanation based learning approach that initially assumes an incomplete and incorrect model of the domain [49]. They use the observation of an unexpected effect in the world model as an opportunity to extend and debug the domain knowledge of the system. Their approach is to hypothesize known processes that might have caused the inconsistency, and to use directed experimentation to question the beliefs that fail to support the existence of these processes. This approach is similar to that of Simmons in that presuppositions that fail to support the detected event are questioned. It differs from the approach taken in SPANDEX in that it addresses only the

¹²Both DeJong's and Hammond's approaches rely on having a complete plan in order to determine inconsistencies between plan results and expectations.

type of exception that may be generated by an underlying "agent-less" process. In SPANDEX, we take a more behavioral viewpoint, and are interested in handling the cases arising in an interactive planning system where intelligent agents actually carry out much of the execution of the task.

2.5 The Completion of Incomplete Theories or Explanations: Learning

In addition to work in the case-based and explanation-based reasoning communities, other research in machine learning and databases addresses the idea of learning by completing incomplete theories or explanations. These approaches are different from explanation-based learning in that the learning comes from improving the domain theory, whereas EBL assumes that the domain theory is capable of producing the explanation.

VanLehn

VanLehn describes an approach in which pieces of knowledge that can complete an incomplete explanation can be added to the domain theory [68]. A parse tree is constructed for the action sequence, which in their case is the solving of an arithmetic problem. The parse tree is constructed from the top and from the bottom as far as possible. Two types of learning are discussed: a) simple: a subprocedure must be generalized to complete a parse; i.e. relax the applicability condition that was violated, and b) complex: no modification will be enough so new information must be acquired.

Borgida

The work of Borgida and others in accomodating and learning from exceptions in databases [8, 75] also addresses similar problems and offers interesting solutions. This work accepts the assumption that databases are incomplete. Therefore, information that violates integrity constraints stored with the database should be allowed, and techniques must be available to accomodate this inconsistent information without rendering the database inconsistent. There are two general techniques provided. In the first approach the violated integrity constraints are modified; this is in essence a relaxation of the integrity constraints in the knowledge base so as to allow the exceptional data. The second type of technique is an explanation-based approach, and uses a detailed domain theory in order to show why the violated constraints needn't hold for the exceptional case.

ODYSSEUS

Wilkins' work on the ODYSSEUS system [70, 71] can also be viewed as an approach at learning by completing incomplete explanations. The basic idea of this system is to give apprenticeship capabilities to a knowledge-based expert system. A "confirmation theory" is used to evaluate proposed repairs (additions and corrections) to domain knowledge in order to make a failed explanation valid.

CHAPTER 3

PERSPECTIVES ON DEVIATIONS BY HUMAN AGENTS DURING PROCEDURAL TASKS

Since one of the major contributions of this thesis is to address the human component in plan execution and plan revision, in this section we review work that has been done on analyzing human procedural behavior and deviations from typical plans of action. First we identify the novel way in which we regard aberrations that are traditionally termed as errors. We then give an in-depth look at studies that document the types of human errors that have been observed during the monitoring of procedural tasks. In a later chapter (Chapter 6) we show how we integrate some of the results of these studies with our constructive interpretations of such deviations.

3.1 Errors as Exceptions

In our work, 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 inflexible and the procedural knowledge involved is clearly 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.

In the remainder of this section, we summarize the research that has been published on human error. It is important to note from the start, however, that when referring to the same phenomena in the context of our system and approach, a shift in perspective will be necessary. The taxonomies that are presented here generally refer to such deviations as "errors." We prefer to refer to some of the same syntactic behaviors as *exceptions* to avoid the negative connotation associated with the word "error." 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 that may have generated the action. In general, the taxonomies we examined intermix these two aspects of exceptions.

3.2 Related Work on Human Error

Several researchers in cognitive engineering have noted the importance of recognizing and interpreting human errors that occur during the performance of procedural tasks. The limits of automated systems with regard to the prediction and capture of all potential errors and the need to increase error tolerance have been described. For example, in summarizing a collection of studies that address the problem of handling human error arising during the use of current technologies, Rasmussen claims that errors are part of normal everyday life, and therefore a robust system should be able to overcome whatever consequences that may ensue. He claims that real systems need to have models of human failure and should incorporate mechanisms for recovery in order to be robust [52].

In addition, it has been recognized that systems may often categorize human actions as errors when they may be meaningful actions from the perspective of the agent who performed them. For example an action and its result may be appropriate from the point of view of the agent who performs it, but may be judged to be inappropriate by the system [51]. Since this discrepancy is the result of the behavior of the total man-task system, it may be either the man or the system that is at fault. In addition, any action that is voluntarily performed by an agent must seem at the time to be valid to that agent, and the intention that must be behind it can not be ignored. In this sense, it can be argued that all volitional actions are, at some level, correct [65].

It has also been noted that error is often a necessary component of the learning process. One of the things that humans do very well is to learn

from their mistakes; this very capability is one which many system builders attempt to build into intelligent machine behavior. To optimize the performance of a task, many people experiment with possible variations. Some of these attempts may be successful, and others not. In any case, errors can be regarded as unsuccessful experiments, with undesirable consequences, that were tried in an attempt to improve performance. When such a deviation has an acceptable result, it is not classified as an error but as a successful variation. A negative judgement is imparted only when the environment providing the context for the deviant action produces unacceptable consequences that cannot be corrected [51].

Perhaps one of the strongest motivations for focusing more attention on intelligent error handling is expressed by Lewis and Norman, who view the system as another participant in a conversation with a human agent. In this dialogue, each conversant attempts to fill in the gaps in the interpretations of what they hear the other say:

“The task is not to find fault and assess blame but rather to get the task done. Failures to understand are commonplace and normal... in a normal conversational setting, both participants assume equal responsibility in understanding. If there is a failure to understand, both take responsibility for repairing the difficulty ... if the system took some of the responsibility ... the result would be a much more cooperative effort¹.”

¹From [41], p. 413

Since there is evidence that errors themselves may often be systematic rather than random in origin [53], it would be helpful to have mechanisms for approaching them systematically, through error classification and categorization. Taxonomies of error that are developed should consider internal cognitive task models and other psychological mechanisms [52].

In accordance with that view, in the remainder of this section we present a summary of the taxonomies that have emerged as a result of the analysis of human error by several researchers in the field of cognitive engineering. There are four major taxonomies that emerge from the literature, 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 [53, 54, 55];
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 [46];
3. A layered mechanistic model suggesting the role of the underlying bases for error as well as their external manifestations [50, 51];
4. A taxonomy of error manifestations, distinguishing between the underlying cause of error and the manifestation [33].

One common distinction found in all the taxonomies examined is that of human planning failures versus execution failures. When a plan fails to achieve its desired outcome, it is generally for one of two reasons:

1. The actions were performed as planned, but the planning was inadequate, or
2. The plan was correct, but the actions dictated were not performed as planned.

This distinction can be applied to divide the general class of "errors" into *slips* and *mistakes*. According to Reason, a *slip* is an error resulting from the failure in the execution stage of an action sequence, whereas a *mistake* is something that originates in the planning phase. A mistake corresponds to a failure in judgement in the selection of a goal or in the specification of the means to achieve it. Slips become apparent after these judgements have been made, when an attempt is made to see if the actions dictated by the decisions run according to plan. Thus, in a sense, mistakes are more dangerous than slips, since while a slip represents a flawed plan execution, a mistake represents an error in intent or an aborted objective. In this thesis, we are particularly interested in slips, since they represent flawed execution.

Reason

One detailed behavioral classification of slips has been established by the work of James Reason. He conducted a study in which 35 volunteers kept a record of their slips of action over a period of two weeks. The results were analyzed to develop a categorization of slip types, that is presented below:

1. **Discrimination failures** (11 percent of total errors). In this class of error, the wrong objects are used in the plan.

- (a) perceptual confusions (physically similar objects)
 - (b) functional confusions (functionally similar objects)
 - (c) spatial confusions (close proximity of objects)
 - (d) temporal confusions (misperception of time and inappropriate actions initiated)
2. **Plan assembly failures** (5 percent of total errors). These errors result from the transposition of elements within the same plan or between two currently active plans, or between an ongoing plan and one that has been performed in the past.
- (a) behavioral spoonerisms (actions in the plan are reversed)
 - (b) confusion between currently active plans
 - (c) confusion between ongoing and stored plans
3. **Test failures** (20 percent of total errors). These errors stem from a failure to properly monitor progress toward the plan goal, and the actions are in service of some goal other than the one that should currently be in focus.
- (a) stop-rule overshoots (actions proceed beyond the intended end-point).
 - (b) stop-rule undershoots (actions are terminated prior to their intended end-point).
 - (c) branching errors (an initial sequence of actions is common to two different plans, and the wrong "route" is taken.)

(d) multiple side-tracking (the actor is diverted from the original intention by a series of minor side-steps).

4. Subroutine failures (18 percent of total errors).

(a) insertions (unwanted actions added to sequence)

(b) omissions (necessary actions left out of sequence)

(c) misordering (correct actions carried out in wrong order)

5. Storage failures (40 percent of total errors).

(a) forget previous actions (e.g. losing one's place)

(b) forget discrete items in plan

(c) revert to earlier plan

(d) forget substance of plan (e.g. "my mind was a blank")

Reason claims that while many slips are due to memory lapses, most of the other slips fall into one of four general categories:

1. Repetition (actions in a sequence are repeated unnecessarily).

2. Wrong object (intended actions taken in relation to wrong objects).

3. Intrusion (unintended actions other than repetitions become incorporated into the sequence).

4. Omission (intended actions left out of the sequence).

Reason also suggests some possible underlying factors that may motivate these flawed behaviors:

- The recency and frequency of a particular routine determines when it might recur uninvited, resulting in intrusion.
- Cues from the environment elicit or trigger deviations and associated action routines. Objects that are frequently handled and locations often visited have strong effects. This is also a type of intrusion.
- The shared features or subsequences of schemata for performing tasks induce confusion.
- Concurrent plans can become intermingled. These are called *blends*.

An examination of the slips categorized by Reason reveals that they represent a mixture of error manifestation (how an error appears contextually and syntactically within a task) and error cause. For example, one of the common slips documented by Reason is that an agent can “forget what was just done.” This is a simple cognitive force at work (forgetfulness). Another slip such as “agent omits something because he has switched to another activity prematurely” really represents a syntactic error class (omission) combined with the hypothesized cause (anticipation).

Elsewhere, Reason attends to the necessity of distinguishing between the “what” and “why” of an error. In [55], he puts forth a framework for classifying errors that takes cognitive factors into account. He suggests that there are *basic error tendencies* (BETS) such as ecological constraints, change-enhancing biases, resource limitations, schema properties, and strategy use that induce systematic error. These BETS are considered in the context of eight *cognitive domains* that represent different stages or operations in human information

processing, such as sensory registration, input selection, temporary memory, long-term memory, recognition processes, judgemental processes, inferential processes, and action control. The result of this tabulation of the basic error tendencies across the cognitive domains is a set of *primary error groupings* such as false sensations, attentional failures, memory lapses, unintended words and actions, recognition failures, inaccurate and blocked recall, errors of judgement, and reasoning errors. Finally *situational factors* such as age give rise to probabilistic predictions of these errors.

Norman

Another error taxonomy that was developed along similar lines to Reason's taxonomy of "actions not as planned" is that of Donald Norman. In [46] Norman discusses human error at two different levels: (1) the sources of the error, and (2) the types of error. He adopts a view of human action that is based on the selection, activation, and triggering of multiple schemas that guide human behavior. Thus, the three major sources of human error are closely tied to this representation:

1. Slips can result from an error during the formation of intention;
2. Slips can result from the faulty activation of schemas;
3. Slips can result from the faulty triggering of active schemas.

The first category, errors resulting during intention formation, is broken down into two classes:

1. Mode errors: An operation is performed that is appropriate for a mode that is not current.
2. Description errors: This type of error is represented by an incomplete or incorrect specification of the action that is selected to achieve the intention. Usually the erroneous act has many characteristics of the appropriate one. Often a user derives an action based on their understanding of a similar one, but that input is insufficient as a description for what was needed/expected. This error class subsumes errors such as incorrect object selection, confusion between objects, etc.

The next category of errors is caused by the inappropriate activation of an unintended schema. These are further subdivided into those errors resulting from unintentional schema activation and those resulting from a schema becoming inactive before the task it is governing has been completed. Examples:

1. Unintentional activation
 - (a) Capture errors: A frequently performed sequence may capture control from the originally activated sequence that may be performed less frequently.
 - (b) Data-driven activation: external events can cause schema activation;
 - (c) Associative activation: currently activated schemas activate associated schemas.
2. Loss of activation

- (a) Forgetting an intention;
- (b) Misordering action components in a sequence;
- (c) Skipping steps in an action sequence;
- (d) Repeating steps in an action sequence.

The third category of errors is caused by faulty triggering of active schemas. This category is divided further into two classes: (1) those errors that result from a "false triggering" of inappropriate schemas and (2) those that result from a failure to trigger the appropriate schema:

1. False triggering

- (a) Spoonerisms: reversal of action components;
- (b) Blends: combinations of components from two competing schemas;
- (c) Thoughts leading to actions: triggering of schemas meant only to be thought, not to govern action;
- (d) Premature triggering;

2. Failure to trigger

- (a) Preemption by competing schemas;
- (b) Insufficient activation due to forgetfulness;
- (c) Bad specification of triggering conditions or insufficient match between existing conditions and required conditions.

In addition to providing an error classification and causal theory of error, in [41], Lewis and Norman claim that such a model of human error should be an

important component of user-interface design. They suggest that interaction with a computer system can be thought of a cooperative endeavor. They discourage the use of the word "error," since the action performed by the user may not be wrong; rather, the system may have inadequate mechanisms to interpret this action. They also note that entire classes of errors can be avoided through appropriate representation. For example, when printing a file, potential errors such as the specification of a nonexistent file or the mistyping of a file name can be avoided by representing available files as icons, and having the user choose graphically. This type of error avoidance is one of the claims of menu-based interfaces.

A system that has been built with the anticipation of errors must have mechanisms for the handling of these deviations. Lewis and Norman discuss different response strategies that a cooperative system can take upon encountering an error:

- **Gag.** A forcing function prevents any further action until the error is corrected.
- **Warn.** The user is notified of a discrepancy but the user can decide what to do.
- **Do nothing.** The system makes no response upon the initiation of an erroneous action, and that action has no side effects.
- **Self-correct.** In this approach, the system attempts to determine what the user intended, and performed whatever modifications are necessary. DWIM² is best example of this approach. Without interaction with the

²DWIM stands for Do What I Mean, and is documented in [66].

user, this approach is subject to potential problems such as overgeneralization of fixes, possibly wrong corrections, etc.

- **Teach me.** The user is asked to define the terms that the system does not recognize. This approach assumes a well-established language for dialogue and definition specification.
- **Let's talk about it.** This paradigm assumes a shared responsibility between the user and the system to explore the problem and come up with a solution: e.g. LISP throws you into the debugger with suggestions.

In general, Lewis and Norman advocate an approach in which the system is considered to be an intelligent agent that cooperates with the human agents who interact with it:

“Think of the input not as an error, but as the user's first iteration toward the goal. The system should do its best to help.... If we think of interaction with a computer as a cooperative endeavor we see that each side has certain talents. The person is good at setting goals and constructing intentions. The computer is great on the details. Let the two work together, with the person in charge. And let the computer go out of its way to make things easy for the user, to make corrections easy, to go that extra step toward understanding that makes conversation with an intelligent colleague so fruitful.³”

³From [41], p. 432.

Rasmussen

The work described by Rasmussen [50, 51] is based on the analysis of industrial accidents and the subsequent investigation of human error. Rasmussen developed a multi-layered taxonomy of human error that was constructed by tracing back the causal chain of events leading to industrial "incidents." There are seven elements to this taxonomy:

1. The *external mode of malfunction* refers to the outward manifestation of the error. This level pertains to the features of the "man-machine mismatch," or in other words, refers to the detection of a mismatch of actual behavior and desirable behavior. Examples: omission of acts in procedural sequences, acts on wrong components, reversals in a sequence, out of order actions, wrong timing, etc.;
2. Rasmussen then lays out some *mechanisms of human malfunction*, that are possible cognitive processes that can give rise to the *external mode of malfunction*. These mechanisms include: stereotype fixation, familiar short-cut, forgetfulness, a condition or side effect that was not considered, etc.;
3. The above cognitive mechanisms can be triggered by *causes of human malfunction*, that include external events such as distraction, or by excessive task demand, intrinsic human variability, etc.;
4. Rasmussen's model also includes *factors affecting performance* such as subjective goals, affect, resources, etc.

5. In addition, other *situation factors* such as task characteristics, work time, and physical environment, can influence performance;
6. Factors regarding the *personnel* dimension of the task such as equipment design and testing, administration, and management may also influence task performance;
7. Finally, the dimension of *internal human malfunction* is used to describe the final action taken by the human that has been influenced by the *causes of human malfunction, mechanisms of human malfunction* etc. It is this action (or lack thereof) that result in the *external mode of malfunction*. Examples are pushing buttons or levers, issuing commands, etc.

This attempt to describe and understand the cognitive and environment factors that can lead to error is important in order for proper interpretation and correction. Elsewhere, Reason emphasizes that the difficulties in classifying errors stem from the implicit underlying causality which is hard to ascertain, rather than from the observable features. On the surface, human errors can be classified fairly easily into a limited number of syntactic categories. But knowing what behavioral category an error falls into does not necessarily reveal anything about what caused the error. In fact, errors that demonstrate the same syntactic features can arise from very different causes, while the same underlying cognitive mechanism can be responsible for observed procedural deviations that fall into different syntactic classes [55].

Hollnagel

Hollnagel reviews the existing work on error classifications by Reason, Rasmussen, and Norman, and concentrates on what would be needed to actually implement an error recognition system. He makes a strong distinction between the underlying cause of an error (referred to as the *genotype*) and its manifestations (referred to as *phenotypes*), and develops a taxonomy of error *phenotypes*. Hollnagel discusses six simple phenotypes:

1. Omission;
2. Intrusion;
3. Replacement;
4. Reversal;
5. Repetition;
6. Insertion.

This categorization does not account for inaccurate performance, i.e. when the right action is performed with the wrong result, or in the wrong fashion. Hollnagel also discusses *complex phenotypes*, that are combinations of simple phenotypes. Examples are: (1) sidetracking: the insertion of a sequence of actions (complex insertion), and (2) three types of capture: (a) a stronger action sequence replaces a current one, (b) branching — an action segment of sequence continues along the wrong path, and (c) overshoot: a continuation beyond the end point of a task.

Hollnagel does not address the topic of genotypes in any detail, but makes some suggestions; e.g. possible genotypes for omission include forgetfulness,

interruption, or an attempt to achieve a short-cut to accomplishing the task goal. In general, it is difficult to relate the behavioral mismatches to psychological mechanisms in any verifiable fashion. Rasmussen claims that "given the knowledge of the task and the particular external error mode found, it is in general possible only for the more familiar routine tasks to judge the internal mode of malfunction from a case story."⁴ Otherwise, he claims, interviews and discussions with the performer are required to determine the cognitive basis of the malfunction.

3.3 A New Theory of Human Procedural Deviations

While Hollnagel's work represents a start, work that has been done on error analysis focuses very little on actual implementation of error detectors or correctors. Hollnagel points out that while error taxonomies cannot be implemented as purely data-driven or bottom up, operationalizing the taxonomy makes the true phenotypes emerge. Sometimes only the consequences of an erroneous action can flag the error; other times errors may be syntactically detected. Little work has also been done to establish mappings from genotypes to phenotypes, although this may be a very difficult if not-impossible problem, as already mentioned. In any case, there is great need to address the detection and correction problem in more detail.

In this thesis, we adopt a novel interpretation of some of the same deviations from planned courses of action that other researchers have categorized as *slips* and *mistakes*. We extract from this research the syntactic categories that

⁴From [51], p. 26.

have been identified, and refer to them as *exceptions*. All exceptions appear as discrepancies at initial detection, but a sophisticated analysis may show that many of these exceptions are not, in fact, errors. When considering such syntactic exceptions in the context of a planning system that a) uses an inherently incomplete model, and b) employs intelligent human agents to perform plan execution, there are three possible interpretations of such discrepancies:

1. The exception represents an action that is slightly "off the mark," but with a simple model of error patterns, this *slip* can be detected and fixed.
2. The exception represents an action that is based on an error in intent, and may simply be wrong and not in service of the goal at hand (mistake).
3. The exception represents a perfectly reasonable thing to do. It was not planned for because of incomplete or inadequate domain knowledge.

It is this third possible interpretation that represents a shift in perspective from the views that have been adopted by other researchers who have looked at exceptions. Adding this perspective as a possible way to interpret exceptional behavior can provide a significant extension to the capabilities of intelligent assistant systems when encountering discrepancies between a plan and its observed execution.

In accordance with this new perspective, in our research we hypothesize constructive explanations for exceptional actions. The explanations are based on: (1) a theory of intentional human procedural behavior, and (2) an expectation that the domain is incomplete and/or incorrect. The former is represented by a set of *plausible rationales* that were designed as possible explanations of

the primary syntactic deviations identified by these studies, such as repetition, intrusion, omission, etc. Examples of intentions that are hypothesized to underly these behaviors are: knowledge of alternatives, attempts at shortcuts, intentional reordering, new actions, etc. The rationales are presented in detail in Section 6.3. In addition, techniques are invoked to constructively interpret unusual actions in the context of an assumed imperfect domain, which is also discussed in Section 6.3.

In later chapters, we will discuss how our theory is used to construct explanations of exceptions that restore the plan to a consistent state. In addition, although we emphasize purposeful explanations of exceptions, we also consider that an exception may have a "non-purposeful" interpretation, as discussed above. We propose to make existing planners more robust by incorporating mechanisms to detect and treat exceptional behaviors as errors if a constructive semantic basis for the deviations cannot be established.

CHAPTER 4

THE SPANDEX APPROACH

In this chapter, we present a formal model of plan execution and the exception problem. We discuss possible approaches towards handling exceptions that differ in explanatory power as well as with respect to computational needs. We present the architecture that we have implemented as a general solution to the exception problem. The chapter concludes with a comparison of our approach to explanation-based learning.

4.1 Planning and Execution

In this section, we define terminology that is relevant to plan execution and exception handling. Our definition of the planning task of a hierarchical nonlinear planner is as follows¹:

Given:

- s_i : an initial world state;

¹This definition is similar to that proposed in [47].

- A : a set of activities, some of which are primitive (A_p) and others that are decomposable (A_c);
- s_f : a desired final goal state;
- α : a set of available agents;

Determine: a final ordering P of primitive activities in A_p or goal states that, when executed or achieved in an initial state s_i by agents in α , will produce a new state containing the final goal state s_f .

Activities are considered decomposable if they can be elaborated by the planner into sub-activities; primitive activities are associated with actions that are executable by an agent such as a human or tool. The ordering P that represents the final plan for a task is the result of a series of transformations applied to a top-level goal representing the final goal specification s_f . The result of each of these manipulations is represented by a *plan network*, that represents the current version of an evolving plan. A plan network in our system is a strict partial order and consists of the following elements²:

- 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 executable by a tool or human agent)³. Every node (both goal and executable) has a goal

²More detailed descriptions of each of these elements can be found in [39].

³We refrain from discussing here a third type of node that is maintained in our plan networks. These are *structural* nodes and are used to mark the beginning and end of task expansions as well as to control the looping constructs that may be applied to subgoals. Complete details can be found in [38].

that is achievable by the processing of that node. In addition, each node is associated with a set of conditions C that must hold in the before-world of that node before it can be processed by the planner;

- L : a set of temporal links that establish a partial ordering among the nodes;
- W : a set of world states, that 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;
- 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 that must hold in order for a plan network to be consistent:

1. All before-worlds and after-worlds in W are consistent. This implies that the models of the domain represented by the worlds are consistent with respect to constraints in the domain model and the goal being achieved by the node.
2. 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.

3. 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.
4. All protection intervals in I must hold.
5. The set of temporal ordering specifications L must be consistent.

We define the process of *planning* as iterative *transformations* on plan networks. A *final plan* is a plan network that has been fully ordered, and every node is either a *phantom*⁴ or it is a primitive activity node that has been executed⁵. Thus, a plan network represents a class of final plans; many different final plans can result depending on the subsequent choices of elaborations and operations. A new plan network results each time an operation is performed by the plan network maintenance system (PNMS [7]), such as node expansion, node execution, node ordering, or protection interval establishment. We will refer to node execution as *execution events* and to all other PNMS operations as *elaboration events*.

The current hierarchical representation of a plan is called the *plan outline*⁶. The plan outline subsumes the current plan network, which is constituted by the leaf nodes. The complete *plan history* is a lattice containing all intermediate plan outlines ordered within planning time, where the distinguished upper

⁴A phantom node [74] is a goal node that has been determined to be true at its position in the plan without further expansion and execution.

⁵Since planning is interleaved with execution, we adopt a broad definition of planning that subsumes execution.

⁶We adopt this term because of the natural analogy of a task breakdown into levels of subtasks with the multi-leveled outline of a document.

bound is the eventual final plan⁷. The relation is a partial order since backtracking may be allowed. The plan history maintains a record of all planning actions performed. As we shall see, the maintenance of the plan outline and plan history is crucial in being able to construct meaningful explanations for exceptional behavior.

We define the concept of a *plan wedge*⁸ in order to be able to refer to the portion of a plan outline that represents the abstractions and subsequent refinements that introduced a 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 composed of a set of nodes defined as follows:

Given the following recursive functions:

1. *node-ancestors*(*n*): returns a set of nodes containing (a) the parent node that, when expanded once, produced an expansion containing *n*, and (b) all nodes in *node-ancestors*(parent node);
2. *node-descendants*(*n*): returns a set of nodes consisting of (a) all children nodes in a single expansion of *n*, and (b) all *node-descendants*(child) for each of these children nodes,

⁷Thus, each plan outline represents a *planning level*, using the term of Wilkins [74]. The final plan outline can be considered to be the final planning level.

⁸Our definition is similar to the definition of a wedge used by Wilkins [74] and produces the semantic equivalent.

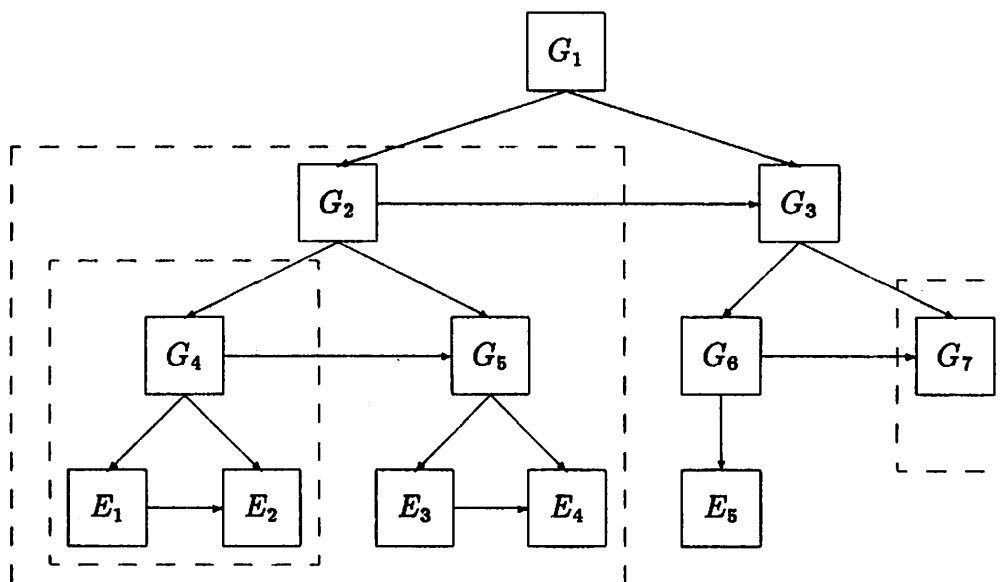


Figure 11: Wedges of a plan outline

a *plan wedge* consists of a distinguished node in $\text{node-ancestors}(n)$ that is chosen as the *apex* of the wedge, and the set of nodes in $\text{node-descendants}(\text{apex})$.

For illustration, Figure 11 shows a plan outline for achieving goal G_1 . The wedge defined by an apex of G_2 consists of nodes $\{G_2, G_4, G_5, E_1, E_2, E_3, E_4\}$, while the wedge defined by using G_4 as the apex consists of $\{G_4, E_1, E_2\}$. Choosing G_7 as a apex defines a single-node wedge $\{G_7\}$.

Plan networks can also be described in terms of what portions of them have been executed. 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 *ready nodes* that are eligible for processing. A node is in this set if and only if:

- all of the conditions specified in the before-world of the node are satisfied in that world;
- all of the node's necessary predecessors are complete and awaiting successors. Specifically, any node that is strictly ordered before the node in question must have a processing status of *complete* and must have at least one (strict) successor that is *not complete*⁹.

The depth-first, left-to-right processing of the plan elements that is implied by the preceding definition is the strategy currently used by our planner; the method of prioritizing possible expansions is a modular parameter in the system and could easily be modified. However, we prefer to elaborate earlier parts of the plan network before later elements since this strategy provides a solution to the problem of *hierarchical promiscuity* as described in [74]¹⁰ In addition, since we are working in an interactive setting, we often need to drive down the earlier sections of the plan in order to obtain information from the user that is required for subsequent plan steps.

⁹A more complete definition of "ready nodes" that defines "conditions" and "necessary predecessors" in detail can be found in [39].

¹⁰Briefly, this term refers to complications that can arise when later sections of a plan are expanded before all the earlier detail has been introduced. When more detailed information is introduced at an earlier part of the plan after later plan elements have been elaborated, problems may result in the later portions of the plan that can require significant overhead to correct.

For example, referring back to Figure 11, if executable nodes E_1 through E_3 have been executed, the set of ready nodes is defined as $\{E_5\}$. After E_5 has been executed the set of ready nodes becomes $\{G_7\}$. The set of ready nodes contains multiple members when partial ordering exists in the network. The network used in this example is fully ordered, so therefore a single ready node exists at any given time.

When multiple ready nodes exist, the plan network node processing algorithm used by our planner currently favors the processing of an eligible goal node over an eligible executable node. The conceptual justification for this preference is to perform as much expansion as possible in sub-parts of the network preceding executable actions in order to provide additional constraints on the subsequent execution.

We refer to an *elaboration cycle* as a process during which a number of PNMS operations are applied to the current plan network so as to expand all ready goal nodes and process all ready structural nodes. When an cycle of elaboration has been completed, a set of *expected action nodes* can be identified in the plan network. These are the executable nodes that are now ready according to the criteria above. One of these *expected* nodes is selected for processing by the planner, and is referred to as the *suggested* action node.

4.2 Exceptions

In general, a run-time failure for a planning system can be defined as an input that is not specified or predicted by the current plan¹¹. Drew McDermott stated in a recent workshop on planning:

"Given a planner, there is a natural model of execution failure. The current plan can specify the proper reaction to a wide range of sensory input, but it can also classify certain input as outside the tolerable envelope. When an input in this range is detected, the current plan just doesn't specify what to do. An event of this type is an execution failure.¹²"

In addition to the type of failure described here that results from an unexpected input, we also define run-time planning failures to include unexpected results that are detected after a planning action. Our overall approach is to provide a general mechanism for handling these unexpected occurrences that are detected during interleaved planning and execution. To use McDermott's terms, we attempt to expand the limits of the "tolerable envelope."

We now describe our particular view of such failures, which we refer to as *exceptions*. We adopt this change in terminology to avoid the negative connotations associated with the words "failure" or "error," since as previ-

¹¹Planning failures can also encompass the inability of a planner to produce a successful plan; this is regarded as an overall system failure and is not included in our considerations here.

¹²From McDermott's summary of the panel on Planning and Execution at the DARPA workshop on Planning [18], p. 128.

ously described, a deviation from the plan may in fact be a semantically valid occurrence.

4.2.1 Detection of Exceptions at Run Time

Given a plan network p with an identified set of expected action nodes, and a distinguished suggested action node, an action a may be performed. The action a may be an elaboration action (e.g.; plan expansion, ordering, etc.) or an execution event. In the case of an execution event, a is an activity descriptor consisting of an *operator* and *parameters*. The operator name is assumed to uniquely identify the activity and the parameters refer to domain knowledge base objects that 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*. An execution event a also has an associated resulting world state w . The world state w is a predicate calculus representation of a fact in the domain model.

The event a is processed as follows:

- If a is an execution event: the specifications of a and w are checked against the action and goal states associated with the suggested action node. If a match is found, the suggested 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 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. Resulting inconsistencies are calculated.

- If a is an elaboration event, the new plan network and context are examined for inconsistencies.

Two classes of inconsistencies are detected by our system. The first class is detected by the execution monitor during interaction with the user and is calculated by comparing an action taken by a human agent with the expectations of the planner. The second class of inconsistencies pertains to the state of the current plan network. As a result of an action (either an elaboration event or an execution event), the plan network may now be inconsistent. Specifically, one or more of the plan network consistency criteria defined in Section 4.1 may have been violated. Both classes of exceptions are discussed in more detail in Section 6.1. The union of these two classes of inconsistencies results in a taxonomy that is broader than other established categorizations of plan flaws [4, 74]. A detailed comparison is presented in Section 6.1.3.

4.2.2 The Exception Problem

At this point, having defined what is meant by a consistent plan and an exception, we can now pose the problem to the SPANDEX exception handling system as follows:

Given:

- p : a partially executed plan network;
- (a, w) : an event-result token (where w is optional, depending on whether a is an elaboration or execution event);
- X : a set of calculated inconsistencies;

Compute: a new successor plan network p' that meets the following criteria:

- The set of executed nodes in p' include all executed nodes in p ;
- p' contains a node representing the exceptional event when a is an execution event;
- p' is consistent,
- p' has the same top-level goal as p .

Resume planning and execution.

The high-level system loop is summarized in Figure 12. The terms *final-plan* and *perform-elaboration-action* refer to the definitions of *final plan* and *elaboration events* presented in Section 4.1.

Note that the procedure followed in the initial portion of the algorithm presented in Figure 12 (exception detection, insertion of a node representing the exception, and subsequent problem computation) is very similar to that followed by other systems, SIPE in particular. However, 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 correlations between an unexpected event and other elements of the ongoing plan. The remainder of the SPANDEX task is to do exactly that.

```
repeat until (final-plan (current-plan-network))
begin
  repeat
    perform-elaboration-action(current-plan-network);
    inconsistencies := compute-inconsistencies
                      (current-plan-network);
    handle-inconsistencies (inconsistencies);
  until (elaboration cycle has been completed);
  suggested-action := select-action-for-execution
                    (current-plan-network);
  suggested-goal := goal (suggested-action);
  performed-action := monitor-executed-action();
  achieved-goal := goal (performed-action);
  insert-into-network (performed-action, achieved-goal,
                     current-plan-network);
  inconsistencies := compute-inconsistencies
                    (current-plan-network);
  handle-inconsistencies (inconsistencies);
end;
```

Figure 12: An interactive hierarchical planning loop

4.3 Explanation of Exceptions

We are now in the position to discuss possible approaches toward recovery after an exception occurs. The simplest approach could be called "wait and see" and would simply insert the unexpected occurrence at the current execution point in the plan network, and hope that any inconsistencies that result will be removed during subsequent planning and execution. There are two possible optimistic attitudes behind a "wait and see" approach:

1. **Problems are detected; assume they will be undone by later events.** Oftentimes an unexpected occurrence will be accompanied by inconsistencies in the world model or violated conditions needed for later parts of the plan. It is possible that a "white-knight" will show up later to fix the problems¹³, rendering the plan valid and capable of achieving its goal.
2. **No problems are detected; assume nothing will change.** Secondly, it may be the case that although the occurrence was unexpected, there are no violations that are detectable at this point (other than the inconsistency detected by the execution monitor indicating that the accomplished action or achieved state failed to match the suggested action and goal). The attitude may be taken that the suggested event is still to come, and that the unanticipated occurrence is extraneous and harmless.

Neither of these two attitudes is very promising. In the first case, relying on a "white knight" to come and save the day is overly optimistic and often

¹³Chapman used the term "white-knight" to refer to steps in a plan that foil steps that would otherwise become clobberers of necessary conditions [13].

without justification. In the second case, although blind insertion of the event as an extraneous event may seem reasonable initially, later planning actions may reveal problems attributable to this action. With no attempt made to justify the unexpected event upon its occurrence, we may later be in a position where the plan is invalid due to an event that happened much earlier and the intervening computation should have been aborted. Thus, in general, the attitudes driving this blind "wait-and-see" approach are hopeful at best, and may prove disastrous after subsequent computation.

A second approach to recovery after an exception would be to invoke a well-defined set of general replanning actions, such as those used by Wilkins [73, 74] to restore consistency to the plan. These measures would be able to eliminate many outstanding inconsistencies and circumvent a number of subsequent planning problems. However, we find this approach inadequate for two reasons. First, the unexpected event is not justified or explained in any way; possible contributions of the event to the ongoing plan escape consideration. For example, a careful analysis of the role of the unexpected event with the current plan may reveal an action that has occurred out-of-order, possibly due to optional intervening steps or an overconstrained ordering. This explanation might allow a viable shortcut towards the achievement of the high-level goal. One of the reasons why such an explanation would not be found by generic recovery measures like SIPE's is that those measures are inspired by a negative attitude in which it is assumed that the action or state change is most likely undesirable. Thus in such systems only a cursory check is made for serendipitous effects.

A second inadequacy of the systems we looked at is that they do not make use of a substantial model of the domain resources and objects that are manipulated by domain actions. As a result, the recovery measures provided do not address constraint violations pertaining to such objects or exploit mismatched goal similarities¹⁴.

As previously mentioned, SIPE does do some checking for possible serendipitous effects introduced by an unanticipated state change, and responds by removing a wedge of the plan if possible. Thus in effect, SIPE searches backward in planning history to look for candidate wedges. The plan outline can be viewed as a pyramid of nodes, where the top of the pyramid is the highest-level goal and the bottom level of the pyramid contains executable nodes and less abstract goal nodes resulting from plan expansion. For example, Figure 13 depicts a plan outline in which the actions of two executable nodes (E_1 and E_2) have been performed, accomplishing goal G_4 , while the action of E_3 is now expected as part of the current wedge headed by G_2 (and by G_1 at the next higher level of abstraction).

SIPE is limited in its exploration of wedges that may be removed in response to serendipitous effects. The algorithms check to see if goals that are to be achieved at a later point in the plan have become true as a result of a "mother nature" occurrence. Once identified, the ancestor nodes of such goal nodes are examined in order to determine a single candidate wedge that may be removed. This single candidate wedge is excised from the plan if no problems

¹⁴Goal similarities are defined by related objects that may be manipulated by the same goal predicate. The idea is to use the library of goal predicate specifications to determine alternative ways of accomplishing the same general goal. These alternatives would not have been identified initially by the limited algorithms of the planner.

ensue from its absence, otherwise the wedge is put back into the plan. No additional candidate wedges are considered.

SPANDEX extends this algorithm by considering other candidate wedges that are overlooked by SIPE's algorithms. We classify candidate wedges according to the status of the apex node chosen and the temporal relationship of the apex to the current execution point. The following classes of plan wedges are explored when looking for serendipitous effects: *current wedges*: wedges whose leaf nodes include one or more ready actions, *completed wedges*: those wedges for which all leaf nodes are *done*¹⁵, *unstarted wedges*: those wedges that do not contain any executed or ready leaf nodes, and *unexpanded wedges*: wedges represented by unexpanded goal nodes¹⁶.

In response to the limitations of previous approaches that we have just discussed, we adopt an approach that: (1) performs a more extensive search for serendipitous effects of unanticipated occurrences, (2) incorporates a rich domain object model and is prepared to address violations found therein, and (3) uses a theory of human procedural behavior to construct explanations of exceptional events and amend the ongoing plan to restore consistency. In sum, our general approach to this problem is to manipulate available domain knowledge to show how the current network and domain knowledge base can

¹⁵A node in one of our plan networks has a *status* field that changes during the evolution of the plan. When a node is first inserted into a plan network, it is assigned a status of *unseen*, if it is a goal node that has been expanded, it has a status of *expanded*, and if it is an executable node that has been executed, its status gets set to *done*. Complete details can be found in [40].

¹⁶In a sense, unexpanded wedges do not yet actually exist, but since every leaf-level goal node is a potential apex for an expansion that is not yet in place, heuristics can be used to explore possible expansions.

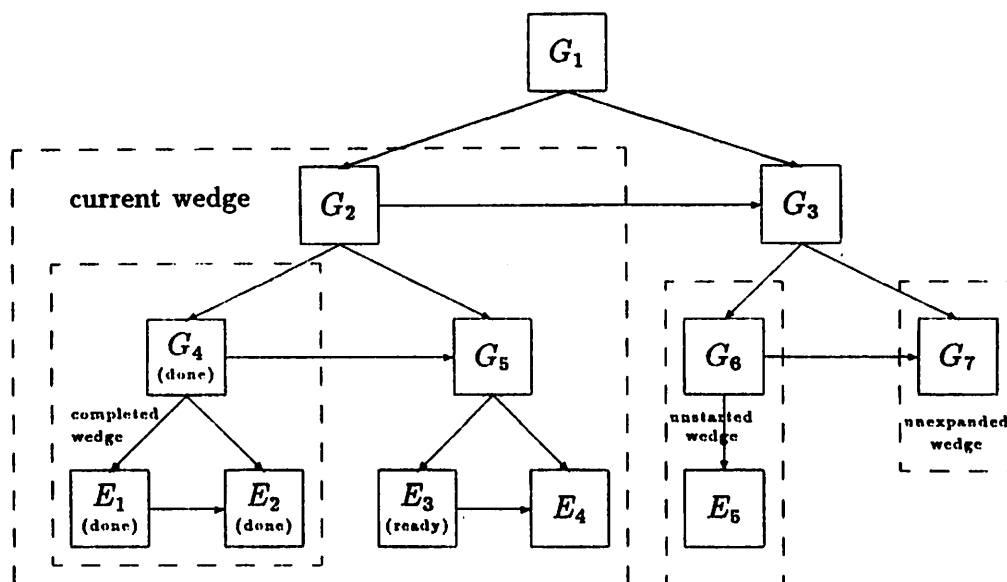


Figure 13: A plan outline as a pyramid of wedges

be transformed to eliminate inconsistencies resulting from the occurrence of the unexpected event.

4.4 A Review of the System Architecture

In this section we present a more detailed version of the system architecture first discussed in Chapter 1. Our general approach to this problem is to manipulate available domain knowledge to generate plausible explanations that indicate how the current network and domain can be transformed to eliminate inconsistencies resulting from the occurrence of an unexpected action and/or

state. The expanded architecture of the system we have designed to handle this problem is shown in Figure 14.

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 5) using plan repair methods based on those of [74].

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 5). The rationales are based on the model of human procedural behavior described in Chapter 3. This model identifies common procedural deviations such as performing actions out-of-order, skipping steps, reversing steps, or exchanging resources with similar functions [54, 46]. We maintain that these aberrations can be intentional rather than random. Although such exceptions may initially appear to be errors, they can in fact represent valid actions. The mechanisms of the exception analyst that use these rationales to produce explanations demonstrating the validity of an exceptional action are described in Section 6.3. In addition, other explanations are considered that account for the possibility of an imperfect or incomplete domain description that is uncovered by a normal planning action.

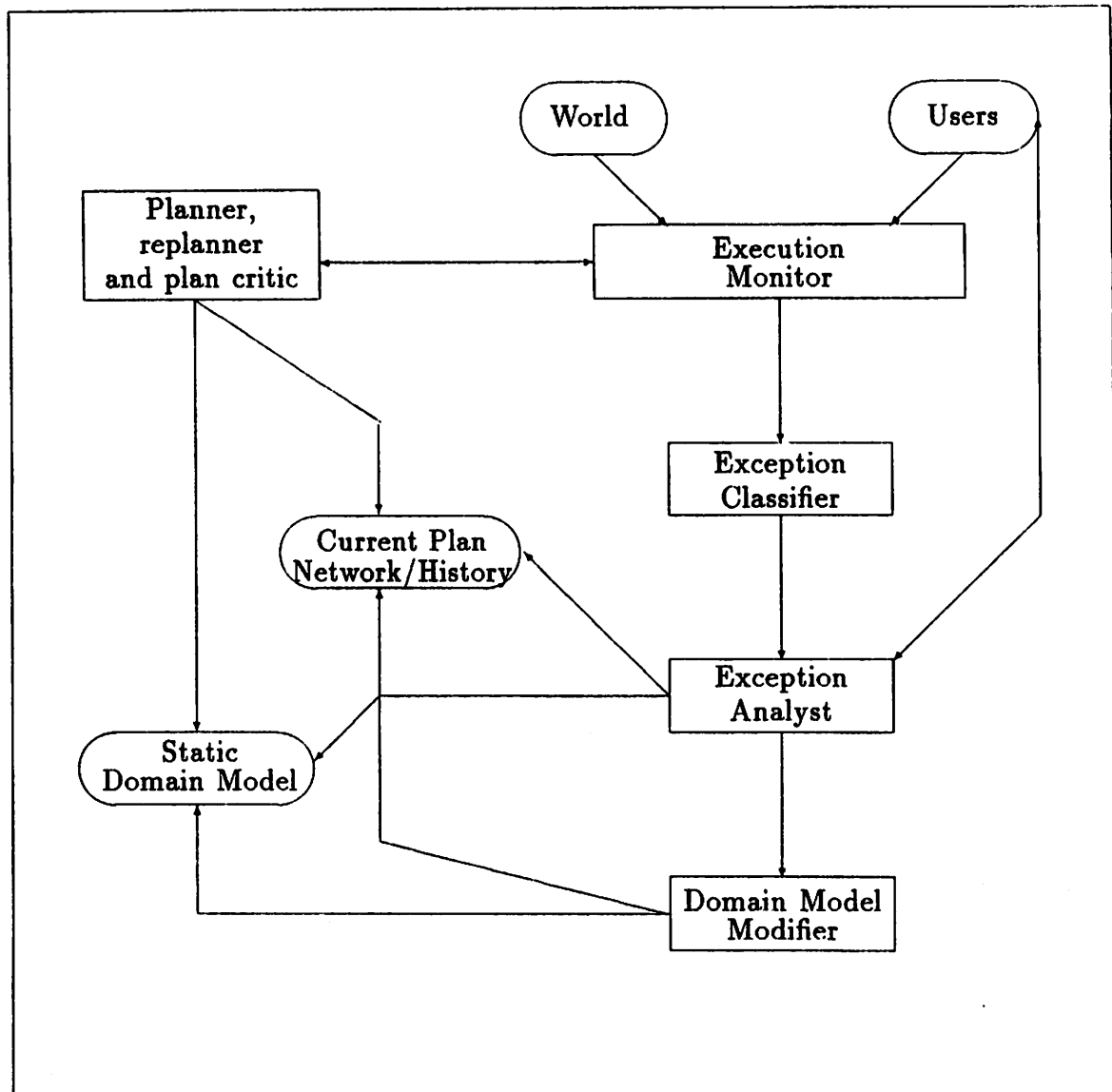


Figure 14: An architecture for planning and exception handling

One of the explanations is selected by the user through a dialogue conducted by the system. The selected explanation specifies a set of changes or *amendments* to be made to either the plan network for this particular instance or to the permanent plan library¹⁷.

The domain model changes that are prescribed by the selected explanation are passed to the *domain model modifier* for implementation. Thus, a successful handling of an exception can result in a refinement and extension of the original knowledge base. The static domain objects and activities may be augmented with knowledge about the exception and thus the system is able to handle future similar exceptions.

We now revisit the system loop originally presented in Figure 12, and expand the step that handles the computed inconsistencies (handle-inconsistencies (inconsistencies)) with high-level details of the approach just outlined (see Figure 15). The variable *max-num-expl-cutoff* refers to a parameter that limits the number of explanations produced and presented to the user during a single cycle. Thus a preset number of explanations are produced during each cycle until either one is chosen by the user or there are no more possible explanations.

¹⁷A proposed extension to this architecture when dealing with multiple agents is to use a paradigm of negotiation [25, 57] to establish a consensus among them regarding the explanation and proposed plan modifications. The negotiation module would identify the agents who are affected by the exception and use the information provided by the exception analyst to conduct a dialogue. The output of this dialogue would again be a selected explanation, along with approved changes. An additional way to extend the architecture would be to generalize the verified explanation, using taxonomic information in the domain model.

```
repeat until (final (current-plan-network))
begin
  repeat
    perform-planning-or-executable-action;
    inconsistencies := compute-inconsistencies
                      (current-plan-network);

  if inconsistencies do
    repeat until ((chosen-explanation and chosen-amendments)
                 or
                 (null explanations)) do
      begin
        explanations := instantiate-rationales
                      (rationale-types
                       inconsistencies,
                       current-plan-network,
                       max-num-expl-cutoff);

        chosen-explanation := user-choose
                             (explanations);

        if chosen-explanation then
          possible-amendments
            := amendments (chosen-explanation)
          chosen-amendments := user-choose (amendments);
        end;

        if (null explanations) process-error;
        else implement-amendments (amendments,
                                   current-plan-network,
                                   domain-model);

        inconsistencies := compute-inconsistencies
                          (current-plan-network);

        if inconsistencies then replan();
      end;
    end;
  end;
end;
```

Figure 15: An interactive hierarchical planning loop (extended)

4.5 An Explanation-Based Learning Perspective

Now that the exception problem has been described in detail along with a sketch of the solution that we provide, we can reconsider the paradigm of explanation-based learning and examine parallels that can be identified in the SPANDEX approach. There are numerous similarities. In a sense, the methodology we propose in SPANDEX is a type of explanation-based learning. We can map SPANDEX into the general explanation-based learning paradigm in the following manner:

1. **Goal concept.** In EBL, the goal concept is defined as a goal state, which is an incomplete world state specification. In SPANDEX the goal concept is also a goal state, but it has a slightly different role in the explanation process (see the section below that discusses the **explanation**).
2. **Domain theory.** In EBL, the domain theory consists of objects with properties, inference rules for inferring more about object relationships and properties, and problem-solving operators (activities). In SPANDEX the domain theory consists of the same types of entities. However, our inference rules are not represented explicitly as rules; rather such rules are embedded in actual object definitions in the object hierarchy. SPANDEX also has access to knowledge about agents, motivations, etc. Another difference is that the representation used by SPANDEX provides a richer constraint language to be used in schema definitions than the limited constraint vocabulary that seems to be available in the EBL systems previously reviewed.

3. **Single Training Example.** In EBL, the single training example is a sequence of operators (with some operators potentially missing) that represents the observed problem solving behavior of an agent. Similarly, in SPANDEX, the example is also an observed sequence of primitive actions that have been performed. Specifically, an example in SPANDEX is an ordered string of action instantiation tokens which thus far have been either executed by the planner or performed by the user, including the exceptional action as the last token. In the standard EBL approach, the observed sequence is given as a single input representing a complete plan. In contrast, SPANDEX is given only a partial sequence of primitive operators, as an indication of the incremental nature in which SPANDEX observes and attempts to understand the problem-solving behavior of an agent.
4. **Explanation.** In EBL, the explanation is an "operator sequence that solves a problem, together with an annotation that captures how the effects of one operator match the preconditions of another." An explanation includes expansion matching choices, and is meant to justify how the precondition of each operator is achieved, and how the goal is achieved. Operators and states that don't "causally" support the goal are eliminated. Thus, in EBL, an explanation is really a *proof* of how the observed operator sequence solves the initial problem. In SPANDEX, at the point where an exception is encountered, we do not yet necessarily know the complete sequence of primitive steps that define the final plan. Thus we cannot absolutely prove that the goal will actually be met. In SPANDEX, we use explanation to show that the observed partial action

sequence (including the exception) *can be part of at least one possible plan network that is consistent with achieving the goal concept*. The explanation may require that alterations be made to the current plan network, however.

In SPANDEX, an explanation is constructed within the context of a current plan network. This plan network is an intermediate plan specification containing goal nodes at different levels of abstraction and set of identified expected action nodes. A SPANDEX explanation is a new tree-like plan network that contains at the leaf level the token string of observed and performed actions (the initial training example). The inner nodes of the plan-network tree are nodes that are the more abstract specifications of the primitive plan steps, as well as unexpanded goal nodes. The root node of the explanation structure is the goal concept. This explanation structure contains causal links and is annotated with the justifications of the role of the unexpected action in the evolving plan. The previous state of the plan network may be referred to in the annotations, in order to express justifications such as which step in the previous plan network was replaced, what previous steps are no longer necessary due to the unexpected action, etc.

5. **Result.** The target result for both the EBL and SPANDEX approaches is a general schema of which the instance is just an example. The primary focus in the EBL work is on techniques for generalization, while our initial focus in SPANDEX is to develop a set of techniques for constructing different types of explanations. We do, however, recognize the need

for generalization techniques in order to learn through the handling of exceptions, and make use of existing methods such as those used by [8].

In summary, the goals we are trying to achieve in SPANDEX can be mapped fairly closely into an explanation-based learning perspective. However, we have identified the following as novel aspects to our problem and approach:

1. Our planning model is one of incremental planning and execution. Planning and execution are interleaved.
2. Our planning system is interactive, and this interface design imposes a cooperative framework upon the users and the system.
3. We assume an incomplete domain theory, and suggest the paradigm of negotiation to extend the theory. Negotiation supplements the more bounded task of *reorganizing* existing knowledge through automated explanation construction (shared in common with EBL).
4. We introduce human agents as an important component of the planning model. Motivations for unusual agent behavior are identified. Negotiation must take place between agents (including the system) regarding an explanation.
5. We rely heavily on an abstraction/generalization hierarchy of activities and objects to construct the explanation of an exception. EBL usually does not use this type of information explicitly during explanation construction, although similar reasoning plays a role during the generalization process.

CHAPTER 5

AN INTERACTIVE PLANNER

In this chapter we describe the implementation of our interactive planner with particular attention to aspects relevant to the exception handling problem. Details about the plan representation and planning algorithms used by the POLYMER planner are specified in Section 5.1, and Section 5.2 describes modifications that were made to the underlying representation used by the implementation in order to accommodate the handling of exceptional events. Section 5.3 describes the process by which activities are selected to achieve goals posted by the planner. Finally, Section 5.4 discusses a general facility that was developed to intelligently guide the choices of resources and other required information that must be made by human agents while engaged in interactive planning. This facility was provided in an attempt to minimize the number of exceptions that could occur by encouraging the user to make binding choices that are compatible with the constraints of the domain and the current plan context. The basis for the computation of these *suggestions* is a heuristic exploitation of relevant constraints in the domain model.

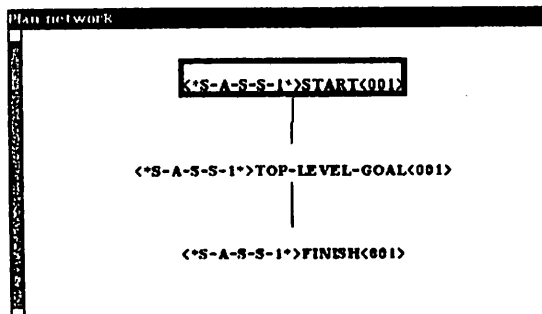


Figure 16: POLYMER plan network before expansion

5.1 POLYMER Planning Details

POLYMER [39, 40] is a knowledge-based system that works interactively with human agents to develop and execute a detailed plan of action to achieve a high-level goal. The initial situation presented to the planner consists of a desired final goal state. This state is represented in POLYMER by a initial plan network containing a single goal node representing the desired state, preceded and followed by structural nodes to delineate the beginning and end of the task (see Figure 16). The planning algorithms then access and manipulate entities in a library of domain knowledge containing descriptions of activities, objects, and agents to expand the initial top-level goal into a final plan. In the remainder of this section, we briefly describe each of these knowledge base entities.

CHAPTER 5

AN INTERACTIVE PLANNER

In this chapter we describe the implementation of our interactive planner with particular attention to aspects relevant to the exception handling problem. Details about the plan representation and planning algorithms used by the POLYMER planner are specified in Section 5.1, and Section 5.2 describes modifications that were made to the underlying representation used by the implementation in order to accommodate the handling of exceptional events. Section 5.3 describes the process by which activities are selected to achieve goals posted by the planner. Finally, Section 5.4 discusses a general facility that was developed to intelligently guide the choices of resources and other required information that must be made by human agents while engaged in interactive planning. This facility was provided in an attempt to minimize the number of exceptions that could occur by encouraging the user to make binding choices that are compatible with the constraints of the domain and the current plan context. The basis for the computation of these *suggestions* is a heuristic exploitation of relevant constraints in the domain model.

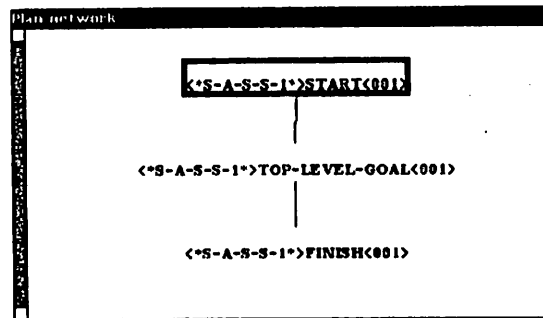


Figure 16: POLYMER plan network before expansion

5.1 POLYMER Planning Details

POLYMER [39, 40] is a knowledge-based system that works interactively with human agents to develop and execute a detailed plan of action to achieve a high-level goal. The initial situation presented to the planner consists of a desired final goal state. This state is represented in POLYMER by a initial plan network containing a single goal node representing the desired state, preceded and followed by structural nodes to delineate the beginning and end of the task (see Figure 16). The planning algorithms then access and manipulate entities in a library of domain knowledge containing descriptions of activities, objects, and agents to expand the initial top-level goal into a final plan. In the remainder of this section, we briefly describe each of these knowledge base entities.

Activities

Activities are retrieved to accomplish a desired state of the application domain. Every intermediate and final goal state that must be achieved during the processing of a task is expressed in predicate calculus as a well-formed formula (wff), and is contained in a goal node of the current plan outline. An activity is described in terms of its goal, input-goal-vars, precondition, decomposition, effects, agent, and various types of constraints. Every activity must specify a goal and a responsible agent, but the other clauses are optional. An example activity that models the planning and staging of an academic symposium is shown in Figure 17. The goal of this activity is to stage a successful meeting. The way to do this activity is generally to plan the symposium, then conduct it and process the necessary paperwork after the symposium has concluded. The constraints state that the meeting must be of type symposium, and the responsible agent for the activity must be the elected conference chair for the symposium. The steps involved in planning the symposium enable the symposium to be held, and the symposium generally takes place before any postprocessing paperwork is done. This particular activity does not have a restricting precondition, nor any other effects other than those explicitly described by the goal.

An activity can be selected to achieve a goal if its goal clause unifies with the goal state that needs to be achieved, and if its precondition is met. The precondition must hold in the model of the domain state as projected at the point immediately prior to the goal node in focus. We subsequently will refer to this state as the *before-world* of the goal node (as originally defined in Section 4.1). Preconditions in POLYMER are treated as static rather than

```
{ACTIVITY stage.a.symposium
*agents: ?chairperson
*comment: "Top level activity to set up and hold a symposium."
*goal: completion.status(?meeting, successful)
*input-goal-vars: ?meeting
*decomposition:
    (symposium-is-planned completion.status(?meeting,planned))
    (symposium-took-place completion.status(?meeting, held))
    (symposium-postprocessing-done
        post.conference.paperwork(?meeting, done))
*plan-rationale: enables(symposium-is-planned, symposium-took-place)
*control: (BEFORE symposium-took-place, symposium-postprocessing-done)
*constraints: elected.conference.chair(?meeting,?chairperson)
             member(?chairperson, conference.chair)
             member(?meeting, symposium)
}
```

Figure 17: The stage.a.symposium activity

dynamic qualifications on activity applicability, that is, no attempt is made to achieve a precondition if it is not true¹. Thus, preconditions act as a filtering mechanism during activity selection. An activity's main purpose is described in its goal, but other changes that may result can be described as the effects of the activity. These effects are regarded as state changes often associated with the completion of an activity, but they are not interpreted as rigidly as postconditions. In other words, the states described in the effects clause of the activity do not necessarily have to be true after the activity has completed in order to consider the execution of the activity to be successful.

The decomposition clause describes the substeps that make up the activity. Causal and temporal relations among these steps are expressed as plan-rationale and control constraints, respectively. The causal relations specified in the plan rationale clause of an activity imply temporal constraints; such temporal constraints are viewed as being more rigid than the more simple temporal constraints specified in the control clause of the activity. The reason for this is that the temporal constraints deduced from causal constraints have a specified semantic basis. For example, a plan rationale specification may state:

"the banquet food has arrived for the conference"

enables

"the conference banquet is held"

¹This interpretation of preconditions is identical to that used by SIPE [72]; other planning systems have imposed different interpretations of preconditions. For example, the GRAPPLE system [34] allows the knowledge base designer to delineate between the preconditions that should be treated statically and those that should be treated dynamically (i.e., efforts should be made to achieve them if they are not true).

This specification implies the temporal ordering that the meal must be ordered before it can be consumed, because the meal is a resource needed for the second step. Such a causal specification is handled by protecting the first state (having the banquet food) until the second step (holding the banquet) takes place. On the other hand, a simple temporal ordering may be specified in the control clause of the activity such as:

“location established for conference”

is-before

“call for papers issued”

Semantically, this ordering is viewed as less rigid, since, although it is standard practice to establish the site of the conference before issuing a call for papers, the location is not a necessary prerequisite needed to carry out the second step. There is no explicit causal basis for this ordering, although it may be a preferable way to do things.

Once an activity has been selected to achieve a goal, the user may have to specify values for the activity resource variables specified by the input-goal-vars clause of the activity. This clause contains the minimal set of variables manipulated by this activity that must have ground bindings in order to integrate the activity into the ongoing plan and continue. The set of variables in this clause are provided by the knowledge engineer during the static definition of the domain. The process by which an activity is selected to achieve a goal (including the binding of the input-goal-vars) is described in Section 5.3.

After the binding of input-goal-vars has been completed, the decomposition is used to generate a replacement for the goal node that is being achieved. The

activity decomposition is represented internally as a mini plan-network. This mini-network is instantiated (using the bindings established for the input-goal-vars) and spliced into the overall current plan network, replacing the goal node. The instantiated mini-network conforms to the description of plan networks given in Chapter 4 and thus is a partially ordered set of goal nodes, structural nodes, agent-executable nodes and tool-executable nodes. For example, suppose the planner is to achieve the high-level goal of successfully running an academic symposium. The original top-level-goal node in Figure 16 on page 106 has the goal of (a completion-status of planning-in-uncertain-environments-symposium is successful). The activity stage-a-symposium (shown in Figure 17) is chosen to achieve the goal. A plan network fragment representing the partial ordering of substeps contained in that activity is instantiated and used to replace the goal node. Figure 18 illustrates the plan network of Figure 16 after the goal node is replaced by the piece of plan network representing the decomposition of the activity chosen to achieve that goal node.

5.1.1 Agents

Every activity is performed by an assigned agent. Agents are selected for activities according to knowledge in the domain model about their capabilities. If the description for an activity that is selected to achieve a goal specifies the responsible agent as a variable instead of as a named agent instance, the current agent who is responsible for the desired goal (the "contracting" agent) is selected as the default agent to carry out the activity. The exception to this heuristic is when the default agent does not satisfy the constraints on the agent variable specified in the selected activity. In that case, an agent is selected

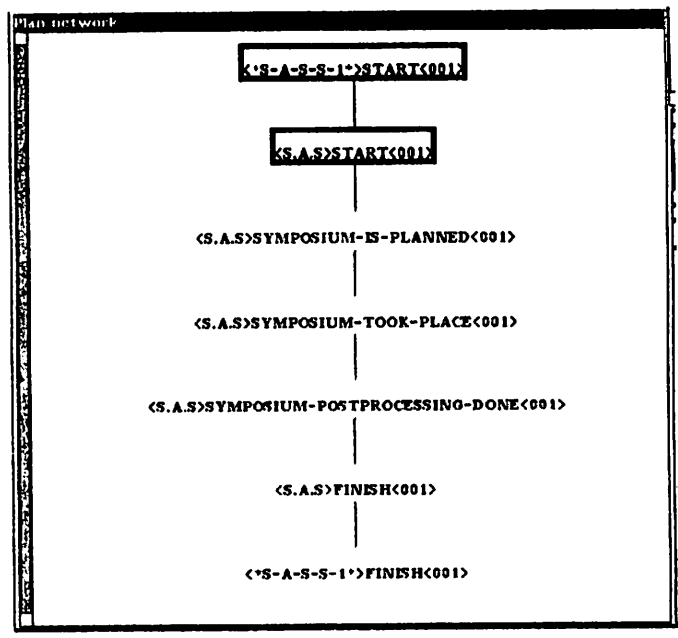


Figure 18: POLYMER plan network after expansion

interactively by the user from the set of known agents in the knowledge base who satisfy the relevant constraints.

5.1.2 Composite versus Primitive Activities

Activities are either composite or primitive. Composite activities are those that have non-null decompositions, and primitive activities are associated with actions that are atomically executable by an agent or a tool. Primitive activities can be mental activities to be performed by agent (e.g., making a decision) or activities that are to be contracted out atomically to an agent at another site (e.g., have the butler take care of it). Such activities are represented as *advertisements*, sometimes also referred to as *agent-executable* activities. Primitive activities that are described in terms of tool invocations are represented as *tool-executable* activities. Although primitive activity descriptions do not contain any information about the manner in which the goal is actually to be achieved (i.e., null decomposition), the constraints or effects contained in their descriptions are incorporated into the domain model and used to model the effects that the execution of these activities will have on the task plan.

When an agent-executable-activity (or tool-executable-activity) is selected by the planner to achieve a goal, the plan network fragment that is used to replace the goal-node in focus consists of a single agent-executable (or tool-executable) node representing the atomic action, along with the structural nodes necessary to delimit the start and finish of the activity expansion.

5.1.3 Objects

The description of an activity refers to objects that will be manipulated in the domain. These objects may be physical (e.g., a proposal form to be submitted in order to hold a symposium) or conceptual (e.g., the symposium itself). Objects relevant to a particular domain may be described in terms of their properties, constraints on these properties, and constraints between sets of objects in the knowledge base. Object descriptions are maintained in the same knowledge base as the activity descriptions that manipulate them. The dynamic model of the domain consists of the state of the objects within the domain at a given point in the plan. An example static object description is shown in Figure 19. The value-class specifications state restrictions on property values of the object, similar to the range constraints used in ISIS, that uses the representation language SRL [26].

5.1.4 Constraints

POLYMER provides a language and facilities for expressing and maintaining a variety of constraint types. Constraints may be attached to fields in object or to activity definitions. They may govern the values that a field can take on or determine the possible orderings for sub-activities or possible resources that can be used by a particular activity. A summary of the types of constraints and how they are used in POLYMER is stated below:

1. Static constraints

- (a) In object schema; valid ranges or values; for object field values

```
{OBJECT submissions
  *subclass-of: instantiable.entities
  *paper: VALUE-CLASS: papers
  *submitted.to: VALUE-CLASS: technical.meeting
  *decision: VALUE-CLASS: possible.decisions.on.conference.papers
  *refereed.by: VALUE-CLASS: program.chair
  *considered.reviewers: VALUE-CLASS: reviewers
  *invited.reviewers: VALUE-CLASS: reviewers
    INVERSE-SLOT: papers.asked.to.review reviewers
  *declined.reviewers: VALUE-CLASS: reviewers
  *reviewers: VALUE-CLASS: reviewers
    INVERSE-SLOT: papers.currently.reviewing reviewers
  *reviews: VALUE-CLASS: reviews
  *review.status: VALUE-CLASS: (one.of received-for-review reviewed)
}
```

Figure 19: An example domain object description

- (b) In activity schema; about objects used;
- (c) In activity schema; temporal and causal constraints among sub-goals;

2. Dynamic constraints

- (a) Instantiations of constraints of type 1b above;
- (b) Instantiations of constraints of type 1c above;

A complete definition of the language used in POLYMER to represent domain activities, objects, agents, and plan networks can be found in Appendix A, and the reader is referred to [40] for a more detailed description of the internal operations of the planning system itself.

5.2 Modifications to POLYMER Representation to Handle Exceptional Occurrences

In order to appropriately represent activity that may differ from the straightforward execution of a detailed plan (exceptions), modifications were made to the basic activity and plan representation used by POLYMER. Two specific modifications are described in this section: (1) the addition of *offline plan networks* to modify external activity, and (2) the *user assertion* activity to model unexpected state changes that are introduced by the user and not tied to any particular activity.

5.2.1 Offline Plan Networks

As previously discussed, every activity has an associated decomposition that is represented internally as a plan network fragment, and is stored in the `plan.network` slot of the activity. This plan network fragment represents the typical way to go about achieving the task at hand. To represent an unexpected way of carrying out the activity, an additional field referred to as the `offline.plan.network` was added to each domain activity. This field contains a second plan network that represents an arbitrary atomic "non-standard" way of performing that activity. This network always consists of a single `offline-agent-executable-node`, which is a special class of `agent-executable node`.

Unlike nodes in standard plan networks for an activity, the distinguished node in an `offline-plan-network` does not inherit the constraints specified in the activity description. An `offline-agent-executable-node` is used to represent an activity that was deemed to have "occurred" in some unobservable fashion and

was not monitored by the planning system. The rationale behind this is the following: constraints on an activity represent what should be true in order to perform the activity in the manner prescribed by the activity decomposition. When the activity is accomplished offline, it is no longer known whether it was done in the particular way that was expected. Therefore the question of whether or not the conditions particular to this activity are satisfied at this point in the plan is no longer relevant.

The original notion of accomplishing a plan goal *offline* is based on Huff [34], who uses special offline operators to model activity that is not observable. Our usage is similar. However, while Huff's offline operators are pre-coded into the domain description for particular activities, we operate under the assumption that every "on-line" activity can potentially take place in an offline fashion, and thus every activity has both an "on-line" and "offline" representation. Whenever it has detected that a state has been achieved in an unexpected way, it is represented using the offline representation.

5.2.2 User-assertion Activity

During the planning and execution of a task, an agent may have acquired some knowledge about the domain and the problem they are working on that would be useful to provide to the system². This knowledge may not be tied to the performance of any activity, known or unknown to the system.

²In general, it would be useful to provide a general facility for an agent to supply newly acquired information to the planning system at any quiescent point during the planning/execution of a task. Currently, the only point at which such information can be provided is when a sub-task is selected for execution and the agent is thus directly involved.

In order to model this behavior, a distinguished static activity description called a user-assertion is created for a domain at load time. The generic form of this activity is devoid of any template specifications; that is, it does not contain a goal, precondition, decomposition, or any constraints. When the user provides feedback to the execution monitor that is not tied to any specified activity name, an instantiation of the special user-assertion activity is created. The new information is represented by assigning the goal of the newly instantiated node to be the state that was achieved by the agent.

5.3 Activity Selection

When a goal node is being processed by the planner, an activity must be selected to achieve the goal³. This selection process can be viewed as having four phases: First, a preliminary computation of possible activities is computed at domain load time for each goal step in every activity. This is accomplished through a simple unification process that determines the set of activity goals and bindings that match the uninstantiated goal in each such goal step.

Then, at run time, the planner further filters this possible activity set by checking preconditions and constraints of each candidate activity with respect to the before-world of the goal node. Currently, the candidate activity must pass three additional filters. The second phase of filtering determines whether the precondition of the activity is satisfied in the before world of the goal node. The third phase checks whether all type restrictions on resources mentioned in

³Actually, the planner first checks to see if the node can be made into a phantom, in which case no activity is necessary.

the goal of the candidate activity are satisfied in that world as well. The fourth and final filtering for activity selection computes whether the set of constraints currently being imposed on the context of the goal to be achieved is consistent with the set of constraints that are to be introduced by the candidate activity. The first three of the filtering mechanisms just described require simple invocations of the unification mechanism supplied by our KEE[®] 4 implementation.

The fourth filter is necessary in addition to the first three filters since goal variables are often unbound at the time of activity selection. Therefore it may not be possible to check whether a ground binding exists to satisfy the specified constraints since the object may not yet exist. For these cases, we can use the semantics of the constraints to determine whether the combined constraint set is inherently inconsistent. Since a general mechanism for computing whether an arbitrary set of constraints is inconsistent is not tractable, an approximation has been implemented. Given a set of predefined inconsistencies, the system has the ability to recognize when a constraint set exhibits any of these specified illegal characteristics⁵.

An example of a general type of inconsistency that has been prespecified and implemented is the following: Suppose type constraints are specified on a given resource in the goal node (referred to here as parent-constraints) and additional type constraints are specified on the same resource⁶ in the activity description being considered (referred to as child-constraints). If the activity

⁴KEE is a registered trademark of IntelliCorp, Inc.

⁵The provision of a more general mechanism for determining the inherent unsatisfiability of a set of constraints is a topic for future research.

⁶The "sameness" of the resource is determined by the variable mappings resulting from the goal unification process that was carried out during the initial phase of activity selection.

under consideration is referring to a resource specified in the goal node, any type restrictions that are specified must be consistent with those type restrictions found in the goal node the activity is meant to achieve. Specifically, the types specified in the candidate activity must be either the same types or specializations of the types specified in the goal node. In exact terms, in order for the combined set of constraints to be consistent, each class restriction on the resource mentioned in the parent must subsume all class restrictions mentioned in the child.

To make this more concrete, suppose we are planning a task for building a house. The goal currently being worked on is to install wiring and is specified as (installed house ?wiring) and has the constraint that (?wiring is of type wiring). One of the candidate activities has a goal of (installed house ?electrical-part) and the constraint that (?electrical-part is of type switch). The goals themselves unify and the preconditions and goal-node constraints are satisfied in the before-world of the goal node (there are known instantiations of switches available), so this activity passes the first three phases of filtering. But note that the combination of the activity constraints and the goal node constraints produces a resource constraint of ((?x is of type switch) and (?x is of type wiring)). This is an inconsistent constraint set, since an object that is a wire cannot also be a switch (see partial fragment of knowledge base in Figure 20)⁷. Therefore, since a switch is not a subclass of wiring, this activity would not be selected. On the other hand, an activity whose goal was (installed house ?electrical-part) where (?electrical-part is of type copper-wiring) would pass the constraint consistency test, since copper-wiring is a subclass of wiring.

⁷We adopt the closed-world assumption when interpreting possible subclass relationships; that is, if a wire is not known to be a switch we assume initially that it can not be.

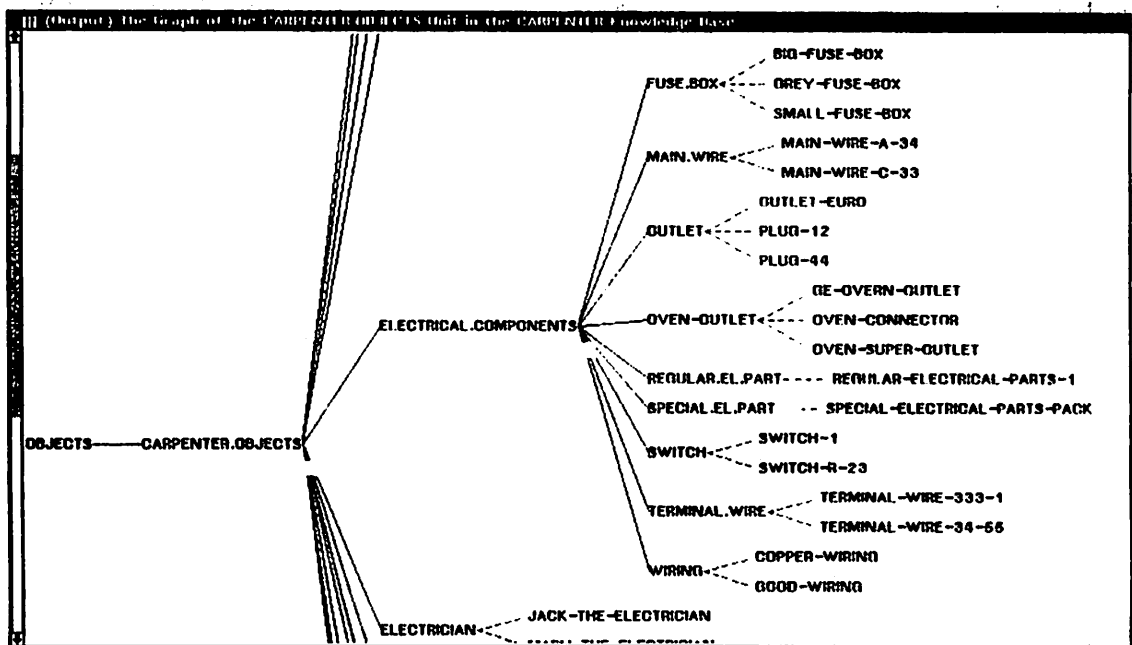


Figure 20: A partial graph of entities in the CARPENTER domain

Once these levels of filtering of known activities have been applied, three possible cases exist. First, if a single activity has survived the filtering, it is automatically selected and instantiated. Any necessary bindings for the newly instantiated activity are determined (as described in Section 5.3). The goal node is then replaced by a plan network fragment representing the decomposition of the chosen activity, which has been instantiated with those established bindings. A second case exists when multiple candidates survive all the phases of filtering just described. In this case the user is asked to select one of the activities and it is then instantiated and incorporated as described. A third case exists when no activity is known to achieve the target goal. In such a case, a no-activity-possible exception is triggered (see Section 6.1) and the DACRON [43] interface can be used to elicit a description of a new activity from the agent that can be incorporated into the domain model and used for subsequent planning.

Binding Activity Input Variables

As previously mentioned in Section 5.1, when an activity has been selected to achieve a goal, the user may have to specify values for the input variables that were specified statically by the knowledge engineer and are contained in the input-goal-vars clause of the activity. In this section, we describe the process by which this is done.

All variables in the input-goal-vars clause are retrieved, and if the agent specified for the chosen activity is also specified as a variable it is added to this list, since an agent must be known in order to activate an activity. The activity that has been selected is being instantiated in the context of some

known bindings. These bindings are being inherited from the selected goal node along with mappings resulting from the unification of the target goal with the goal of the activity description. The planner then tries to find unambiguous bindings for the input-goal-vars (and agent if necessary) from these incoming bindings. If any required input variables remain for which there are no unambiguous incoming bindings, the user is queried to choose from a set of binding possibilities. The computation of the possibilities is described below in Section 5.4.

5.4 Suggestions

This thesis is aimed at improving the system understanding of unanticipated human agent input. However, no matter how much knowledge is embedded in order to construct reasonable alternative explanations of unusual actions, it is always possible that the agent has done something truly incorrect (i.e., a real error). In addition, we initially assume that our domain model is mostly accurate at predicting how action should take place in the domain. These two factors — the assumption of a largely accurate model, and the propensity for humans to commit true errors — suggest that the agents in an interactive planning system should initially be steered towards the anticipated responses whenever nondeterministic action is possible. In sum, agents should be encouraged to do what is suggested but allowed to deviate if they insist.

It is this philosophy that underlies the provision of a facility for suggesting choices of resources and other required information that must be made by human agents while engaged in interactive planning. In the POLYMER system,

the user must choose bindings in two different situations: (1) during goal node expansion and the associated activity selection (this is the binding of input-goal-vars described in Section 5.3), and (2) during the execution of an agent-executable action, whether it is suggested or an unexpected event (all variables in such an action must be bound at execution time). The user is encouraged at these times to choose as resource bindings objects that are compatible with the constraints of the domain and the current plan context. The basis for the computation of these possible values is an exploitation of relevant constraints in the domain model. This is a heuristic approach to a generally intractable problem, and is described below.

5.4.1 Constraints: Used as Parsers

POLYMER, like many other knowledge-based systems built on top of a frame-based representation tool, provides a language for declaring constraints that must hold when applied to those objects or when values are provided for fields of those objects. For example, when an order-form for an office purchase is defined using such a representation language, constraints might be declared on:

- the form object as a whole (e.g. every form must have a source-of-payment specified in order to be considered complete), or
- a particular field of the form object (e.g. the source-of-payment field of a form must contain a reference to a umass-purchase-order-number).

Constraints such as those specified above are generally used by knowledge-based systems to check the validity of object specifications or values stored in object fields. Thus a membership test is triggered by the assertion of a specified value into a specified slot of a specified object. In a sense, the constraints taken as a whole are used to "parse" the specified value to determine its validity. For example, an assertion might be made that the source-of-payment for the order contained in form-1 is C.O.D. The constraint: (the source-of-payment for an order can be a umass-purchase-order-number) is evaluated with regard to the specified value C.O.D. and fails; signalling the user that this is an invalid value for this field.

5.4.2 Constraints: Used as Generators

The use of specified constraints to determine the validity ("parse") of a suggested value is common. We also use constraints to generate a set of possible valid values for a field of an object. The application of constraints as a *generator* of valid values provides valuable assistance to human agents by suggesting known valid values to the agent during the interactive binding of variables and thus steers the agent away from supplying a value that may introduce inconsistency into the plan.

For example, suppose the user is asked to provide a source-of-payment for an order. Rather than simply providing a facility for the user to type in an arbitrary value (that may possibly be invalid), the system could present the user with suggested values. In this case, a list of all umass-purchase-order-numbers that the current agent is authorized to use would be presented to the user for selection.

Sources of constraints

As mentioned above, the user is interactively involved in binding selection in two situations: (1) when bindings must be selected for input variables in the goal of the activity selected, and (2) during the execution of a tool-executable or agent-executable activity. In both of these cases, the variable(s) to be bound are situated within a context that imposes constraints on its possible values. This context is heuristically exploited to generate possible values for the user to select a binding from. The sources of constraints in this context are as follows:

- **Dynamic node constraints.** These constraints represent all known constraints on variables inherited from elsewhere in the plan. They are checked as part of the selection of an activity to achieve a goal. The potential bindings that result are passed in to constrain further binding selection of input-goal-vars.
- **Static knowledge base constraints.** There are three main sources of static constraints that are used to constrain and generate potential binding values:
 - Constraints that are specified in the selected activity and refer to the variables of concern.
 - The goal of the selected activity, that specifies structural information that can also be viewed as constraints.
 - Type constraints that are attached to slots of class objects in the knowledge base.

Processing constraints to generate suggestions

The process by which these constraints are exploited to generate a set of possible values is sketched as follows:

All bindings that result from the evaluation of the dynamic node constraints are collected as *bindings*₁. Next, the constraints for the selected activity are extracted and transformed into a single constraint wff. If no explicit bindings are found to exist in the before-world of the goal-node for this wff, compatible bindings are found by examining relevant units and type constraints in the static knowledge base (described below in the section on find-compatible-bindings-from-kb). Resulting bindings are collected in *bindings*₂. Then compatible bindings for the goal of the activity are determined, also by examining relevant KB units and type constraints. This third set of bindings are collected into *bindings*₃.

The three binding sets (*bindings*₁, *bindings*₂, *bindings*₃) are then combined to produce a consistent set of bindings that constitute the computed suggestions for the variables that need to be bound. In the remainder of this section, we describe the internals of the process by which the structure of the knowledge base is used to derive this set of compatible bindings, with specific attention to the processing of the type constraints attached to slots.

Find-compatible-bindings-from-kb. This function implements the process by which the static knowledge base structure is exploited to generate possible bindings. It operates on a supplied wff about objects in the domain knowledge base, and possible bindings are returned for the variables that are

in the wff. The general form of a goal or constraint wff is *slot-name(unit-name, slot-value)*⁸. There are three possibilities that must be pursued⁹:

- **The slot-value and slot-name are constants, but the unit-name is a variable.** The suggestions mechanism determines the set of known units that contain the property specified by the slot-name. This initial set is filtered if necessary by checking any constraints attached to the slots in question on the supplied slot-value. Unit-names that survive the filtering are returned as suggestions. The names returned represent the units that *can* have this value in the specified slot.
- **The slot-name and unit-name are constants, but the slot-value is a variable.** The suggestions mechanism determines values that are compatible with the specified slot-name of the specified unit. Constraints attached to the slot are used as generators (described below in Section 5.4.2) in order to determine what known values *could* go into the slots of these objects.
- **The slot-name is a constant but the unit-name and slot-value are both variables.** The set of known units that have the specified slot is computed. Then all possible unit/value pairs are computed by using attached slot constraints for each unit as generators, as described below in Section 5.4.2.

⁸Conjuncts of such simple wffs are also possible, in which case the conjunct is broken up into simple wffs and each is handled separately.

⁹For completeness, there actually exists other cases in which the slot-name is unbound, but this never occurs in our implementation as a result of our domain specification, so it is not discussed or implemented here. In addition, a fully bound wff is not considered, since it contains no variables that can be used to generate bindings.

Using individual type constraints as generators

The generation of a set of possible values given an arbitrary set of constraints is not trivial. A constraint may represent an infinite set of potential values: e.g.; (?x is a number), or it may represent an infinite set for which an enumeration can be approximated: e.g., ((?x is a company) or (?x is of type text))¹⁰.

For the purposes of this discussion, we define the term *enumerable* here to denote cases in which all known members of a type can be enumerated by choosing the appropriate subset of known object values represented in the knowledge base; e.g. members or subclasses of a domain discrete class object are enumerable, but members of other continuous class objects such as number or string are not enumerable. In the current implementation, the nonenumerable-classes include objects of the following types: interval, list, number, set, string.

In addition, although a constraint may represent an infinite set of possible values, it may be possible to partially or fully enumerate the impossible values. Since the field of an object may have several attached constraints of arbitrary complexity, we are also faced with the potential problem of combining several sets of partially or completely enumerable sets of possible and impossible values to produce a final set of suggested values. Approximations are possible, but it is important to indicate whether this final set represents a complete enumeration or not.

¹⁰Here, the list of objects of type company that are in the knowledge base can be enumerated, but not all the objects of type text.

The set of constraints that can be attached to the field of an object can be arbitrarily complex. We are particularly concerned with *valueclass* constraints. *Valueclasses* are special attachments to slots in KEE that specify criteria used to judge the validity of a proposed slot value. The attachment of multiple valueclass constraints is interpreted as an implicit intersection. In other words, the values that would satisfy the valueclass constraints as a whole are:

$$values_1 \cap values_2 \cap values_3 \dots \cap values_n$$

where *values_n* indicates the set of values that satisfy the *n*th valueclass constraint.

Given a reasonably sized knowledge base, a single valueclass constraint can be classified and processed to produce a set of known values that will pass the constraint, as well as a set of known values that will fail the constraint. Depending on the type of constraint (whether or not the values that can satisfy the constraint are members of an enumerable type), these suggested possible and impossible value sets can be viewed as partial or complete. In accordance with this general claim, an individual valueclass constraint can be analyzed to produce the following three values:

- **Known positive examples.** The set of known enumerable values in the knowledge base that pass this constraint.
- **Known negative examples.** The set of known enumerable values in the knowledge base that are known to violate this constraint.
- **Enumerability.** A value that describes the enumerability of the known positive examples that satisfy this constraint. It is obviously not possible

to enumerate all values that do not satisfy a given constraint, except in the case of the universal valueclass constraint, where all values pass the constraint so impossible values trivially become the null set. If all known positive examples are enumerable (according to the definition given on page 129), the value is set to complete, otherwise it is partial.

After each individual constraint is processed, the resulting valuesets are combined to produce the final sets of known positive examples, negative examples and enumerability results. This final result set is processed to produce a final set of known bindings, which are then presented to the user for selection.

Within the above framework, we now introduce the types of valueclass constraints available in KEE. The constraint language that is provided is general and similar to other representation languages such as KNOWLEDGE CRAFT[®]¹¹. We describe how valueclass constraints are processed to derive a set of possible and negative examples and the determination of the enumerability of the positive examples¹². Note that each *valueclass - spec* below may be involved recursively as the value of a *< valueclass - specs >* parameter.

1. (ONE.OF *< specified - values >*)

- **Semantics:** The only valid values for this slot are those specific values in *< specified - values >*.
- **Known positive examples:** all values in *< specified - values >*.

¹¹KNOWLEDGE CRAFT is the commercial product that resulted from the SRL representation language used in ISIS [26].

¹²Note: generators are not implemented for KEE valueclass restrictions memberp and list.of.

- **Known negative examples:** The set difference between all domain object instances in the knowledge base and the $\langle \textit{specified-values} \rangle$.
- **Enumerability:** complete.

2. (NOT.ONE.OF $\langle \textit{specified-values} \rangle$)

- **Semantics:** The values in $\langle \textit{specified-values} \rangle$ are invalid values for this slot.
- **Known positive examples:** The set difference between all domain object instances in the knowledge base and the $\langle \textit{specified-values} \rangle$.
- **Known negative examples:** The values in $\langle \textit{specified-values} \rangle$.
- **Enumerability:** partial.

3. (NOT.IN $\langle \textit{valueclass-specs} \rangle$)

- **Semantics:** All values that are instances of the classes defined by $\langle \textit{valueclass-specs} \rangle$ are invalid values for this slot.
- **Known positive examples:** The union of all known negative examples for each of the valueclass specifications in $\langle \textit{valueclass-specs} \rangle$.
- **Known negative examples:** The union of all known positive examples for each of the valueclass specifications in $\langle \textit{valueclass-specs} \rangle$.

- **Enumerability:** *partial*.

4. (UNION *< valueclass - specs >*)

- **Semantics:** All values that are instances of classes defined by *< valueclass - specs >* are valid values for this slot.
- **Known positive examples:** The union of all known positive examples for each of the valueclass specifications in *< valueclass - specs >*.
- **Known negative examples:** The intersection of all known negative examples for each of the valueclass specifications in *< valueclass - specs >*.
- **Enumerability:** Determined by combining the values for **enumerability** for the individual valueclass-specs. If at least one of these value-specs has only partially enumerable positive examples, the **enumerability** for the entire UNION value-class spec is set to *partial*, otherwise it is set to *complete*.

5. (INTERSECTION *< valueclass - specs >*)

- **Semantics:** A valid is value for this slot if it is an instance of every class defined by *< valueclass - specs >*.
- **Note:** The computation for this valueclass is performed anytime there are multiple valueclass constraints on a slot. The determination of the positive examples, negative examples, and enumerability for this constraint is more complex than the other valueclass constraints, and the actual algorithm is presented in Appendix B.

6. (SUBCLASS.OF < object - class >)

- **Semantics:** Valid values for this slot are objects that are subclasses of the object specified by < object - class >.
- **Known positive examples:** Subclasses of < object - class >.
- **Known negative examples:** The set difference between all domain object entities (class objects and instances) in the knowledge base and the subclasses of < object - class >.
- **Enumerability:** complete

7. (MEMBER.OF < object - class >)

- **Semantics:** Valid values for this slot are objects that are instances of the object specified by < object - class >.
- **Known positive examples:** Instances of < object - class > (which includes instances of all its subclasses).
- **Known negative examples:** The set difference between all domain object entities (class objects and instances) in the knowledge base and the instances of < object - class >.
- **Enumerability:** complete

8. (EMPTY)

- **Semantics:** No values are valid for this slot.
- **Known positive examples:** None.
- **Known negative examples:** All enumerable entities in the knowledge base.

- **Enumerability:** complete.

9. (UNIVERSE)

- **Semantics:** Any value is valid for this slot.
- **Known positive examples:** All enumerable entities in the knowledge base.
- **Known negative examples:** None.
- **Enumerability:** partial.

The process that has been described in this section can provide the user with intelligent assistance in choosing "good" bindings for variables during interactive planning, thus avoiding the introduction of inconsistency into the plan. But what happens when the process fails to produce a reasonable set of suggestions? Failure in this case can be defined as cases in which the system does not provide meaningful suggestions; particularly when either no suggestions are provided, or the suggestions consist of all possible knowledge base values. This can mean one of two things: (1) not enough knowledge was available in the KB to provide suggestions (e.g. missing valueclass specs on a slot will result in a unconstrained list of possible bindings)¹³, or (2) there is nothing valid that is known about.

The first of these cases is obviously less serious than the second, since the allowable values are probably lurking in the list of all possible values and may be chosen by an informed user, whereas in the second case there are

¹³In general, we encourage the KB designer to put as much information as possible into the initial KB so it can be exploited in this fashion.

no allowable values. The latter case results in a `no-known-resource` exception, which is described in Section 6.1. In such a case SPANDEX will be invoked to determine if any modifications may be made to the knowledge base or to the constraints so as to produce potential bindings.

In addition, the recognition that the knowledge base may be incomplete or incorrect implies that the suggested bindings that are computed by the process just described may also be incomplete or incorrect. To allow for this case, the interface allows the user to override the suggested bindings. This action will result in a `resource-choice-inconsistent-with-expectations` exception and SPANDEX will be invoked.

CHAPTER 6

THE SPANDEX IMPLEMENTATION

In this chapter, we present the SPANDEX implementation, concentrating on the actual mechanisms involved in exception explanation and resolution. A description of the types of plan flaws that can arise during interactive planning in our framework is given in Section 6.1. Our taxonomy of plan flaws and the subsequent system responses are discussed within the context of two other systems which also incorporate replanning techniques. Then, Sections 6.2, 6.3, and 6.4 present details about the main modules of the SPANDEX system, those which perform exception classification, explanation generation, and amendment implementation, respectively. The chapter concludes with a discussion of the limitations of the current implementation.

6.1 Exception Detection

As discussed earlier, exceptions are signalled by the detection of one or more of a defined set of inconsistencies. In this section, we outline the types of possible inconsistencies, and in later sections we show how the intelligent

<i>— Exceptions during user interaction</i>
no-activity-possible
activity-choice-inconsistent-with-expectations
no-known-resource-available
resource-choice-inconsistent-with-expectations
agent-response-inconsistent-with-executable-expectation
<i>— Exceptions resulting from planner actions</i>
plan-network-inconsistent-during-planning
plan-network-inconsistent-after-execution

Figure 21: The exception types detected and handled by SPANDEX

handling of these inconsistencies avoids what would otherwise be a planning failure, and can result in an improved and extended domain model.

There are two classes of inconsistencies that trigger the exception handler. The first class is detected during interaction with the user, when planner expectations are contradicted by input from the user. The second class of detectable inconsistencies results from planner actions and is not directly tied to user interaction. Both types of inconsistencies may represent error or gaps in the initial domain model.

The seven types of exceptions identified by SPANDEX are shown in Figure 21, and are described in the following sections.

6.1.1 Exceptions During User Interaction

In POLYMER, user interaction occurs whenever information or verification is required from the user. There are four such points during interactive planning; the first two of these occur when the user is called upon to help the planner

disambiguate at choice points, and the second two interaction points occur during the execution of primitive activities.

- **The user chooses from possible activities.** The user can choose from a set of possible activities to pursue during goal expansion. For example, when developing a plan for staging a conference, the planner may find two possible activities for making hotel arrangements for a conference: contract-out-hotel-arrangements or choose-and-book-hotel-yourself. The user is asked to disambiguate. Two types of exceptions are possible here:
 - no-activity-possible: The planner may be unable to find any existing activities that can achieve the posted goal. The planner is unable to continue without additional information.
 - activity-choice-inconsistent-with-expectations: The user may choose other from the activities choice menu, suggesting an activity to use for planning that is not among the presented activities known to be capable of accomplishing this goal.
- **The user makes resource choices.** The user must choose bindings for the input-goal-vars (that represent necessary resources) when an activity is selected to achieve a posted goal¹. For example, a particular program.chair must be chosen before conference submissions can be received and reviewed. The user is presented with a menu of choices that are known to satisfy the current constraints. The user is asked to disambiguate.

¹This process was described in Section 5.3.

biguate, if multiple resource bindings are possible. Resulting exception types are:

- **no-known-resource-available:** No objects currently in the knowledge base satisfy the current constraints on this resource. This exception is a result of a null suggestions computation, as described in Section 5.4.
- **resource-choice-inconsistent-with-expectations:** The user may elect to choose a resource binding that is not among the suggested bindings known to satisfy the current constraints. This exception is a result of overriding suggestions, as described in Section 5.4.
- **The user performs an agent-executable action.** When the user is asked to perform an agent-executable activity to achieve a goal, they are asked to respond either with the appropriate verification or an indication that they have proceeded in an alternative fashion.
- **The user is asked to verify the invocation of a tool action.** When the system is about to invoke a tool to perform a tool-executable action, verification is elicited from the user. Again, the user can override the intended action of the planner at this point.

The exception that may result in either of the above cases:

- **agent-response-inconsistent-with-executable-expectation:** the agent does not verify the agent action or tool action proposed by the planner; instead chooses to do something different.

6.1.2 Exceptions Resulting from Planner Operations

The second class of potential inconsistencies are those pertaining to the state of the current plan network. The plan network may have become inconsistent, either as a result of: (a) planning actions which do not involve the user, or (b) the incorporation of what appeared to be a valid user action. Specifically, one or more of the plan network consistency criteria defined in Section 4.1 may have been violated. Two additional exception types result²:

- **plan-network-inconsistent-during-planning**: inconsistencies result after a planning action in which an agent is not directly involved, e.g., during straightforward plan expansion.
- **plan-network-inconsistent-after-execution**: execution by an agent proceeds normally, but after asserting the goal and effects of the monitored agent action, plan network inconsistencies result.

The inconsistencies in this second class of exceptions involve the violation of domain object constraints or domain activity constraints. In our implementation, static domain object constraints become apparent via the detection of inconsistent worlds, and the violation of dynamic activity constraints is detected by noting unsatisfied node conditions³.

All of the exceptions in the two classes outlined above can represent errors or gaps in the domain model, and can be used as focal points for the acquisition

²More standard violations, such as the violation of preconditions or protection intervals, can be handled with more traditional replanning techniques, while other knowledge base constraint violations are handled by the SPANDEX process of explanation.

³A condition is the dynamic instantiation of a constraint specified in the activity.

of new domain knowledge. Note that exceptions in the first class are detected before the results of the interaction are reflected in the domain and plan model, so handling them a priori can avoid additional inconsistencies that would result from a simple insertion of the agent response. Exceptions that fall into the second class are not detected during interaction with the user, but rather when the incorporation of a seemingly valid planner or agent action into the global plan context results in a violation.

6.1.3 Plan Flaws Revisited and Comparison With Other Systems

All inconsistencies that are detectable by our planning and exception handling system have been described. In this section, we provide an in-depth comparison of the categorization and handling of plan flaws in SPANDEX to other systems that were reviewed more generally in Chapter 2. First, we compare SPANDEX with SIPE along the dimensions of exception detection, exception incorporation, inconsistency computation, and plan recovery. Then we give a comprehensive comparison of our taxonomy of plan flaws and fixes with two of the taxonomies identified in the literature.

1. Detection of exceptions:

- SIPE: The system is invoked with a set of effects from an unknown agent.
- SPANDEX: The system is invoked with an action and a set of effects from an agent, who is often known to the system. A special blank action models "mother nature" type occurrences, and unusual effects that are brought about by a human agent, and are not associated with the performance of a known activity, are modeled by a distinguished user-assertion activity.

2. Initial incorporation of exceptions:

- SIPE: A node representing the exception is inserted at the current point of execution.
- SPANDEX: Same as above. However, a complex analysis of the significance of the exceptional event follows.

3. Computation of inconsistencies:

- SIPE: The system checks for the existence of a predefined set of problems, such as unsatisfied preconditions and protection violations.
- SPANDEX: In addition to the set of inconsistency types computed by SIPE, SPANDEX also detects: (a) mismatches between actual actions and resources versus predicted actions and resources, (b) activity constraint violations, and (c) arbitrary object domain constraint violations.

4. Recovery:

- SIPE: A general set of recovery measures are invoked.
- SPANDEX: The system constructs explanations for inconsistencies that recommend the modification of or addition to the domain model and/or current plan network. The use of generic recovery measures is often unnecessary or significantly reduced after the generation and incorporation of the explanation. SPANDEX can also invoke generic recovery measures such as those used by SIPE if necessary.

We now revisit the general topic of plan flaws (originally presented in Chapter 2) as an additional point of comparison. In particular, Wilkins [73] and Ambros-Ingerson et. al. [4] have outlined categorizations of potential plan flaws that may be detected during planning and execution. We present a comparison of the categorizations of plan flaws identified in SIPE [73] and IPDM [4] with our own taxonomy in Table 22. Our categorization of plan flaws is shown to subsume comparable plan flaws identified by other systems, although

an exact mapping cannot be constructed due to the shifts in underlying plan representation⁴.

Each entry in the table consists of a plan flaw (**F**), followed by the system response (**R**). The system response may be a set of measures invoked to correct for the resulting problems. Blank entries in the table indicate that this type of flaw is not recognized or handled by the system heading that column of entries. The following clarifications are provided here for the abbreviated statements contained in the table:

- The *violated protection interval* detected by POLYMER/SPANDEX is a more general version of the *phantom not maintained* flaw in SIPE, and comparable to the *unsupported range* in IPEM, since it refers to the situation in which any state that is needed beyond the current execution point becomes false. This includes achieved goals as well as phantom states.
- The *future phantom false* flaw identified by SIPE currently cannot arise in the POLYMER/SPANDEX system due to the current implementation of the node selection algorithm used. Specifically, because due to the depth-first plan network processing algorithm employed by POLYMER, a node will not be recognized as a phantom until it is ready⁵, so a POLYMER/SPANDEX plan network currently can never contain a future phantom. However, the node selection algorithm is parameterized and

⁴For example, IPEM includes a optimization measure when dealing with an "unextended range" flaw, and our current implementation does not attempt to handle "timed-out actions."

⁵Refer to definition of ready nodes in Section 4.1.

SIPE	IPEM	POLYMER/SPANDEX
F: False precondition R: Reinstantiate or pop-redo	F: Unsupported precondition. R: Reduce or introduce	F: Unsatisfied precondition R: Reduce, pop-redo, introduce
F: Phantom not maintained R: Reinstantiate or retry	F: Unsupported range R: Excise range + reduce, or introduce	F: Violated protection interval R: Reduce, introduce or retry
F: Purpose not achieved R: Redo	—	F: Goal not achieved R: No action – planner retry
F: Future phantom false R: Reinstantiate or retry	—	F: Future phantom false R: Reinstantiate or retry
F: False parallel postcond. R: Insert-parallel	—	F: N/A
F: Unknown process variable R: Reinstantiate	—	F: No known resource R: Get binding from user (may require modifying KB)
F: N/A	F: Avoid clobbering (Special action: linearize)	F: Violated protection interval R: Linearize
F: N/A	F: Unexpanded action R: Expand	F: N/A
F: N/A	F: Unexecuted action R: Execute	F: N/A
—	F: Redundant action R: Excise	F: Redundant action R: Explain
—	F: Timed-out action	—
—	F: Unextended range R: Extend range	—
—	—	F: No activity possible R: Acquire new activity (user)
—	—	F: Activity choice inconsistent with expectations (planning) R: Explain + modifications
—	—	F: Inconsistent plan/planning (obj/act constraint violated) R: Explain + modifications
—	—	F: Inconsistent plan/execution (obj/act constraint violated) R: Explain + modifications
DETECTED BY EXECUTION MONITOR		
		F: Activity choice inconsistent with expectations (unexpected activity or goal) R: Explain + modifications
		F: Resource choice inconsistent with expectations (unexpected binding in goal) R: Explain + modifications

Figure 22: A comparison of plan flaw taxonomies

thus the problem could arise with a slight change in the implementation. In such a case, we would handle this flaw using the methods that SIPE employs.

- The *false parallel postcondition* identified by SIPE is not applicable in POLYMER/SPANDEX because our plan representation does not treat effects as postconditions; i.e. effects do not necessarily have to hold after an activity has been performed in order to consider the execution of the activity to be successful.
- The *unexpanded action* and *unexecuted action* flaws identified by IPEM do not arise in SIPE or POLYMER/SPANDEX since the responses to the existence of these two features of a plan network are normal planner actions. IPEM includes them as flaws to force a uniformity of planner actions.

6.1.4 Extensions

The taxonomy of plan flaws and system responses used by SPANDEX represents an extension of the replanning facilities proposed and implemented by other systems. In particular, the last six flaws identified by POLYMER/SPANDEX in Figure 22 represent additions to the other taxonomies. SPANDEX recognizes the violation of arbitrary static domain object constraints in addition to violations of preconditions and intra-activity protected states. Violations due to an incomplete specification of the domain (e.g. no-activity-possible) are also anticipated and trigger knowledge acquisition, allowing for

incremental refinement of the domain model. Our responses to these additional flaws, as well as to the no-known-resource-available and redundant-action flaws (which are also identified by the other systems) involve a constructive process of explanation that has not been employed by either SIPE or IPEM.

6.2 Exception Classification

In this section, we describe in detail the role of the exception classifier in the SPANDEX system. The execution monitor detects the occurrence of an exception. SPANDEX is then invoked with a description of the type of problem that occurred, along with any relevant parameters. For example, if an agent notifies the system that they have rejected a conference submission when they were expected to select a reviewer for the submission, the type of problem would be *agent-response-inconsistent-with-expectations*, and the relevant parameters would include the performed action and the achieved goal, along with the activity and goal which were anticipated. Control is then passed to the exception classifier for a rudimentary classification of the detected exception.

In the first part of this section, we discuss the overall role of the exception classifier in the SPANDEX system. We then give a concise characterization of the context in which the exception classifier is invoked by the execution monitor when exceptions are detected during user interaction. Terminology is defined which will be used in the initial syntactic analysis of detected exceptions. Then we enumerate the various exception classifications and describe what they signify.

6.2.1 The Role of the Exception Classifier

Every exception initially exhibits itself as some type of mismatch. An action or assertion has taken place that conflicts with an existing expectation or fact. The seven exception types shown in Figure 21 are all cases of mismatch at some level. Therefore, the initial task of the exception classifier in designating an exception class is to determine the *source* of the exception. There are two components which need to be identified during this process: (1) the event or fact which triggered the violation, and (2) the constraint or expectation that was violated.

In the case of the exception types *plan-network-inconsistent-during-planning* and *plan-network-inconsistent-after-execution*, the inconsistency is manifest as:

- An static object constraint violation: this is viewed as a mismatch between a value being asserted and a static constraint on the affected object; or
- A dynamic plan constraint violation: this is viewed as a mismatch between a value being asserted and a condition attached to a node.

For the *resource-choice-inconsistent-with-expectations* exception, the mismatch is between the choice of resource, and existing constraints on that resource choice, which consist of both static object constraints as well as dynamic node constraints. When detecting an *activity-choice-inconsistent-with-expectations* exception, the mismatch is between the activities which the planner chose to expand a goal with and the choice that the user made. In the case

of the no-activity-possible and no-known-resource-available exceptions, a special case of mismatch is detected, since no selections are possible.

Finally, when a agent-response-inconsistent-with-executable-expectation exception occurs, the mismatch is either between activities (expected and performed) or goals (expected and performed). For example, when the source of an exception is an unanticipated goal state achieved by the user, the violated expectation is the expected goal, and the violating assertion is the goal which actually was achieved.

A general framework is used in handling the exception. The basic idea is to obtain one of two results: (1) a modification can be made to one of the components of the mismatch, thus restoring consistency, or (2) reasoning is performed that concludes that the violating event/state and the violated expectation are allowed to coexist, and the flagged inconsistency is removed. For example, in the first approach, either the representation of the new assertion or action being performed can be modified to be consistent with existing information, or the pre-existing information is massaged to accomodate the new assertion. An example of the second approach would be the application of heuristic rules to establish that an unanticipated action is really a specialization of the anticipated action and therefore can take the place of the anticipated action.

Both approaches involve the following general steps:

1. The trigger for the violation is determined. Is it an assertion or action being established by the user, or an assertion by the planner of a statement embedded in the domain description?

2. Relationships between the violating assertion or event and other plan and domain elements are computed.
3. Mechanisms are invoked to suggest ways to resolve the exception. If the resolution involves destructive changes, the changes must be propagated appropriately throughout the plan and/or the domain knowledge base.
4. Computation is made of any additional inconsistencies that have been introduced as a result of the instituted changes. If necessary, additional SPANDEX measures or standard replanning measures are invoked.

The first two of these steps are done during the exception classification phase, and the remainder of this section is devoted to the details of how this is done. In the next section, we present terminology particular to the classification of agent-response-inconsistent-with-executable-expectation exceptions⁶. An additional level of classification is performed by the exception classifier for these exceptions to determine the degree of mismatch detected. The following section presents the actual classifications of exceptions which are derived and used to guide the subsequent explanation generation.

6.2.2 Context and Terminology

In the general case, the execution monitor is invoked in a context where the planner needs to accomplish a certain goal, g_{comp} , and has selected a particular catalogued activity⁷ ACT_{comp} to achieve the goal. The activity ACT_{comp} is

⁶Violations which involve constraint violations are not classified further at this point; control passes directly to the SPANDEX modules which construct and implement explanations for the exception.

⁷A "catalogued" activity is one which is contained in the static plan library.

selected by *select-activity-for-goal* as described in Section 5.1. In particular, its static goal template (referred to as $Goal(CT_{comp})$) unifies with the posted goal g_{comp} ⁸.

Once the planner has identified g_{comp} and CT_{comp} , action is taken to achieve the goal via the selected activity. There are two ways in which this may be done. In the first case, the planner may be able to automate the step and the agent is informed of what is about to happen. The agent can input information at this point to override what the planner is about to accomplish. In the second case, the planner may require that an agent perform CT_{comp} to accomplish g_{comp} , in which case an agent is directed to initiate the activity. Again, at this point, the agent may introduce information to override the suggestion.

The input that an agent can provide (as mentioned above) is denoted by (CT_{acc}, g_{acc}) , which specifies that the agent has accomplished some goal state g_{acc} by performing CT_{acc} . We refer to the goal template specified by activity A_{acc} as $Goal(CT_{acc})$. In general, g_{acc} is assumed to be an instance of $Goal(CT_{acc})$. We identify the following special cases:

- The accomplished goal state g_{acc} appears to be unrelated to the CT_{acc} which was performed. To handle this situation, a knowledge acquisition tool is invoked to acquire and assimilate a new activity description. Details on how this is done can be found in [43, 42].
- CT_{acc} may be a null specification, allowing the agent to specify that an

⁸In general, lower-case terms are used to represent instantiated goal forms while upper-case terms refer to the template goal forms found in the generic activity descriptions.

arbitrary goal state has been achieved without performing a specific activity (user-assertion). When ACT_{acc} is null, the template $Goal(ACT_{acc})$ is also null by definition.

In general, the activity and goal state instantiations (ACT_{acc}, g_{acc}) that represent the agent's action demonstrate some degree of match to the planner expectations ACT_{comp} and g_{comp} , ranging from a complete match to a total mismatch.

We also make the following definitions:

1. Every goal specification can be viewed as a state specification consisting of a goal *skeleton* and a set of arguments *args* to which the skeleton applies. Each element of *args* may be either a variable, a constant, or another state specification (to allow for conjunctive goal specifications). For goal templates (e.g. $Goal(ACT_{comp})$ and $Goal(ACT_{acc})$), the skeleton is equivalent to the goal template itself. For instances of goal templates (e.g. g_{comp} and g_{acc}), the skeleton is constructed by replacing all constants in the goal with variables.
2. Every posted goal g_{comp} is represented by a node in a plan network. As stated earlier, the current plan network consists of the leaf level nodes of the current plan outline. As a result, g_{comp} (and its host node) can be said to be embedded within a context C that is derived from the planning history. This context C is made up of:
 - A set of constraints c which apply to variables or bindings of vari-

ables that are specified in g_{comp} . We further divide these constraints c into two classes:

- c_{local} : constraints (with bindings, if they exist) posted by the activity that introduced the node containing g_{comp} into the net.
 - $c_{inherited}$: constraints (with bindings) inherited from a more abstract part of the network, earlier in planning history.
- A set of established bindings b_e that specify the bindings that have been applied as substitutions for goal variables in the original static goal template to produce the current goal specification g_{comp} . These bindings may have been established by the following different operations of the planning process:
 - *goal-matching* during the activity selection that produced the introduction of the node containing g_{comp} (see example in Figure 23).
 - *propagation* of bindings that are established in nodes that precede the node containing g_{comp} . Bindings get established and propagated forward whenever any of the following planning actions are taken: (1) the execution of an action by an agent, (2) the user chooses a binding for an *input-goal-var* during activity selection, (3) a tool action sets a binding.
 - A set of suggested bindings b_{s1} that are possible bindings for variables in g_{comp} but have not yet been applied as a substitution. This set of bindings is based on a computation of possible bindings which meet the conditions c in the context C of g_{comp} . This computation

assumes all established bindings b_e to be permanent. The computation of suggested bindings is explained in detail in Section 5.4.

- A less constrained set of suggested bindings b_{s2} which are possible bindings for variables in g_{comp} but have not yet been applied as a substitution. This set of bindings is computed in a manner similar to b_{s1} , with the stipulation that all established bindings b_e in g_{comp} and the context C of the corresponding node are discarded and replaced by variables for the duration of the computation. This binding set represents bindings for variables that would be possible if previous planning actions could be “rolled back” or modified.

3. Every activity also may specify a static set of constraints on its goal variables. Let $C_{acc-static}$ be the static set of constraints from $ACT_{acc-static}$ on goal variables in $Goal(ACT_{acc})$, and $C_{comp-static}$ be the static set of constraints from ACT_{comp} on goal variables in $Goal(ACT_{comp})$.

The terminology just introduced is summarized in Figure 24.

6.2.3 Classifications

In the preceding discussion, we have pointed out that at any point during the planning and execution of a task, the planner has expectations about what goal and action are to occur next (g_{comp} and ACT_{comp}), and the agent is able to specify what actually has taken place (ACT_{acc} and g_{acc}). The execution monitor must be able to determine whether the agent’s input matches the planner’s expectations. Did the agent accomplish the goal state that was expected by the planner? Was the activity performed by the agent the same as the activity

-
1. A goal g posted in the current plan is:
number-of-rooms-reserved(Campus-Center-Hotel, 5).
 2. Activity $ACT1$ is selected to accomplish g which has goal:
number-of-rooms-reserved(?hotel, ?number),
and declares ?hotel and ?number as input-goal-vars.
 3. When instantiated, goal matching produces bindings:
((?hotel = Campus-Center-Hotel), (?number = 5)).
 4. Matching with activity results in the posting of its subgoals.
Suppose one subgoal is contacted(?hotel, agent1).
 5. This subgoal gets posted as:
contacted(Campus-Center-Hotel, agent1) and becomes the new g_{comp} .
 6. The binding ((?hotel = Campus-Center-Hotel)) is an established binding of
 g_{comp} .
-

Figure 23: Example of establishing bindings during goal matching

Term	Meaning
g_{comp}	a goal posted by the planner
ACT_{comp}	the activity selected by planner to achieve g_{comp}
ACT_{acc}	the activity which was accomplished
g_{acc}	the particular goal state achieved by the agent
$Goal(ACT_{comp})$	the template goal of activity ACT_{comp}
$Goal(ACT_{acc})$	the template goal of activity type ACT_{acc}
$args(Goal(ACT_{comp}))$	the arguments to $Goal(ACT_{comp})$
$args(Goal(ACT_{acc}))$	the arguments to $Goal(ACT_{acc})$
$skeleton(g_{comp})$	the goal skeleton of g_{comp}
$skeleton(g_{acc})$	the goal skeleton of g_{acc}
$args(g_{comp})$	the arguments to the goal skeleton of g_{comp}
$args(g_{acc})$	the arguments to the goal skeleton of g_{acc}
C	the context from the planning process to apply to g_{comp}
C_{local}	local context - bound constraints coming from activity introducing g_{comp} into net
$C_{inherited}$	inherited context - from plan history prior to expansion of activity introducing g_{comp} .
$C_{acc-static}$	the static set of constraints from ACT_{acc} on goal variables in $Goal(ACT_{acc})$.
$C_{comp-static}$	the static set of constraints from ACT_{comp} on goal variables in $Goal(ACT_{comp})$.
b_e	bindings that have been applied through goal matching during activity selection to produce g_{comp}
b_{s1}	bindings suggested for goal variables which remain in g_{comp} , using established bindings as constraints
b_{s2}	bindings suggested for goal variables which remain in g_{comp} , disregarding the established bindings

Figure 24: Terminology used during exception classification

expected by the planner? Various degrees of matching between predictions and results may emerge from this comparison, involving either actions, goals, or both. For example, ACT_{acc} may match ACT_{comp} , indicating that the action taken matched the expected action, while the goal state achieved (g_{acc}) may not match the anticipated goal state (g_{comp}). We are interested in determining how closely the action taken and the goal achieved match the expected action and goal, and when a mismatch is detected, a determination is made of the relationships that the mismatched action or goal has with other elements of the global plan as well as with other related elements in the domain knowledge base.

Although we have initially defined g_{comp} to be the currently expected goal, the exception classifier will fix g_{comp} to point to other nodes (and their goals) in the plan, depending on what part of the plan is being examined for possible relationships. For example, when trying to determine if the unexpected action represents a way to accomplish a step expected later in the plan, g_{comp} will be set to point to later goals, if a strategy is checking whether a redundant action has been performed, g_{comp} will point to goals which have already been achieved, and so on. Thus, the general case is that g_{comp} is considered as a *comparison goal* when examining how the currently achieved goal g_{acc} is related to different parts of the overall plan context.

We have identified five semantic comparison dimensions using this terminology. We use these dimensions as an estimate of the degree of mismatch between a user-generated event and current expectations. These dimensions are described below.

1. **Dimension 1: World-states.** How well does the achieved goal match the comparison goal? In other words, how well does g_{acc} match g_{comp} ?

- **Classification 1.1: Exact-match.** $g_{acc} = g_{comp}$, and g_{comp} is a fully grounded wff (i.e.; no variables). By definition, g_{acc} must also be a fully grounded wff. The goal state achieved and the comparison goal state match exactly. The agent achieved the goal that the planner anticipated.

In the next four cases, the skeletons of the achieved and comparison goals are the same (i.e. $skeleton(g_{comp}) = skeleton(g_{acc})$), but there are varying degrees of consistency between the arguments to the skeleton of the comparison goal and those to the skeleton of the accomplished goal⁹:

- **Classification 1.2: Same-skeleton / unifies / consistent.** In this case, g_{acc} is not a fully grounded wff, but $skeleton(g_{comp}) = skeleton(g_{acc})$ and the bindings established through unification of g_{comp} with g_{acc} are consistent with the established bindings b_e and the suggested bindings b_s .
- **Classification 1.3: Same-skeleton / does-not-unify / inconsistent-e.** In this case, g_{comp} is not a fully grounded wff, but $skeleton(g_{comp}) = skeleton(g_{acc})$, and $g_{acc} \not\equiv g_{comp}$, and $g_{acc} \cong Goal(ACT_{comp})$ ¹⁰ The bindings established through unification of g_{acc} with $Goal(ACT_{comp})$ are not consistent with the established bindings b_e , but are consistent with the suggested bindings b_s .
- **Classification 1.4: Same-skeleton / unifies / inconsistent-s.** In this case, g_{comp} is not a fully grounded wff, but $skeleton(g_{comp}) = skeleton(g_{acc})$, and $g_{acc} \cong g_{comp}$. The bindings established through unification of g_{comp} with g_{acc} are consistent with the established bindings b_e , but are not consistent with the suggested bindings b_s .
- **Classification 1.5: Same-skeleton / does-not-unify / inconsistent-se.** In this case, g_{comp} is not a fully grounded wff, but $skeleton(g_{comp}) = skeleton(g_{acc})$, $g_{comp} \not\equiv g_{acc}$, and $g_{acc} \cong$

⁹In the following definitions, "inconsistent-e" means that the bindings provided by g_{acc} are inconsistent with *established* bindings in g_{comp} , "inconsistent-s" means that the bindings are not consistent with the *suggested* bindings for g_{comp} , and "inconsistent-se" means they are inconsistent with both. The goals cannot unify whenever the bindings provided by g_{acc} are not consistent with the established bindings e .

¹⁰The statement $A \cong B$ means that A unifies with B .

$Goal(ACT_{comp})$. The bindings established through unification of g_{comp} with $Goal(ACT_{comp})$ are not consistent with the established bindings b_e , nor are they consistent with the suggested bindings b_s .

The final case in this dimension occurs when the goal template of the activity which generated the comparison goal does not unify with the skeleton of the accomplished goal. In other words, the accomplished goal conflicts with even the most general form of the goal expected.

- **Classification 1.6: Same-skeleton / does-not-unify / inconsistent-args.** In this case, the skeletons of g_{comp} and g_{acc} are the same, but the goals do not unify because the bindings in g_{acc} conflict with the original "hard-wired" args in $Goal(ACT_{comp})$.
- **Classification 1.7: Different-skeleton / does-not-unify.** In this case, the skeletons of g_{comp} and g_{acc} are not the same, so unification is impossible even between g_{acc} and $Goal(ACT_{comp})$. The goal state achieved by the agent has no immediate relation to the anticipated goal form. The agent has accomplished a goal which is apparently unrelated to the activity which was supposed to take place.

2. **Dimension 2: Activities.** Does the performed action match with the comparison action posted by the planner? In other words, does ACT_{acc} match ACT_{comp} ?

- **Classification 2.1: Yes.** $ACT_{acc} = ACT_{comp}$. The agent performed the activity that the planner anticipated.
- **Classification 2.2: No, but is a specialization.** $ACT_{acc} \neq ACT_{comp}$, but ACT_{acc} is a specialization of ACT_{comp} ¹¹.
- **Classification 2.3: No, but is a generalization.** $ACT_{acc} \neq ACT_{comp}$, but ACT_{comp} is a specialization of ACT_{acc} .
- **Classification 2.4: No relationship.** $ACT_{acc} \neq ACT_{comp}$, and ACT_{acc} is neither a specialization or generalization of ACT_{comp} .
- **Classification 2.5: Unknown-accomplished-activity.** When ACT_{acc} is null, a user-assertion has been made, and we can make no comparison.

¹¹For a formal definition of what it means for an activity to be a specialization or a generalization of another activity, see Section 6.3.4.

- **Classification 2.6: Unknown-comparison-activity.** When we are comparing the accomplished goal and activity to a goal node in the net which has not yet been expanded, an activity has not yet been selected by the planner to achieve the goal, so we have no comparison activity.
- **Classification 2.7: Unknown comparison and accomplished activities.** This classification is valid. When both the comparison-activity and the accomplished-activity are unknown.

3. **Dimension 3: Activity-goals.** How well does the template goal of the comparison activity match the template goal of the performed activity? In other words, is there a relationship between $Goal(ACT_{comp})$ and $Goal(ACT_{acc})$? This dimension abstracts away the dynamic context of the comparison and actual goals within the current plan network, and determines whether the static activity goals are similar¹².

We now present the actual classifications:

- **Classification 3.1: Same-goal/consistent.**
 $Goal(ACT_{comp}) = Goal(ACT_{acc})$ and $consistent(C_{comp-static} \cup C_{acc-static})$.
- **Classification 3.2: Same-goal/inconsistent.**
 $Goal(ACT_{comp}) = Goal(ACT_{acc})$ and $not(consistent(C_{comp-static} \cup C_{acc-static}))$.
- **Classification 3.3: Goals-unify/consistent.**
 $Goal(ACT_{comp}) \cong Goal(ACT_{acc})$ and $consistent(C_{comp-static} \cup C_{acc-static})$.
- **Classification 3.4: Goals-unify/inconsistent.**
 $Goal(ACT_{comp}) \cong Goal(ACT_{acc})$ and $(not(consistent(C_{comp-static} \cup C_{acc-static}))$.
- **Classification 3.5: Goals-not-unify.**
 $Goal(ACT_{comp}) \not\cong Goal(ACT_{acc})$.

¹²Note that this dimension need only be considered in the cases where the activity performed is not of the same type as the activity expected (otherwise, this comparison results in a trivially true result). In other words, this dimension is relevant only when $ACT_{comp} \neq ACT_{acc}$.

4. Dimension 4: Achieved-state-and-performed-activity-goal.
 Does the achieved goal match the template goal of the type of activity which was performed? In other words, does g_{acc} unify with $Goal(ACT_{acc})$ ^{13?}

- **Classification 4.1: Internally consistent.** $g_{acc} \cong Goal(ACT_{acc})$. The goal achieved unifies with the template goal of the activity type which was actually performed. The goal that the agent achieved is the goal of the performed (unexpected) activity.
- **Classification 4.2: Internally inconsistent.** $g_{acc} \not\cong Goal(ACT_{acc})$. The goal achieved does not unify with the template goal of the activity type which was actually performed. The goal that the agent achieved is not related to the goal of the performed (unexpected) activity. Achieved goal and performed activity are apparently unrelated.

The execution monitor constructs a match record for each comparison between (ACT_{acc}, g_{acc}) and (ACT_{comp}, g_{comp}) containing values calculated for each of the dimensions¹⁴. A typical match-record is shown in the partial illustration of an explanation in Figure 25. The four dimensions have each been assigned a value; this match.result indicates that the achieved state is an exact match (no variables) with the goal of the comparison node (Classification 1.1), the accomplished activity is a generalization of the activity selected to achieve the goal of the comparison node (Classification 2.3), the activity goals match

¹³Note that this dimension is at first cut only relevant when the performed activity differs from the expected activity (otherwise the comparison would trivially result in the same answer as Dimension 1 (above)). Within this subset of cases, this comparison is also only relevant when $Goal(ACT_{comp})$ does not completely match $Goal(ACT_{acc})$ (otherwise, again, the result would be trivially equivalent to the result of Dimension 1).

¹⁴The current implementation does not consider Dimension 4; if the user specifies a goal which is inconsistent with the specified activity, an error is signalled. The DACRON interface can be invoked at this point to acquire a new activity (with a different name) that has the specified goal, but this currently is not done. Therefore, the value of this dimension for all exceptions considered by SPANDEX is trivially set to 1.

The SUBSUMES CURRENT STEP RECORD 108 Unit in SPANDEX Knowledge Base	
Unit:	SUBSUMES.CURRENT.STEP.RECORD 108 in knowledge base SPANDEX
Created by:	BROVERMAN on 7-21-90 15:43:16
Modified by:	BROVERMAN on 7-21-90 15:43:28
Member Of:	SUBSUMES.CURRENT.STEP
Own slot:	COMPARISON.NODE from SUBSUMES.CURRENT.STEP.RECORD 108
Values:	<ISSUE.CALL.FOR.PAPERS>CALL-FOR-PAPERS-PROCESSED-AND-ISSUED<001> in CONFERENCE
Own slot:	MATCH.RESULT from SUBSUMES.CURRENT.STEP.RECORD 108
Values:	#S(:MATCH-RECORD :WORLD-STATES 1 :ACTIVITIES 3 :ACTIVITY-GOALS 1 :ACHIEVED-STATE-AND-PERFORMED-ACTIVITY-GOAL 1)

Figure 25: An example match record stored in an explanation

exactly and have consistent constraint sets (Classification 3.1), and the goal achieved is consistent with the performed activity (Classification 4.1).

The values of the dimensions presented here are accessed by the explanation strategies discussed in the next section, in order to determine evidence for a given explanation basis. For example, possible activity substitution is suggested when a match record exhibits that the performed activity was a specialization of a selected activity (a value of 3 for the :activities dimension), or a :world-states value of 3 or 5 indicates that an inconsistency exists between parameters of otherwise similar goal states, suggesting possible resource substitution. The identification of the different levels of goal and activity matching represented by these dimensions can be used to rank explanations by closeness of match, and to provide information about whether information in the domain description would have to change, and by how much, in order to accommodate the suggested explanation.

6.3 Explanation Construction

When an exception is detected, the first action of the SPANDEX system is to perform a rudimentary classification of how the triggering aberration relates to the suggested event, as described in Section 6.2. The search for explanations of the exception then begins. In this section, we present the *exception analyst* component of our general architecture in more detail.

Explanations of an exceptional action are generated by the controlled application of strategies represented by a set of *plausible inference rules* (PIs). An example PI is shown in Figure 26. In simple terms, this PI states that when an inconsistent agent response is detected while executing a primitive activity, and the type of action that was accomplished is a specialization of the type of action that was expected by the planner, the accomplished action can be considered to be an alternative way to achieve the pending goal. In such a case, a representation of the predicted action can be replaced by a node representing the accomplished action.

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. The explanation E also has two components: a *explanation basis*, and an *amendment*.

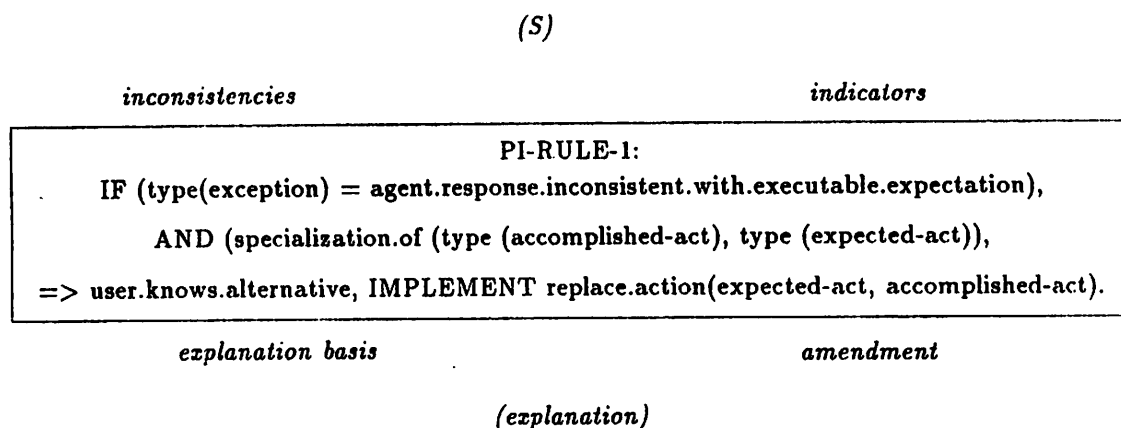


Figure 26: An example plausible inference rule (PI)

6.3.1 Explanation Basis Types

The *explanation basis* gives a semantic basis for the exception, suggesting its interpretation in the context of the ongoing plan. The set of explanation basis types which are currently used by the system are shown in Figure 27. Each explanation basis is associated with strategies that dictate how to look for corroborating evidence (the inconsistencies and indicators described by the state specification *S* in Figure 26). Such evidence provides a measure of support for the explanation basis in justifying the exceptional behavior.

6.3.1.1 Rationales

An explanation basis is considered *purposeful* if it is hypothesized as a rational choice on the part of the acting agent to act in accordance with the high-level goals of the ongoing plan. Such a basis for explanation is referred

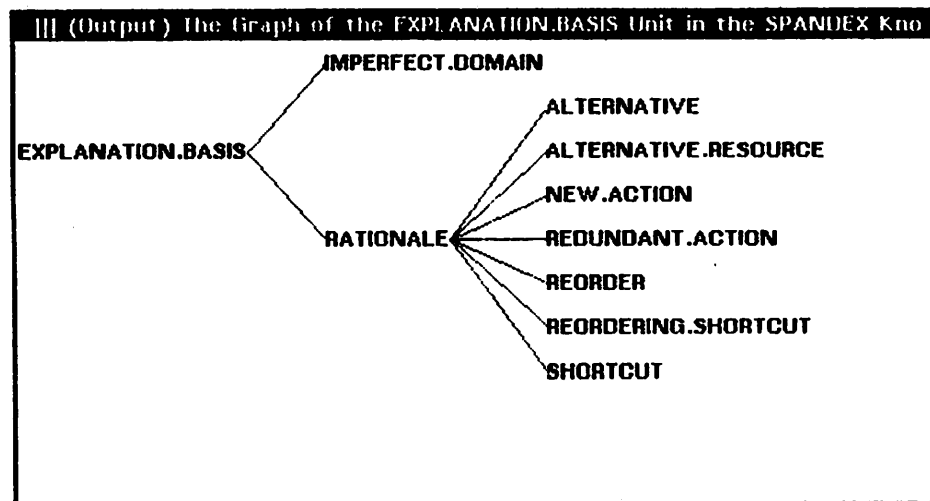


Figure 27: The explanation.basis types used by SPANDEX

to as a *plausible rationale*. We have derived a set of these rationales from analyzing the models of human process behavior summarized in Chapter 3 in the context of an interactive hierarchical nonlinear planning model. The set of rationales used by SPANDEX is described below:

1. **Knows alternative.** The agent is performing an action that is an alternative way to achieve an action anticipated by the planner.
2. **Use alternative resource.** The agent is performing an action that is using a resource similar to the resource the planner expected to use.
3. **Introduces new action.** The agent is introducing a new action that is not anticipated by the planner but could be considered to be part of achieving the high-level goal. The original activity description may be incomplete.
4. **Introduces redundant action.** The agent is repeating a plan step that has already been performed.
5. **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.

6. **Knows reordering shortcut.** The agent is able to perform an action that accomplishes a number of plan steps that are anticipated later in the plan. (This is a combination of the rationales shortcut and reorder.)
7. **Knows shortcut.** The agent knows a way to skip some of the anticipated steps of the plan and still achieve the high-level goal. Alternatively, the agent is able to accomplish an abstract plan goal in "one fell swoop."

6.3.1.2 *Imperfect Domain, the "Else"*

In addition to the purposeful rationales just described, an additional explanation basis for exceptions is the assumption of an imperfect domain. That is, the exceptional event may be valid, and the flagging of its occurrence as exceptional is due to the inherent imperfections in the domain description. As we will show later, the subsequent addition or modification of domain knowledge can eliminate the violation.

6.3.2 **Determining the Scope of an Exception**

The explanation bases just outlined are used by SPANDEX to drive the explanation process that follows the identification of an exception. In order to constrain the search space for explanations, the system cordons off a section of the plan outline to examine. If no explanations are found while examining this region, the system determines incrementally larger portions of the plan history and presents them to the exception analyst for examination. These sections of plan history are referred to as *wedges of reference* since they represent the section of the plan history that the exception analyst may refer to during the process of explanation building.

Given a plan network p , a set of *expected actions*, a distinguished *suggested action*, and an *exception* generated by an agent a , we define a *wedge of reference* that is accessible to the SPANDEX system for the production of explanations for the exception. The informal notion behind a wedge of reference is twofold: (1) the SPANDEX philosophy is that agent A may have behaved in an exceptional way intentionally, and (2) agent A is only aware of the sections of the plan for which he has primary responsibility. The wedges of reference thus define the portions of plan history that are known to that agent and consequently may come under scrutiny by SPANDEX. In other words, SPANDEX may attempt to establish a relationship of the exception to any node contained in an established wedge of reference ¹⁵.

We give a recursive definition of the *wedges of reference* for an exceptional occurrence:

For each expected action whose responsible agent is the same agent who generated the exception, a set of wedges of reference is defined as follows:

1. A node is an apex for a wedge of reference if it is:
 - the *parent goal*¹⁶ of the expected action, and the agent responsible for the parent goal is identical to the agent generating the exception, or

¹⁵This is a general philosophy that applies to a multi-agent model, but we concentrate on a single-agent model in this thesis. In order to extend the system to handle the more general case, the formal definition of the wedges of reference would most likely require refinements.

¹⁶The *parent goal* of a node is defined to be the goal node whose expansion contains that node.

- the parent goal of an established apex for a wedge of reference with the same responsible agent (as the apex).
2. For each wedge apex defined above, a wedge of reference is defined. The nodes contained in the wedge are exactly the apex and all nodes that are derived from the apex through one or more levels of plan expansion.

The set of wedges of reference that result can be ordered by the level of abstraction of the associated apices. The *broadest wedge of reference* is the distinguished wedge whose apex is at the highest abstraction level. Every other wedge of reference is considered a *subwedge of reference* and is subsumed by the broadest wedge of reference.

6.3.3 Establishing the Basis for Explanation

Once a wedge of reference is established as the currently selected scope to use for explanation generation, the exception analyst chooses an explanation basis and instantiates each of its associated explanation-establishment-strategies. An explanation-establishment-strategy represents a method by which the current wedge of reference may be searched for criteria that would justify the hypothesized explanation basis. There may be several such strategies for any given explanation basis.

Every strategy incorporates a method to use to determine the set of relevant *comparison nodes* within the current wedge of reference. Each of these nodes will be examined to determine what relation its associated goal and event information has to the exceptional event. When a strategy is instantiated,

an explanation-establishment-record is created for each such comparison node. This structure is used to hold information that may result when the strategy is invoked using the comparison node as a parameter. In other words, for a given explanation basis B , a set of explanation-establishment-records will be instantiated and initialized for each strategy associated with B . Each member of one of these sets corresponds to a particular comparison node that will be used when executing that particular strategy.

For example, the knows shortcut rationale has six associated strategies: subsumes current step, subsumes concurrent step, accomplish later goal step skip intervening steps, accomplish later action step skip intervening steps, subsumes later step skip intervening steps, and part of later step skip intervening steps. Now consider one of these strategies: subsumes-later-step-skip-intervening-steps. This strategy attempts to show the following informally expressed connection: "Maybe the agent is trying to do a later chunk of the ongoing plan in one step, and doesn't need to do the intervening steps at all." In order to show this, SPANDEX must examine the goal nodes in the wedge of reference that subsume later plan steps, since each goal node inherently defines a "chunk" of the plan. The comparison nodes for this strategy are those n goal nodes in the wedge of reference that are wedge apices for one or more nodes that temporally follow an expected node in the current plan network¹⁷. Thus, n explanation-establishment-records are created during the instantiation of this explanation-

¹⁷In this context we are concerned with the order of execution. Thus, we define that "b temporally follows a if a) we assume that a is to be executed at the current time point t , b) no node that is in the wedge defined by b has been executed, and c) b is not concurrent with a."

establishment-strategy, each initialized with one of the n computed comparison nodes.

Figure 28 shows the explanation-establishment-strategies that are used by the SPANDEX system, illustrating the associations between the strategies and the explanation bases for which they are appropriate. We describe each strategy below in the context of the explanation basis(es) for which it is appropriate. We include a brief description of the semantics of each strategy, along with a description of the comparison nodes for which the strategy will be parameterized. A requirement of all comparison nodes is that they are contained within the current wedge of reference.

1. Knows alternative.

- (a) **Strategy: Accomplish-current-step.** Did the agent accomplish the expected step in an alternative fashion? The plan context is examined for semantic connections between the performed action and the action that was suggested. **Comparison nodes:** the suggested expected action node.
- (b) **Strategy: Accomplish-concurrent-step.** Did the agent accomplish one of the other expected steps (concurrent to the suggested one) in an alternative fashion? This strategy attempts to establish a semantic connection between the performed action and an expected action node other than the suggested node. **Comparison nodes:** the set of executable nodes that are concurrent to the suggested node.

2. Alternative-resource.

- (a) **Strategy: Incomplete-knowledge-base.** Could knowledge be added to the knowledge base to make the resource choice consistent with expectations? **Comparison nodes:** The suggested node.
- (b) **Strategy: Incorrect-knowledge-base.** Could knowledge be modified in the knowledge base to make the resource choice consistent with expectations? **Comparison nodes:** The suggested node.

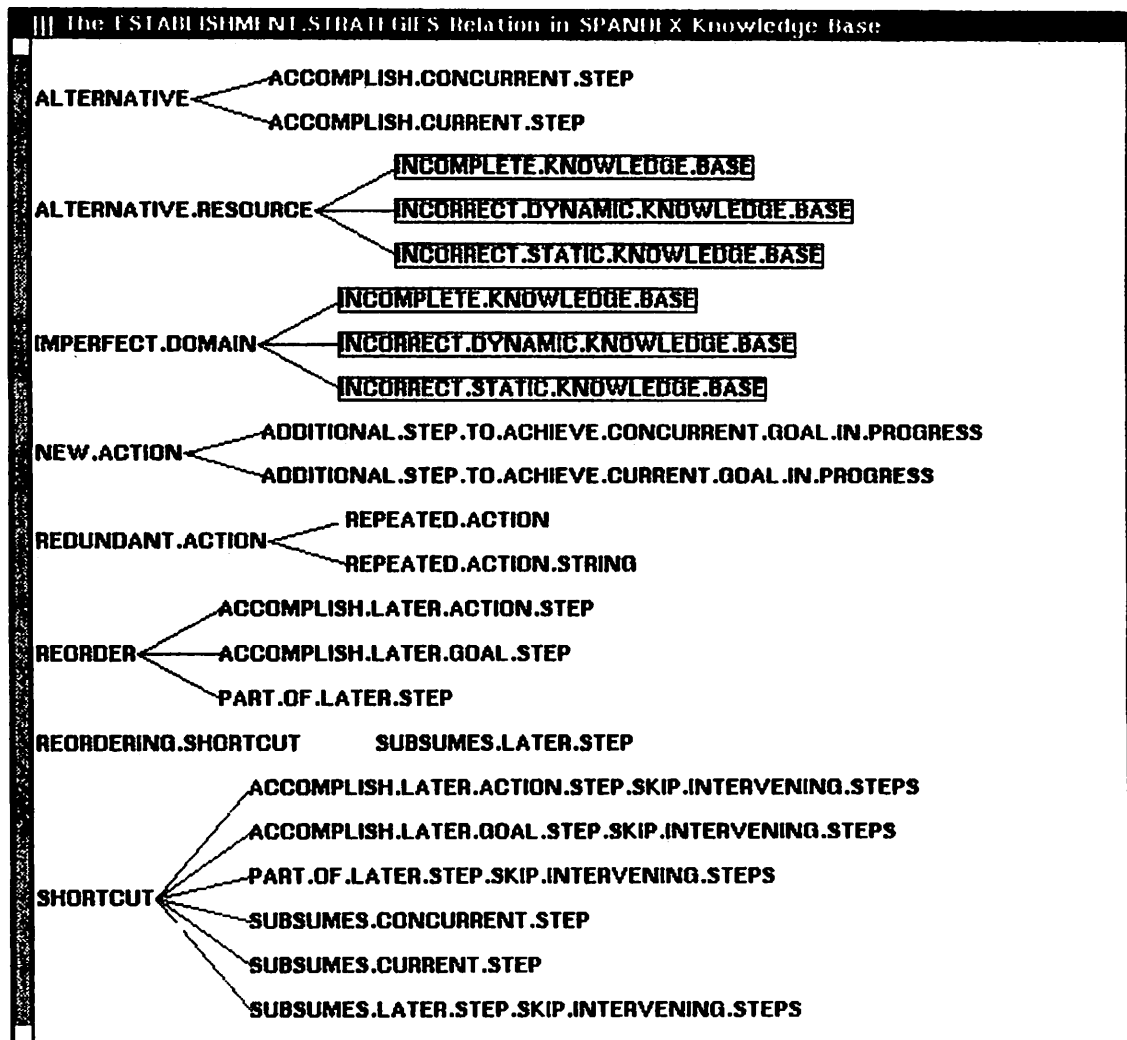


Figure 28: SPANDEX explanation establishment strategies

3. Introduces new action.

This rationale and the associated strategies can be used as a "default" explanation for any action that occurs "not as planned." This is similar to the notion that *intrusion* is the default hypothesis for any erroneous action¹⁸.

- (a) Strategy: Additional-step-to-achieve-current-goal-in-progress. Is the agent doing something that can be considered to be part of the goal in progress? This strategy attempts to establish that the performed action can be part of the expansion to achieve the goal in progress. This strategy does not attempt to form semantic connections between the performed actions and the plan network. The only thing the strategy must check is that the state achieved by the performed action must not violate the goal in progress. Comparison nodes: The wedge apices for the currently suggested node.
- (b) Strategy: Additional-step-to-achieve-concurrent-goal-in-progress. Is the agent doing something that can be considered to be part of a partially accomplished goal other than the one subsuming the suggested action? This strategy attempts to establish that the performed action can be part of the expansion to achieve a goal that subsumes one of the nodes concurrent to the suggested node. Specifically, the state achieved by the performed action must not violate the goal of the chosen comparison node. Comparison nodes: The set of wedge apices for expected action nodes, excluding those that are apices of the suggested node.

4. Introduces redundant action.

- (a) Strategy: Repeated-action. Has the agent repeated an action that has already been performed? This strategy attempts to establish that the performed action is simply a redundant execution of an action that has already been performed. Comparison nodes: All agent-executable and tool-executable nodes that have already been executed.
- (b) Strategy: Repeated-action-string. Has the agent essentially accomplished the results of a set of actions that have already been performed? This strategy attempts to establish that the performed action repeats a string of actions that have already been performed.

¹⁸p. 3 of [33].

Comparison nodes: The unique set of wedge apices in the current wedge of reference whose leaf nodes have all been executed.

5. Can reorder.

- (a) **Strategy: Accomplish-later-goal-step.** Is the agent reordering the plan by accomplishing a later goal step before doing the suggested action? This strategy attempts to find a semantic connection between the performed action and another unexpanded goal node that temporally follows all expected action nodes. **Comparison nodes:** the set of unexpanded goal nodes that temporally follow all expected actions.
- (b) **Strategy: Accomplish-later-action-step.** As above, but with respect to later action steps in then plan. **Comparison nodes:** the set of action nodes currently in the plan network that temporally follow all expected action nodes.
- (c) **Strategy: Part-of-later-step.** Is the agent reordering the plan by starting to accomplish a later goal step before doing the suggested action? This strategy attempts to determine if the performed action could be the initial action in a potential expansion of a later goal. **Comparison nodes:** the set of unexpanded goal nodes that temporally follow all expected actions.

6. Reordering shortcut.

- (a) **Strategy: Subsumes-later-step.** Is the agent reordering the plan by accomplishing a later chunk of the plan before doing the suggested action? **Comparison nodes:** The set of unique nodes, each of which is a) a wedge apex for a node that temporally follows all expected action nodes, and b) not also a wedge apex for the any expected action node.

7. Knows shortcut.

- (a) **Strategy: Subsumes-current-step.** Did the user accomplish a chunk of the plan that includes what was suggested? This strategy attempts to find a semantic connection between the performed action and a higher level node in the plan outline that subsumes the suggested action. The implication is that the agent may have accomplished a set of steps in "one fell swoop." **Comparison-nodes:** the wedge apices of the suggested node.

- (b) **Strategy: Subsumes-concurrent-step.** Did the user accomplish a chunk of the plan that includes one of the actions parallel to the suggested action? This strategy attempts to find a semantic connection between the performed action and a higher level node in the plan outline that subsumes one of the (unsuggested) expected actions. Again, the implication is that the action performed by the agent may subsume (and thus make no longer necessary) several anticipated actions. **Comparison nodes:** The set of unique nodes each of which is a) a wedge apex for a node that is expected (but not currently suggested), and b) not also a wedge apex for the currently suggested node.
- (c) **Strategy: Accomplishes-later-action-step-skip-intervening-steps.** Did the agent skip ahead and accomplish a later action step in the plan, making the intervening steps no longer necessary? This strategy attempts to find a semantic connection between the performed action and a node that temporally follows the suggested node. The assumption is maintained that the intervening steps are no longer necessary, if it can be shown that there is no causal connection (represented by a POLYMER protection-interval) between any of the intervening steps and a step that comes after the identified later action node. **Comparison nodes:** The set of executable nodes that temporally follow all expected action nodes.
- (d) **Strategy: Accomplishes-later-goal-step-skip-intervening-steps.** Same as above, only with respect to a later goal in the plan. **Comparison nodes:** The set of unexpanded goal nodes that temporally follow all expected action nodes.
- (e) **Strategy: Subsumes-later-step-skip-intervening-steps.** Did the agent skip ahead and accomplish a later chunk in the plan, making the intervening steps no longer necessary? This strategy attempts to find a semantic connection between the performed action and a higher-level node in the current wedge of reference that subsumes one of the nodes that temporally follows the suggested node. Again, the assumption is maintained that the intervening steps are no longer necessary, if it can be shown that there is no causal connection between any of the intervening steps and a step that comes after the identified later step. **Comparison nodes:** The set of unique nodes, each of that is a) a wedge apex for a node that temporally follows all expected action nodes, and b) not also a wedge apex for any expected action node.

- (f) **Strategy: Part-of-later-step-skip-intervening-steps.** Did the agent skip ahead and start accomplishing a later (unexpanded) goal in the plan? This strategy attempts to determine if the performed action could be the initial action in a later expansion of a later goal. The assumption is maintained that the intervening steps are no longer necessary (as above). Comparison nodes: the set of unexpanded goal nodes that temporally follow all expected actions.

8. Imperfect-domain.

- (a) **Strategy: Incomplete-knowledge-base.** Could knowledge be added to the knowledge base to satisfy expectations? Comparison nodes: The suggested node.
- (b) **Strategy: Incorrect-knowledge-base.** Could knowledge be modified in the knowledge base to satisfy expectations? Comparison nodes: The suggested node.

6.3.4 Strategy Establishment and the ISS Record

The strategies that are associated with a particular explanation basis are executed in order to provide evidence for that explanation basis. Every strategy has a set of associated inconsistent-state-specs (ISSes). Each ISS represents an (*S*) component of a plausible inference rule (described previously and depicted in Figure 26). An ISS consists of *inconsistencies* and *indicators*, which specify domain object relationships and properties of the current plan state. The verification of the set of indicators and inconsistencies in an ISS specification serves as evidence for the proposed explanation basis.

Each indicator is represented by a structure that specifies a function that looks for the presence of corroborating evidence¹⁹. The structure for an indicator also optionally specifies an applicability function used to determine

¹⁹The same applies to inconsistencies.

whether it is appropriate to apply this indicator in the current context. For example, it is useless to search for a relationship between performed and expected activity types when the unexpected action is represented solely as a state change (e.g., when a user assertion occurs, no performed activity is specified). An applicability function can be attached to the ISS to avoid such irrelevant applications (see IND2 of ISS3 in Figure 29).

Figure 29 illustrates the inconsistent-state-specs (ISSes) that are attached to the Accomplish-current-step strategy of the Alternative explanation basis. The first ISS (ISS3) specifies a set of two indicators to verify in the context of an inconsistent agent response. The first indicator (IND1) specifies that when the achieved state is known, it is reasonable to compute whether the achieved state matches the goal state that was suggested. If this is so, we need only to verify the second indicator to establish evidence for this strategy and rationale. IND2 states that when the performed activity is known, it is reasonable to check whether the activity that was performed is a variation of the suggested activity whereby it contained a more specialized precondition than the expected action. If both of these indicators are verified in the context of an unexpected-agent-response, there is evidence that the agent accomplished the current step in an alternative fashion.

The other inconsistent-state-specs (ISS4, ISS5 and ISS6) are similar to ISS3, but they consider other indicators (e.g. a specialization relationship between variables used in the goals of the two actions, specialization through both preconditions and goal variable constraints, and a relationship between the effects of the accomplished action to the expected goal) to establish evidence for this explanation establishment strategy.

***Rationale*: ALTERNATIVE**

***Establishment.strategies*:**

ACCOMPLISH-CURRENT-STEP

***Inconsistent-state-spec*: ISS3**

***Inconsistencies*: unexpected-agent-response**

***Indicators*:**

IND1: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-SUGGESTED-STATE?

When applicable?: (ACHIEVED-STATE-KNOWN?)

IND2: Establishing-fn: ACCOMPLISHED-ACTION-PRECONDITION-SPECIALIZATION-OF-SUGGESTED-NODE?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

***Inconsistent-state-spec*: ISS4**

***Inconsistencies*: unexpected-agent-response**

***Indicators*:**

IND1: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-SUGGESTED-STATE?

When applicable?: (ACHIEVED-STATE-KNOWN?)

IND2: Establishing-fn: ACCOMPLISHED-ACTION-GOAL-VARS-SPECIALIZATION-OF-SUGGESTED-NODE?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

***Inconsistent-state-spec*: ISS5**

***Inconsistencies*: unexpected-agent-response**

***Indicators*:**

IND1: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-SUGGESTED-STATE?

When applicable?: (ACHIEVED-STATE-KNOWN?)

IND2: Establishing-fn: ACCOMPLISHED-ACTION-PRECOND-AND-GOAL-VARS-SPECIALIZATION-OF-SUGGESTED-NODE?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

***Inconsistent-state-spec*: ISS6**

***Inconsistencies*: unexpected-agent-response**

***Indicators*:**

IND1: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-BEING-COMPARED?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

Figure 29: Inconsistent-state-specs for the accomplish-current-step explanation strategy

The complete set²⁰ of explanation basis types, along with their associated explanation establishment strategies, ISS specs, and amendments specifications is presented in Appendix C. In the section below, we describe how we have chosen to operationalize the notion of activity specialization, since it is an important indicator used to gather evidence for several of the explanation bases.

Activity specialization

In this section, we discuss our chosen operationalization of activity specialization, and describe other possible semantics. Assuming that we have activities A and A' , we say that A' is a specialization of A if one or more of the following criteria is met:

- The resources in the goal of A' are more specialized than the resources in the goal of A . This specialization relationship can only apply if the goals of the two activities unify. The goal variables are extracted from each activity. A specialization relationship holds if (a) there is at least one goal variable of A' whose constraints are a superset of the constraints on the corresponding variable in A , and (b) there are no goal variables A' whose constraints are a subset of the constraints on the corresponding variable in A . For example, suppose the goal specification of A' is `repairs("Joe," ?car(type "cadillac"))`, the goal of A is `repairs("Joe," ?car(type "car"))`, and that the knowledge base contains the fact `(cadillac is-a car)`. We may say that since `cadillac` is a specialization of `car`, then A' can be considered to be a specialization of A .
- A' has a more restrictive precondition than A . We define a "more restrictive" precondition as one that has additional conjunctive clauses²¹.

²⁰The set that is currently implemented. This set could easily be extended as required.

²¹This is also the operationalization of specialization that is used by Tenenberg [67].

In addition to these two definitions that we have adopted for activity specialization, there are several other possible semantics that could be proposed:

- Taxonomic link between two activities in the knowledge base. A' may be explicitly stated to be a specialization of A via a taxonomic link in the knowledge base. It is possible that in such a case, the goals and preconditions may be unrelated — or related only through some deeper semantic knowledge. For example suppose that A' is update-software-system-activity with a goal of updated(system) and A is unix-make-system-activity with the goal being consistent(system), and unix-make-system-activity is linked taxonomically in the knowledge base as a specialization of update-software-system-activity. If we were to implement this interpretation, we may want to insist that the set of goals and effects for A' be a superset of the effects and goal set for A .
- Additional steps (or overriding steps) in the decomposition. Activity A' has all of A 's substeps, along with (a) additional steps or (b) a specification to override a step of A 's of the same name with its own specified step.
- Additional conjunctive clauses in goal. A' 's goal subsumes all the clauses in the goal of A , and contains additional clauses as well. This relationship would imply that A' can achieve states in addition to what A is able to achieve, and amounts to identify the goal state *over-achievers* discussed in [34].

Future versions of the system could extend the operationalization of the specialization relation in the directions suggested above.

6.4 Amendments

Once a valid explanation has been constructed, its associated *amendments* prescribe the changes needed to establish a verified explanation basis and restore system consistency. An *amendment* consists of one or more plan network alterations and primitive modifications made to the domain model. Examples

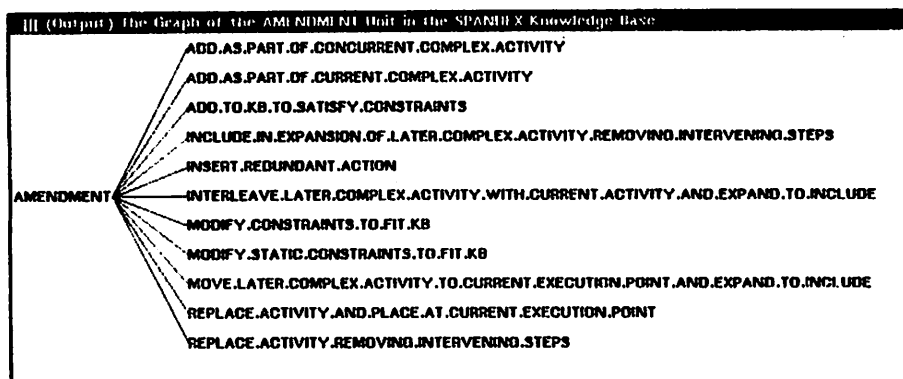


Figure 30: SPANDEX amendment types

of plan network alterations are: (1) replacing an expected action node with a node representing the unexpected action, (2) deleting a wedge of the plan, and (3) inserting a node representing an unexpected action. All plan network alterations involve the introduction or removal of nodes, the reordering of nodes, or the modification of the dynamic constraints on a node. Possible domain model modifications are assertions that add or remove objects in the knowledge base, add or delete values from the field of an object, insert or removal taxonomic links, or modify static object constraints.

The set of amendments currently implemented²² in SPANDEX is shown in Figure 30. The type of amendment appropriate to institute in response to an exception is contingent on the actual type of exception detected and the explanation strategy that was used. The remainder of this section is organized around the different exception types and our taxonomy of amendments are presented within this context.

²²With the exception of the `acquire.new.resource` amendment, which has not yet been implemented.

6.4.1 Amendments for Unanticipated Actions

When the flagged exception involves an unexpected activity or an unexpected achieved goal (agent-response-inconsistent-with-executable-expectation or activity-choice-inconsistent-with-expectations exceptions), the system responds by investigating ways to link the performed activity and goal into the current plan outline. Amendments are then invoked to reflect the connections that have been established. In general, this process involves the modification of the structure of the current plan outline. The particular types of amendments that are appropriate for a given explanation strategy are specified within the completed explanation establishment record. A message is sent to a method that chooses one of the possible amendment types for the strategy and implements the primitive changes to the plan structure.

The amendments described here generally involve one or more of the following operations: the introduction of a new node, the excision of a node, the excision of a plan outline wedge, the expansion of a node using a predefined expansion path (as constructed by the explanation strategy) or a temporal reordering. The invocation of an amendment is parameterized with the comparison node stored in the explanation record (as described in Section 6.3.3) and other optional arguments²³. Below we describe the amendments that are implemented for this class of exceptions:

- Accomplish-current-step strategy. The amendment for this strategy is replace-activity-and-place-at-current-execution-point. The comparison node (in this case, the node that represents the currently suggested executable

²³These arguments may include specifications of whether to reorder or remove intervening nodes. See Appendix C.

step) is excised from the plan outline, and replaced with a node representing the activity that was performed.

- **Accomplish-concurrent-step strategy.** Same as above, except for that the comparison node in this case is one of the other nodes concurrent to the suggested action node.
- **Accomplish-later-goal-step-skip-intervening-steps strategy.** The amendment for this strategy is **replace-activity-removing-intervening-steps**. All nodes in the plan outline that temporally follow the current execution point (including the suggested node) and temporally precede the comparison node (a later goal step) are excised from the plan outline. The comparison node itself is then excised and replaced with a node representing the activity that was performed.
- **Accomplish-later-action-step-skip-intervening-steps strategy.** Same as above, except that the comparison node is a later action step.
- **Subsumes-later-step-skip-intervening-steps strategy.** Same as above, but the comparison node is a wedge apex that subsumes one or more later steps. All nodes in the wedge defined by the comparison node are removed from the plan outline, in addition to the intervening steps as described above.
- **Subsumes-current-step strategy.** The amendment for this strategy is **replace-activity-and-place-at-current-execution-point**. The wedge defined by the comparison node (which is an apex of a wedge subsuming the suggested node) is excised from the plan outline. The comparison node (apex) itself is then excised and replaced with a node representing the activity that was performed.
- **Subsumes-concurrent-step strategy.** Same as above, except that the comparison node in this case is the apex of a wedge that subsumes a node concurrent to the suggested node.
- **Part-of-later-step-skip-intervening-steps strategy.** The amendment for this strategy is **include-in-expansion-of-later-complex-activity-removing-intervening-steps**. The comparison node is expanded to include a node representing the performed action. All nodes that temporally precede the comparison node and have not yet been executed are excised from the plan outline.

- **Accomplish-later-goal-step strategy.** The amendment for this strategy is **replace-activity-and-place-at-current-execution-point**. Since intervening nodes are not excised, this effectively accomplishes a reordering of plan steps.
- **Accomplish-later-action-step strategy.** Same as above, with the comparison node being a later executable node.
- **Part-of-later-step strategy.** There are two amendments for this strategy. The first is **interleave-later-complex-activity-with-current-activity-and-expand-to-include**. The comparison node is expanded to include a node representing the performed action. The node actually representing the performed action is then moved to the current execution point. Thus, the new leaf nodes of the newly expanded comparison node are interleaved with the nodes that come between what was just executed and the prior expectations. The second amendment type, **move-later-complex-activity-to-current-execution-point-and-expand-to-include** does not attempt to interleave the new expansion with intervening expectations. Rather, the comparison node is first regressed back in the plan to the current execution point, and then expanded in that position to include the node representing the performed sub-action.
- **Subsumes-later-step strategy.** The amendment type is **replace-activity-and-place-at-current-execution-point**. This is the same as for **subsumes-later-step-skip-intervening-steps** only the intervening steps are not excised; they remain predicted.
- **Additional-step-to-achieve-current-goal-in-progress strategy.** The amendment is **add-as-part-of-current-complex-activity**. A node is simply introduced at the current execution point and linked into the comparison node (a goal node subsuming the suggested node) as an additional substep.
- **Additional-step-to-achieve-concurrent-goal-in-progress strategy.** The amendment is **add-as-part-of-concurrent-complex-activity**. A node is simply introduced at the current execution point and linked into the comparison node (a goal node subsuming an expected node that is concurrent with the suggested node) as an additional substep.
- **Repeated-action strategy.** The amendment **insert-redundant-action** is applied, which simply inserts a node representing the redundant action and links it into the comparison node a substep that was performed redundantly.

- Repeated-action-string strategy. Same as above.

6.4.2 Explicit Knowledge Acquisition for no-activity-possible Exception

While in the midst of a plan elaboration cycle, the planning algorithms may be unable to find a known activity to satisfy a goal. This is recognized as a point of failure due to incomplete knowledge, and an exception of type no-activity-possible is flagged. The acquire-new-activity amendment is applied, and the DACRON interface is invoked to acquire the description of a new activity that might satisfy the pending goal. This new description is then incorporated into the existing knowledge base, and thus may provide the planner with an additional way to achieve the goal of other subsequent subtasks. The manner in which a user can graphically specify a new activity which can then be used to resolve this exception is described in [43].

6.4.3 Explicit Knowledge Acquisition for no-known-resource

If a resource binding is required by the planner, and none are available that meet the current constraints, one of the possible amendments is to acquire a description of a new object that satisfies the constraints²⁴. The acquire-new-resource amendment is applied, and the DACRON interface is again invoked to acquire the new description.

²⁴Another strategy both both no-activity-possible and no-known-resource exceptions would involve assuming an incorrect knowledge base. The corresponding amendment would then attempt to relax the existing constraints so that currently known resources would become viable choices.

6.4.4 Discussion of Amendments that Involve Constraints

The remaining exception types all involve problems with constraints, either as a result of their explicit violation or their inability to be satisfied in the current plan context. Therefore, a different approach is required for the types of amendments that can be applied to restore a consistent plan state in these situations.

To review, problems in satisfying constraints can arise in our model in three situations:

- The resource chosen by the user does not meet constraints²⁵ (exception-type: resource-choice-inconsistent-with-expectations).
- The expansion of a goal introduces information that violates constraints (exception-type: plan network inconsistent during planning). Either dynamic activity or static object constraints may be violated²⁶ as a result.
- The execution of an action introduces information that violates constraints (exception-type: plan-network-inconsistent-after-execution). Again, either activity or object constraint violations may result.

²⁵This includes the case where there are no resources to suggest to the user for selection that meet the known constraints.

²⁶Violated activity constraints are detected differently than violated object constraints in our implementation. This is because activity constraints are represented as conditions on nodes, and are evaluated as a form of checking, but do not explicitly trigger inconsistencies unless checked. On the other hand, when constraints on domain objects are violated, a built-in mechanism is triggered and a violation is immediately flagged by KEE. The built in mechanisms of KEE must be intercepted in order to handle domain object constraint violations properly.

In general, when constraints are violated, there are different approaches that can be taken towards the resolution of the problem. Two possible general strategies are:

1. Relax the constraints.
2. Remove the source of constraint violation.

Conceptually, the first approach implies that a constraint was specified incorrectly; it was too strict. In this case, the specification of the constraint itself is changed to accommodate the exception. A set of constraint relaxation measures that we have proposed and implemented are described in Section 6.4.5.3. The second approach implies that some knowledge is either not explicit or is incorrect in the knowledge base. Constraints may have failed because of unknown information. If the missing knowledge can be acquired, the constraints may be successfully evaluated. Alternatively, information in the knowledge base may be incorrect, and a change can be made to the knowledge base so that the subsequent evaluation of the constraints is successful. In either case, the constraint itself is left untouched, but additions or modifications are made to the domain knowledge base, thus removing the actual source of the constraint violation. In the following sections, we discuss each of the cases in which constraint violations can arise, illustrating with the mechanisms that have been implemented to resolve the exception.

6.4.5 Inconsistent Resource Choices

When a resource is specified by the user during activity execution that is not consistent with known constraints, a resource-choice-inconsistent-with-expectations exception is triggered. If this resource is used, the plan will become inconsistent, so SPANDEX attempts to explain the choice of resource and suggest modifications that will make this resource become a valid choice. This approach is different than the one outlined above for unexpected actions. Rather than examining other nodes in the current wedge of reference for connections to the performed action, focus is directed to the immediate context of the suggested node and the constraints that were imposed on the choice of resource. If it is determined that constraints have been violated by this choice of resource, there are three fundamental approaches to resolving the problem:

1. The system can ask the user to choose a different binding.
2. The constraints may be modified in order to make the resource choice compatible with the plan context.
3. The knowledge base may be modified in order to satisfy the failed constraints.

We can note some important aspects that differentiate these three approaches. The first approach goes somewhat against the philosophy of this thesis. Since we initially assume purposeful behavior on the part of the human agent, we prefer to investigate if it is possible to make this resource choice compatible with the current plan context, rather than reject the user's choice.

The second approach, modifying the constraints, is nontrivial and difficult to implement in the current version of our interactive planner. In order to correctly implement amendments that modify constraints, we must be able to determine how to propagate an arbitrary constraint change throughout the plan. Because of limitations of the constraint representation of our planner, we can only approximate the determination of the source of constraints, and have added bookkeeping functions to maintain a record of where resource choices (bindings in constraints) were originally established in order to facilitate an implementation of a constraint relaxation approach. However, we prefer to invoke this mechanism as a last resort, since the current implementation is not well-designed to handle this method efficiently. In sum, a strong argument can be made here for a more sophisticated representation of constraints in a future implementation of the system, which is discussed in Section 6.6.

The third approach, the addition of knowledge to the knowledge base, is attractive because by modifying the knowledge base, there is no need to propagate the change; implications of the change are propagated automatically to later parts of the plan²⁷, and the constraints are unaffected. Therefore, in the section that follows, we describe an implementation of the `add.to.kb.to.satisfy.constraints` amendment, which modifies the knowledge base to satisfy a set of constraints. This approach will work for the following cases: when the choice of resource is inconsistent with the set of known constraints (`resource-choice-inconsistent-with-expectations`), when no resources

²⁷This is accomplished via the KEEworlds mechanism used in our implementation.

are known to satisfy constraints²⁸, or when the plan network consistency checker detects that dynamically posted plan constraints are not satisfied in a node world (plan-network-inconsistent-after-execution, plan-network-inconsistent-during-planning exceptions).

6.4.5.1 Adding to the Knowledge Base in Response to Unsatisfied Constraints

In order to describe our implementation of the add-to-kb-to-satisfy-constraints amendment, and its inherent capabilities and limitations, we first define some terminology.

The amendment is applied in a situation where we have identified an arbitrary set of constraints, C . We define a *constraint* to be a simple world-state specification (SWSS) about the domain, of the form shown in Figure 31. Conjunctive and disjunctive constraints are transformed into two or more SWSSes. The set C of SWSSes is viewed implicitly as a conjunct. Upon the triggering of the exception, we know this constraint set to be *currently unsatisfiable*. We define a currently unsatisfiable constraint set to be a conjunct of SWSSes for which no bindings can be found in the current knowledge base. We further define a constraint set to be *inherently unsatisfiable* if it can be shown that the conjunct of SWSSes can never have any bindings, as a consequence of the semantics involved. For example, it may be the case that no conference is known that satisfies the constraint set (locale(?conference, Barcelona), scheduled-month(?conference, August)). This constraint set may be said to be

²⁸This case is distinguished from the case in which no resources will ever be able to satisfy the constraints; that is, the set of constraints is inherently unsatisfiable, e.g. $((x < y) \text{ and } (y < x))$.

```

simple-world-state-spec := predicate (kb-term, kb-term) |
                        not(simple-world-state-spec)
kb-term                := predicate (kb-term) | kb-entity | value | variable
predicate              := system-predicate | kb-predicate
system-predicate       := member | subclass | equal
value                  := string | number | symbol-not-a-var
variable               := unconstrained-variable | constrained-variable
unconstrained-variable := any symbol whose first character is a "?"
constrained-variable  := ?(symbol-not-a-var {world-state | kb-entity})
symbol-not-a-var       := any symbol whose first character is not a "?"

```

Figure 31: The form of a simple constraint

currently unsatisfiable in the context of the current KB. However, the constraint set (locale(?conference, Barcelona), locale(?conference, Paris), scheduled-date(?conference, August-10-1990)) is *inherently unsatisfiable* since no matter what the contents of the KB are, no single conference can be scheduled for two different places on the same day²⁹.

Our initial situation is the following: we have a constraint set C along with an optionally specified proposed resource choice or state change³⁰. If specified, the proposed resource choice or state change is also represented as a constraint and added to C . The amendment is appropriate to apply if C is now determined to be *currently unsatisfiable*. It is assumed that C is not *inherently unsatisfiable*³¹. We refer to each individual SWSS in C as a clause.

²⁹Of course, the actual proof that this set of constraints is inherently unsatisfiable would require that the additional constraint mentioned (e.g., a conference can not be scheduled for two different places at the same time) be represented. All available constraints would then be transformed into the appropriate form and used by a resolution theorem proving mechanism to prove the impossibility.

³⁰The specification of the optional parameters depends on what type of situation the planner is in: whether an inconsistency is detected during resource selection, or after a planning or execution action.

³¹The determination of inherent unsatisfiability is not implemented at this time; all

The fundamental idea behind the algorithm is to partition the original set of N clauses in C into two groups, such that one of the groups is a *currently satisfiable* subset of the original constraint set. If such a partitioning is found, we can use the bindings found for this constraint subset along with the remainder of the constraint set to propose changes that can be made to the KB to make the entire original constraint set C *currently satisfiable*.

The implemented version of the algorithm partitions the constraint set C into two groups: C_p , known as the *primary set*, and C_s , which is referred to as the *secondary set*. The primary set contains M clauses, where $M = \text{*primary-constraint-group-size*}$. The secondary set contains $(N - M)$ clauses. Thus, there are $(N!) / ((M!) * (N-M)!)$ possible partitionings.

A partitioning is represented as sets of constraints and bindings as shown in Figure 32, and the algorithm for determining changes that may be made to an existing knowledge base to satisfy the constraints is described in Figure 33³².

We initially set $M = 1$ when running the algorithm, since the solutions subsequently found during this first pass will represent the smallest amount

constraints in the domain are not represented uniformly in the current implementation, and it would be very costly to extract them. The computation itself would also very costly. This is an area for future extensions.

³²Notes about the algorithm: (1) As N gets large, the number of possible partitions becomes combinatorially explosive. With a large N , we may want to establish a cut-off for the number of partitions to generate; (2) Step 4 is necessary since otherwise, no bindings will be available to use to construct suggested additions to the KB, (3) An assumption that must hold in order for this algorithm to work is that the primary and secondary constraint sets are *fully interdependent*; that is, all variables in the primary constraint set must be present in the secondary constraint set (see step 5). Otherwise, no bindings will be available for the secondary constraint set to propagate into the primary constraint set to propose KB additions.

```
STRUCTURE constraint-partition
  node ;the node to which the constraints are attached
  primary-constraints ;m constraint clauses
  primary-bindings ;bindings-sets ( ((var1 . val1) (var2 . val2))
                                     ((var1 . val3) (var1 . val4))
                                     etc. )
  secondary-constraints ;the (n-m) other constraint clauses
  secondary-bindings ;;combined bindings from the (n-m) clauses
)
```

Figure 32: The constraint partition structure

of change (a single SWSS) that needs to be made to the KB to make the constraints *currently satisfiable*. If this is insufficient to produce results, partitionings with larger sized M may produce possible additions with a larger grain size of change, i.e., multiple SWSSes. We illustrate the constraint-partitioning algorithm just described with the following examples.

Example 1: Suppose the user is selecting a site to hold the SIGMOD conference. Constraints on the site to be chosen specify that a university must be located in the site chosen, and that the site must be a city. In addition, the month in which the conference is held should be during one of the popular seasons for visiting the selected site. The constraints in C are represented as follows:

- (?site is a city)
- (a month of sigmod is ?month)

-
1. Fix M (the *primary-constraint-group-size*). The default value is 1.
 2. Compute each of the " N choose M " possible partitionings of the constraint clause set C .
 3. For each partitioning, compute bindings for the primary and secondary sets.
 4. Filter out all partitionings in which the secondary set is *currently unsatisfiable*.
 5. Filter out all partitionings in which the primary and secondary sets are not *fully interdependent* (i.e., the secondary set does not share all of the primary set's variables).
 6. For each partition that remains, do the following:
 - (a) Extract the possible sets of bindings for the secondary set that refer to the variables in the primary set.
 - (b) For each such binding set, bind the primary constraint clause set with this binding set, and add the resulting fully bound wff to *possible-assertions-to-restore-consistency*.
 7. If *possible-assertions-to-restore-consistency* is empty, increment M and return to Step 2. Otherwise, return *possible-assertions-to-restore-consistency* as the set of possible KB additions to present to the user. The assertion of any one of them would result in the successful evaluation of the original constraint set C .
-

Figure 33: The constraint partitioning algorithm

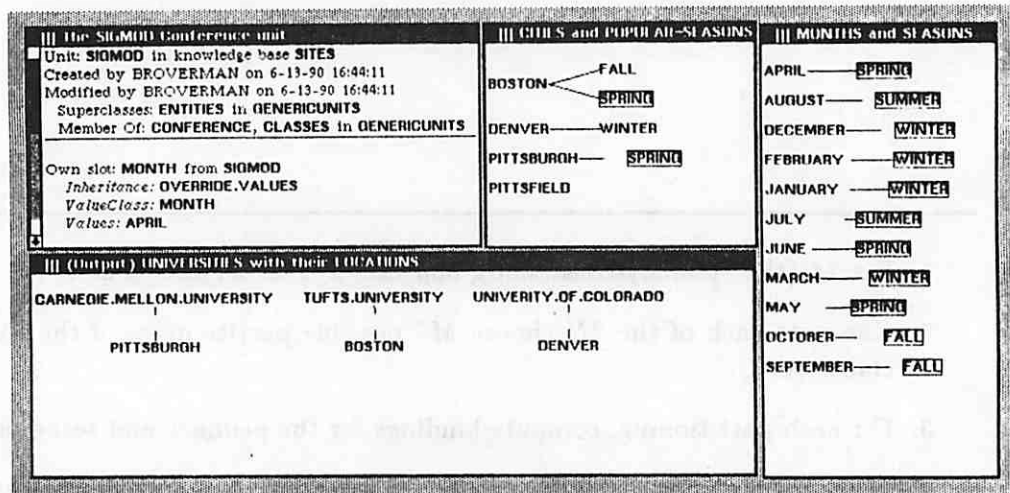


Figure 34: A partial view of the SITES Knowledge base

- ((a season of ?month is ?season) and (a popular-season of ?site is ?season))
- ((?university is a university) and (the locale of ?university is ?site))
- (sigmod is a conference)

Suppose the knowledge base contains information about particular cities, universities, seasons, etc. as pictured in Figure 34. The planner interacts with the user as the select-site executable node is processed. A resource binding is needed for the ?site variable, and the computation of suggestions (as described in Section 5.4) results in two possible known bindings that satisfy the constraints: Pittsburgh and Boston.

The user, however, has decided that they want to hold the conference in Denver, and chooses to override the suggestions. A resource-choice-inconsistent-with-expectations exception is signalled, since the choice of Denver is inconsistent with constraints. The imperfect-domain explanation basis is used to

```

[[[ Output ]] The "EXECUTION.RECORD" Unit in SPANDEX Knowledge Base
Unit: "EXECUTION.RECORD" in knowledge base SPANDEX
Created by BROVERMAN on 3-6-89 13:07:56
Modified by BROVERMAN on 6-12-90 22:25:46
Member Of: ENTITIES in GENERCONITS

Maintains information about the current execution cycle.

Own slot: EXCEPTION.TYPE from "EXECUTION.RECORD"
Values: S:RESOURCE-CHOICE-INCONSISTENT-WITH-EXPECTATIONS

Own slot: OVERALL.EXECUTION.RESULT from "EXECUTION.RECORD"
Values: P:RESOLVED-EXCEPTION

Own slot: PROPOSED.KB.ADDITIONS from "EXECUTION.RECORD"
Values: #[Wff A P:POPULAR-SEASONS OF P::DENVER IS P::SPRING ],
#[Wff A P:SEASON OF P:APRIL IS P:WINTER ],
#[Wff A P:MONTH OF P:SIGMOD IS P:DECEMBER ],
#[Wff A P:MONTH OF P:SIGMOD IS P:MARCH ],
#[Wff A P:MONTH OF P:SIGMOD IS P:FEBRUARY ],
#[Wff A P:MONTH OF P:SIGMOD IS P:JANUARY ]

Own slot: RETURNED.BINDINGS from "EXECUTION.RECORD"
Values: ((#[Var P:<SELECT.A.SITE>SITE] . #[Wff P:DENVER ]))

Own slot: SELECTED.AMENDMENT from "EXECUTION.RECORD"
Values: S:ADD.TO.KB.TO.SATISFY.CONSTRAINTS

Own slot: SELECTED.EXPLANATION from "EXECUTION.RECORD"
Values: INCOMPLETE.KNOWLEDGE.BASE.RECORD63

Own slot: SELECTED.KB.ADDITION from "EXECUTION.RECORD"
Values: #[Wff A P:POPULAR-SEASONS OF P:DENVER IS P:SPRING ]

```

Figure 35: The execution.record after handling example in Example 1 construct an explanation using the strategy incomplete.knowledge.base. The constraint partitioning algorithm is invoked, and uses a *primary-constraint-set-size* of 1 to produce 7 partitions (there are 7 SWSSes in C), which are shown in full in Appendix D. Three of the original partitions survived the filtering that was described in the algorithm above. The user is presented with the resulting KB additions that are proposed to resolve the exception, and chooses to add the fact (a popular.seasons of denver is spring) in order to make the resource choice Denver consistent with the existing constraints.

Figure 35 shows the contents of the SPANDEX execution.record after the exception has been successfully handled. All the proposed KB additions that resulted from processing the three valid partitions are shown, along with the

```

=====
Chosen Partition for Node: <SELECT.A.SITE>FOR.AGENT<00>
=====
(PARTITION 1:)
Primary-constraints: #[fff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE IS ?<SELECT.A.SITE>SEASON ]
Primary-bindings: ((?<SELECT.A.SITE>SITE . BOSTON) (?<SELECT.A.SITE>SEASON . FALL))
                  ((?<SELECT.A.SITE>SITE . BOSTON) (?<SELECT.A.SITE>SEASON . SPRING))
                  ((?<SELECT.A.SITE>SITE . PITTSBURGH) (?<SELECT.A.SITE>SEASON . SPRING))
                  ((?<SELECT.A.SITE>SITE . DENVER) (?<SELECT.A.SITE>SEASON . WINTER))
Secondary-constraints: #[fff SIGMOD IS IN CLASS CONFERENCE ]
                    #[fff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS UNIVERSITY ]
                    #[fff ?<SELECT.A.SITE>SITE IS IN CLASS CITY ]
                    #[fff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY IS ?<SELECT.A.SITE>SITE ]
                    #[fff A MONTH OF SIGMOD IS ?<SELECT.A.SITE>MONTH ]
                    #[fff A SEASON OF ?<SELECT.A.SITE>MONTH IS ?<SELECT.A.SITE>SEASON ]
Secondary-bindings: ((?<SELECT.A.SITE>SEASON . SPRING) (?<SELECT.A.SITE>MONTH . APRIL)
                    (?<SELECT.A.SITE>SITE . DENVER) (?<SELECT.A.SITE>UNIVERSITY . UNIVERSITY.OF.COLOPADO))
=====

```

Figure 36: The selected constraint partition for example 1

particular amendment chosen. The actual constraint partition that generated the selected KB addition is pictured in Figure 36.

SPANDEX rates the possible KB additions when presenting them to the user based on the following heuristic:

Heuristic: When a choice of resource produces an inconsistency, and there are a number of possible additions to the knowledge base that will restore consistency, prefer those additions that refer to the resource object to facts that refer to other objects in the knowledge base.

The idea behind this heuristic is that since the resource choice was not initially selected, the system should attempt to “massage” features of the controversial choice rather than other facts that are less in focus, if possible.

Example 2: To illustrate how the algorithm may fail on the default **primary-constraint-group-size** of 1, but be able to produce a solution when the size of the primary constraint group is increased, we show what would happen if the user wanted to choose Pittsfield as the site to hold the conference. Notice that we have no knowledge of popular-seasons for Pittsfield nor of any universities that are located there (refer back to Figure 34). Thus, no single simple assertion of one of the bound SWSSes will suffice to satisfy all the constraints. The algorithm can find no solution for **primary-constraint-group-size** of 1. When the value of the parameter is increased to 2, 21 partitions are constructed, one of which produces possible solutions. The solution is shown in Figure 37 and the constraint partition which produced the solution is shown in Figure 38.

Discussion: It is apparent from looking at the proposed solution that there is a potential problem. The solution involves asserting that (a locale of Tufts.University is Pittsfield), where Tufts.University is already known to be in Boston. Common sense tells us that a university cannot be located in two places at once³³. Thus, the assertion of this solution could produce a further inconsistency, e.g., a violated cardinality constraint. One way to avoid this would be to treat the solution as a *modification* of the knowledge base rather than as an *addition* when the proposed change is being made to a field of an object that is constrained to have a single value. In this case, the locale of Tufts.University would actually be *changed* to Pittsfield.

Limitations of this approach: In general, when we have an inconsistent set of constraints it is possible that the knowledge base is either *incomplete* or

³³Unless of course, there is a another branch of the same university in another town.

```

III (Output) The "EXECUTION.RECORD" Unit in SPANDEX Knowledge Base
Unit: "EXECUTION.RECORD" in knowledge base SPANDEX
Created by BROVERMAN on 3-6-89 13:07:56
Modified by BROVERMAN on 6-14-90 12:42:24
Member Of: ENTITIES in GENERCUNITS

Maintains information about the current execution cycle.

Own slot: EXCEPTION.TYPE from "EXECUTION.RECORD"
Values: S:RESOURCE-CHOICE-INCONSISTENT-WITH-EXPECTATIONS

Own slot: OVERALL.EXECUTION.RESULT from "EXECUTION.RECORD"
Values: P:RESOLVED-EXCEPTION

Own slot: PROPOSED.KB.ADDITIONS from "EXECUTION.RECORD"
Values: #[Wff (A P::POPULAR-SEASONS OF P::PITTSFIELD IS P::SPRING) AND (A P::LOCALE OF P::UNIVERSITY.OF.COLORADO IS P::PITTSFIELD) ],
#[Wff (A P::POPULAR-SEASONS OF P::PITTSFIELD IS P::SPRING) AND (A P::LOCALE OF P::TUFTS.UNIVERSITY IS P::PITTSFIELD) ],
#[Wff (A P::POPULAR-SEASONS OF P::PITTSFIELD IS P::SPRING) AND (A P::LOCALE OF P::CARNEGIE.MELLON.UNIVERSITY IS P::PITTSFIELD) ]

Own slot: RETURNED.BINDINGS from "EXECUTION.RECORD"
Values: ((#[Var P:(SELECT.A.SITE)SITE] . #[Wff P::PITTSFIELD ]))

Own slot: SELECTED.AMENDMENT from "EXECUTION.RECORD"
Values: S:ADD.TO.KB.TO.SATISFY.CONSTRAINTS

Own slot: SELECTED.EXPLANATION from "EXECUTION.RECORD"
Values: INCOMPLETE.KNOWLEDGE.BASE.RECORD113

Own slot: SELECTED.KB.ADDITION from "EXECUTION.RECORD"
Values: #[Wff (A P::POPULAR-SEASONS OF P::PITTSFIELD IS P::SPRING) AND (A P::LOCALE OF P::TUFTS.UNIVERSITY IS P::PITTSFIELD) ]

```

Figure 37: The execution.record after handling exception in Example 2


```

=====
Chosen partition for Node: <SELECT.A.SITE>FOR.AGENT<001>
=====
Primary-constraints: #[(? A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE
                        IS ?<SELECT.A.SITE>SEASON ]
                    #[(? A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY
                        IS ?<SELECT.A.SITE>SITE ]
Primary-bindings: ((?<SELECT.A.SITE>UNIVERSITY . TUFTS.UNIVERSITY)
                  (?<SELECT.A.SITE>SITE . BOSTON) (?<SELECT.A.SITE>SEASON . FALL))
                  ((?<SELECT.A.SITE>UNIVERSITY . TUFTS.UNIVERSITY)
                  (?<SELECT.A.SITE>SITE . BOSTON) (?<SELECT.A.SITE>SEASON . SPRING))
                  ((?<SELECT.A.SITE>UNIVERSITY . CARNEGIE.MELLON.UNIVERSITY)
                  (?<SELECT.A.SITE>SITE . PITTSBURGH) (?<SELECT.A.SITE>SEASON . SPRING))
                  ((?<SELECT.A.SITE>UNIVERSITY . UNIVERSITY.OF.COLORADO)
                  (?<SELECT.A.SITE>SITE . DENVER) (?<SELECT.A.SITE>SEASON . WINTER))
Secondary-constraints: #[(? SIGNED IS IN K::CLASS CONFERENCE ]
                      #[(? ?<SELECT.A.SITE>UNIVERSITY IS IN K::CLASS UNIVERSITY ]
                      #[(? ?<SELECT.A.SITE>SITE IS IN K::CLASS CITY ]
                      #[(? A MONTH OF SIGNED IS ?<SELECT.A.SITE>MONTH ]
                      #[(? A SEASON OF ?<SELECT.A.SITE>MONTH IS ?
                          <SELECT.A.SITE>SEASON ]
Secondary-bindings: ((?<SELECT.A.SITE>SEASON . SPRING) (?<SELECT.A.SITE>MONTH . APRIL)
                    (?<SELECT.A.SITE>SITE . PITTSFIELD)
                    (?<SELECT.A.SITE>UNIVERSITY . UNIVERSITY.OF.COLORADO))
                    ((?<SELECT.A.SITE>SEASON . SPRING) (?<SELECT.A.SITE>MONTH . APRIL)
                    (?<SELECT.A.SITE>SITE . PITTSFIELD)
                    (?<SELECT.A.SITE>UNIVERSITY . TUFTS.UNIVERSITY))
                    ((?<SELECT.A.SITE>SEASON . SPRING) (?<SELECT.A.SITE>MONTH . APRIL)
                    (?<SELECT.A.SITE>SITE . PITTSFIELD)
                    (?<SELECT.A.SITE>UNIVERSITY . CARNEGIE.MELLON.UNIVERSITY))
=====

```

Figure 38: The selected constraint partition for Example 2

incorrect. The approach presented is a local solution to the *incomplete* problem. That is, information can be added to the KB to satisfy the constraints. It is local because it only considers a subset of the constraints that might be applicable, i.e., the ones in the node, pertaining to the context of execution. There may be other constraints which are not being considered, that derive from other sources; e.g.; the specifications of object classes which are disjoint, static integrity constraints about objects and their possible property values, limitations on the cardinality of field values, and other domain theories such as basic number theory (e.g., if x is less than y it can't also be greater than y , etc.).

Since the constraint partitioning approach described here considers only the dynamic (local) constraints on the resource being chosen, we run the risk of violating an arbitrary static (global) constraint in the knowledge base as a result of implementing the amendment³⁴. Our solution, therefore, may propose amendments that may generate additional violations as well as those which may be successfully implemented.

The limited approach described here is able to answer the following question: "When the current knowledge base fails to satisfy a conjunct of N particular constraints, what knowledge can be added to the domain so that they would be satisfied?" The more general form of the constraint satisfaction problem here is really: "When the current knowledge base fails to satisfy a conjunct of N particular constraints, what knowledge can be added to the domain so that **all known constraints** are satisfied?" The general solution

³⁴The example of a subsequently violated cardinality constraint in the discussion following Example 2 is just one such example of the general class of potential problems.

to this problem is similar to the problem of determining *inherent unsatisfiability* in that all of the constraints in the knowledge base would have to be represented and manipulated in a uniform fashion.

In order to improve the heuristic "hit-rate" of the partitioning algorithm, this problem would have to be addressed more generally in future implementations. If a more global method were available for determining *inherent satisfiability*, it would be applied to the total set of constraints before any partitioning is done. The partitioning approach then would only be applicable if the original set of constraints passed this filtering. The method could then be applied again during the partitioning process as proposed changes are being constructed. Again, this is an area for extensions and future work.

6.4.5.2 Adding to the Knowledge Base in Response to Violated Constraints

When actions have been performed that cause the actual violation of constraints, it is too late to intercept the effects of the violation before they have been reflected in the plan. These cases therefore require different handling than that provided by the constraint partitioning approach. Such cases are flagged in response to the detection of *inconsistent worlds* by KEE, and result from the violation of static object constraints. The exception classifier designates these occurrences as *plan-network-inconsistent-during-planning* or *plan-network-inconsistent-after-execution* exceptions,³⁵ depending on the status of the plan when the exception actually occurs.

³⁵Only a subset of these exceptions involve static constraints, recall that plan networks can also be inconsistent as a result of inability to satisfy dynamic plan constraints.

Since a violation has already been detected by the system in this case, an informant is stored with the recorded justification for the violation, indicating the static constraint that was violated. One possible response for this exception is to extend the knowledge base to satisfy the constraint. A general implementation of the `add-to-kb-to-satisfy-constraints` amendment when it pertains to an `inconsistent-world` exception has not been completed; however, the current implementation is able to determine when the coercion of the violator object to another valueclass is allowable and can resolve the exception. An example of how this is done is described in the scenario in Section 7.1.1.

6.4.5.3 *Another Approach: Modifying the Constraints to Fit the KB*

In the previous two sections, we described amendments that remove the source of the constraint violation (the “violator”) by adding information, in the form of facts, to the knowledge base. In this section we describe an approach to resolving constraint violations by modifying the constraints themselves, rather than by adding facts in the knowledge base to make them satisfiable. In general, this approach involves the relaxation of violated constraints (the “violatees”) so as to eliminate the violation upon subsequent reevaluation. We have identified a number of possible ways to do this:

1. Disjunct addition (add a disjunct to a constraint specification);
2. Conjunct to disjunct conversion (convert a conjunct of sub-constraints to a disjunct of sub-constraints);
3. Conjunct elimination (eliminate one or more conjuncts from the a constraint specification);

4. Taxonomic generalization (replace a class specification with a more general superclass);
5. Taxonomic expansion (replace a class specification by the set of classes which are subsumed by the original class specification);
6. Range extension (extend a numeric or other ordinal range);
7. Constraint elimination (eliminate the constraint);

An example of applying the `modify-static-constraints-to-fit-kb` amendment with the technique of disjunct addition is illustrated in Scenario 1 in Section 7.1.1.

6.5 Explanation Presentation

In order to intelligently control the application of plausible inference (PI) rules and the presentation and selection of the resulting explanations, we use a set of heuristics similar to those applied to plan recognition problems [12]:

- **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 specification to one proposing more modifications.

A threshold is set to limit the number of explanations produced. The most likely explanations are presented to the user in an interactive fashion and a choice is requested. If none of these explanations is 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 treats the exception as a true error, invoking general replanning measures to counteract the negative effects. 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.

6.6 Limitations of the Implementation

The adequacy and performance of the system described here is by definition limited by the adequacy and performance of the testbed planning system upon which it is built. The algorithms employed during exception handling use the same underlying static and dynamic representations of the current plan and domain that is being manipulated by the planner. The primary shortcomings can be made apparent by an analysis of the representation and handling of constraints.

Our treatment of constraints

One of the major sources of difficulty encountered while implementing the prototype was our limited and non-uniform representation and maintenance of

constraints. These limitations became apparent when working on resolution strategies for the agent-resource-choice-inconsistent-with-expectations exception type as well as with inconsistent plan networks arising from routine planning or execution. The handling of these exceptions requires the identification and potential modification of the violated constraints.

This identification and the appropriate propagation of subsequent modifications is nontrivial due to the way that constraints are currently represented and maintained in our system. Constraints in our knowledge-based planner are specified initially in static activity and object descriptions; instantiations of the activity-related static constraints are posted dynamically on the affected parts of an evolving plan by the plan generation algorithms. When a violation occurs, if a dynamic constraint instantiation is involved, the planner has no record of the original source of the constraint, i.e., it cannot tell if the constraint represents a precondition or if it governs the entire activity decomposition. Therefore it cannot be easily and correctly determined if there are other nodes in the plan which may also be governed by duplicate instantiations of the same constraint.

Specifically, if a constraint c is specified on a goal variable v of an activity, and that variable is manipulated by s subgoals, instantiations of c will be posted dynamically on the nodes ($n_1 \dots n_s$) that represent the subgoals of the activity when it is chosen for expansion of a higher-level goal node. However, at run-time, if the constraint c is violated at node n_1 and is subsequently relaxed by the exception handler, that relaxation should be propagated to the other instantiations of c posted on nodes $n_2 \dots n_s$. Since POLYMER does not maintain a record of the source of a dynamic constraint, nor link together the dynamic

instantiations that were generated from the same initial constraint source, proper propagation requires significant recomputation by referring back to the static activity descriptions and recomputing the mappings to all of the relevant nodes. In addition, if it is desirable to make the relaxation of the violated constraint more permanent (i.e., make a change to the static task description so it will be used in the future), there is no obvious way to determine the static source of the constraint. A dynamically posted constraint originally comes from a clause of a particular static activity definition, but any subsequent violation of the instantiation points to a generic *condition* (represented as a well-formed formula) with no attached history or justification.

In addition, whenever bindings are established³⁶ and propagated, constraints which refer to the affected variables are bound with the established values. Therefore, the original form of the constraint is not maintained, making it difficult to institute amendments which involve the substitution of variable values.

These limitations are obstacles to easily implemented corrective measures. The source of these inadequacies is that the current planning implementation fails to record and maintain justifications for all events of interest, such as plan expansion decisions, the posting of constraints, variable binding decisions, and the interpretation of user actions within an interactive planning framework.

A proposal for a redesign of the current system that incorporates a plan network maintenance system (PNMS) based on an assumption-based truth maintenance system, is described in [7]. A reimplementaion of the planner

³⁶The situations in which this can happen was described in Section 6.2.2 on page 153.

testbed which incorporates a unified treatment of planning events via justification (as described in [7]) would enable the determination of the source of variable bindings and constraints, as well as the retraction and substitution of variable bindings.

In general, constraints might also be better represented as first-class objects, such as in [26], so that links can be maintained to duplicate instantiations of a given constraint, histories can be maintained as to the source of incremental constraint binding, and information can be recorded about relaxations that are instituted in response to exceptions that are handled. In addition, it would also be useful and necessary to maintain information about the priorities of constraint relaxations that can be predetermined or forbidden. For example, some constraints should never be relaxed (e.g. one can never maintain a negative balance in their checking account³⁷), or it may be desirable to assign a penalty rating scheme to different types or degrees of relaxation, as in [26]. The current representation of constraints, and the lack of information maintained about planning decisions is too simplistic for the sophisticated plan reformulation that we would ideally like to accomplish.

On a more philosophical note, it can also be argued that the current system (as well as other comparable systems, for that matter) does not provide a clear definition of what actually constitutes information that can be used to constrain planning and execution. Exactly what constitutes a "constraint" in an interactive planning system? The argument could be made that every aspect of the domain description, including the object type hierarchy, the object part hierarchy, object field value restrictions, iteration, temporal and

³⁷Unless you live in Israel, where this is the rule rather than the exception...

causal constraints among subgoals, and even the form of goal statements themselves, constitutes an implicit constraint on the eventual complete plan that will accomplish a task goal. We currently do not have a complete or uniform language to express and process all this information as constraints. For instance, we do not represent or use general integrity constraints [8], or *policies* [44], that apply to overall entities (aggregate constraints) rather than just to fields of objects. The work of Fox in the ISIS system provides a uniform formulation of these types of knowledge as first order constraint objects, albeit not within a planning framework.

Furthermore, our current plan specification formalism does not allow different-grain sizes for intervals over which constraints that are specified in activities must be maintained; the only types of constraints which can be expressed are preconditions (must hold in order for an activity to be chosen for achieving a goal), or generic (must hold for the entire duration of the activity in which it was defined). Experience with the current domain and implementation has shown that often a constraint is only relevant for a small interval of the decomposition of an activity³⁸. Also, the implementation of temporal constraints (represented by simple initial orderings with no justification) differs from that used for causal constraints (active values on database objects), which is different still than the mechanisms used to implement domain object constraints (embedded in KEE), and that used for activity variable constraints (which are explicitly checked at predetermined times by the planner itself). This nonuniformity in underlying representation and handling does not lend

³⁸In fact, often it is not even possible to evaluate a constraint on a substep of an activity since, with delayed binding of resources, all bindings may not be established until several more layers of expansion have occurred.

itself to a clean treatment of constraint handling, propagation, and relaxation. Since the exception handling component of an interactive planning system is heavily dependent on the underlying representation of all elements of the basic planning paradigm, a subsequent reimplementaion of the underlying planning testbed as described above would be necessary in order to overcome most of the limitations described in this section.

What constraints should be used to guide the planning?

One of the reasons that exceptions arise during planning is that POLYMER only uses a subset of all available constraints (described in Section 5.1.4) to guide the planning process. Variable bindings resulting from the unification used in goal matching, along with the dynamic instantiations of static constraints in the activity schemas (represented as conditions on nodes) are the only constraints explicitly guiding activity selection, step ordering, and conflict detection and resolution during planning.

POLYMER does not exploit static object constraints during planning. The modifications introduced by SPANDEX which have been described in this thesis extend the constraint processing to utilize some of this static information in a limited fashion. The *suggestions* facility uses object field valueclass specifications to preempt bad resource choices, and to further filter the activity selection during plan expansion. Since static constraints are not fully exploited during planning, the result is that they cause run-time planning violations (i.e., inconsistent world during planning), which were not foreseen.

The approach described in this thesis gives methods for handling such violations through constraint relaxation or the addition or modification of domain knowledge which can satisfy the failed constraint. An alternative would be to move more knowledge into planning, to avoid more exceptions. Since knowledge-based planning is by definition a knowledge-intensive process, this alternative may involve significant computation at every plan step. A trade-off exists between a computation-intensive paradigm (use all the knowledge) and considerable exception handling (use only the most basic knowledge), and it is not clear where the line should be drawn.

Another limitation of the current system is that the search for the contributions of an unexpected action can only identify relationships in which the results of the performed action and the planner's expectations are represented at the same granularity level. Specifically, the algorithms provided do not do partial matching, so the system would be unable to detect that an unexpected step achieved the "a" part of a later goal (and a b c). The ability to provide explanations based on partial matching is an obvious area for the extension of the system.

In addition, the explanations posed by SPANDEX are not guaranteed to be correct, and additional problems can result from the incorporation of an explanation. This approach described here is a heuristic one that allows for the graceful degradation of a necessarily rigid planning system operating in an incomplete and incorrect domain.

Exception classifications are not fully exploited

The identification of the different levels of goal and activity matching represented by the dimensions described in Section 6.2.3 can be used to rank explanations by closeness of match, and to provide information about how much information in the domain description would have to change in order to accommodate the suggested explanation. However, the current implementation only considers a partial set of these classifications (e.g. Classifications 1.1, 1.2, 2.1-2.7, 3.1, and 4.1). The scenarios which have been run in our experimental evaluation have produced a manageable number of complete explanations within a reasonable amount of time, so the fine-tuning represented by the other categorizations have not yet been incorporated into the current prototype.

The “completeness” of explanations and the additional opportunity for knowledge acquisition

Explanations can be either *complete* or *incomplete*. A complete explanation results from the application of a PI rule whose state specification S holds completely in the current world model, while an incomplete 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 the validation of incomplete explanations. An attempt should be made to confirm missing indicators through interactive dialogue, producing additional plausible explanations while adding to the domain model. This is not yet implemented, but should be straightforward, guided by the components of the stored incomplete explanation.

CHAPTER 7

DEMONSTRATION AND EVALUATION

The evaluation of an interactive planning system is not an easy task. In [13], Chapman has shown that when specific limitations on the formulation of a planning system are enforced, nonlinear hierarchical planning can be proven to be both complete and correct. However, in order to approximate the use of a planner in a more practical setting and realistic environment, we have employed a representation for plan operators and adopted a set of operating assumptions that do not conform to the limited formulations required in order to demonstrate these theoretical properties. In particular, we do not adopt the closed-world assumption, must be prepared to handle the unexpected effects of actions, and must delay complete knowledge until execution since information used to guide the planning may be elicited from the user.

Therefore, even if it were possible, it is obviously futile to propose producing an exception handling component for this underlying system which is guaranteed to be complete (it will produce an explanation if there is one) or correct (every explanation it will produce will be correct). The best that we can hope for in this "scruffy" setting is to provide techniques that can successfully capture and model some of the exceptions that arise, applying reasoning and modification techniques to an inconsistent partial plan to produce a suc-

cessor partial plan from which planning and execution can be resumed. Our approach is an heuristic one, and our evaluation therefore consists of a demonstration of how the concepts proposed in this thesis have been implemented and successfully applied to improve the performance of our interactive planning system.

In this chapter we present a demonstration and evaluation of our approach to exception handling in an interactive planning framework. This demonstration consists of two components. The first component assumes the acceptance of our exception classification. In Section 7.1, we describe implemented scenarios that exercise the major aspects of the SPANDEX system which have been described in this thesis. The scenarios constitute existence proofs for the exception types outlined; their successful handling as described show that the system is able to respond to such exceptions, resulting in successor states to the initial inconsistent states detected. Planning and execution can be subsequently resumed. The scenarios thus demonstrate the improved capability of the planning system as a result of adding exception handling capabilities. We then follow this demonstration with a brief discussion of the cost complexity of explanation generation (Section 7.2).

The second stage of our evaluation consists of interviews of individuals who have had experience with a selected task (we chose the domain of conference organization). We recorded feedback about the ways in which a plan in the chosen domain can deviate. The data which was obtained in this fashion was used to support our hypothesis about the types of exceptions that can occur. The survey which was used is presented in Appendix F, and an analysis of the results is given in Section 7.3. The complete description of the domain

which was used by planner for the scenarios described in this chapter is given in Appendix E.

7.1 SPANDEX Scenarios

In this section we present examples from the domain of conference planning. The examples illustrate the application of several of the defined rationales and the opportunities for knowledge acquisition, through (1) explicit invocation (as triggered by a gap in the domain model), (2) the incorporation of complete explanations. The first three scenarios are presented in full detail, including variations, while for the remaining scenarios, we present figures for illustration, including a brief selection of planner messages that describe the handling of the exception.

7.1.1 Scenario 1

This scenario illustrates the handling of an inconsistent-world-during-planning exception¹. We show how the exception is detected during what appears to be “normal” planning. The violation of a constraint in the underlying domain model triggers the exception, and we show three possible ways in which the exception can be resolved. In the first two cases, the violated constraint is generalized in order to permit the assertion of a new unanticipated value. The third resolution strategy shows how information can be added to

¹An exception of type inconsistent-world-after-execution would be handled in exactly the same fashion.

the domain model to eliminate the violation. In the first two cases, the violated constraint is targeted for modification; in the third case, an offending feature of the "violator" (the value that triggered the assertion) itself is chosen for modification.

The invocation of any of these resolution strategies achieves two important goals: (1) plan consistency is restored and the planning can continue on from this point without replanning, and (2) a modified domain model results, so that the planner is able to properly anticipate a wider range of circumstances in the future.

The scenario begins in a situation in which papers that are to be presented at a conference are to be chosen. The top-level goal posted to the planner states that the papers-chosen-for-presentation feature of the planning-in-uncertain-environments workshop should be set to done. Two activities are available to the planner which can achieve this goal: judge-papers-for-presentation and invite-papers-for-presentation, which are shown in Figure 39².

Figure 40 shows the initial configuration for the scenario. This figure presents a snapshot of the developer's interface, which is used by system developers when running and testing the system. The display is divided up into screens, each of which shows dynamic information pertinent to the planning process. For example, the Plan Network screen maintains a display of the current plan network, the plan node information screen displays information about the node currently selected for processing by the planner, Performed Actions shows the executable actions that have been performed, and the Current

²The descriptions are presented using the external domain description language which is parsed when a polymer application is read in (see Appendix A).

Bindings screen indicates bindings that are possible for variables in the goal of the currently selected node (in this case there are none, since the goal of the selected node has no variables). The Planner Messages window is used to maintain a history of actions taken by the planner and by SPANDEX. Various pop-up messages appear on the display when user interaction is required. In this example, a pop-up window is shown requesting the user to select from among the two possible activities known to achieve the posted goal. In the remainder of this scenario (and other scenarios presented in this chapter), this interface will only be shown in part. In particular, the plan network window and the pop-up windows will appear in figures for illustration, where relevant.

The agent chooses *invite-papers-for-presentation* since this conference is really being run as an informal workshop and the submissions will not be formally refereed. The planner expands the plan to reflect this choice, and splices in the sub-plan-network that represents the substeps involved in inviting the submissions. The submissions to be invited are selected (see Figure 41). To model the state achieved by the second substep in the activity's decomposition, the planner asserts that the decision of each submission selected during the first substep (in this case, only *submission.004*) will be invited (note the activity decomposition in Figure 39).

Note that the domain specification states that the decision on a submission must be one of *accept*, *reject*, or *accept-with-modification* (see Figure 42). Since *invited* is not among the allowable values, an inconsistent world results from the assertion, and an exception of type *inconsistent-world-during-planning* is signalled (see Figure 43). Note that this exception could not be intercepted earlier and handled with the constraint partitioning techniques described pre-

```

{ACTIVITY judge.papers.for.presentation
 *comment: "Top level activity for conference presentation selection."
 *agents:?(program-chair program.chair)
 *goal: papers.chosen.for.presentation?(meeting technical.meeting), done)
 *input-goal-vars: ?meeting
 *decomposition: (submissions-solicited
                  issued.call.for.papers(?meeting,?(call call.for.papers)))
                  (has-submissions submissions.received(?program-chair, ?subms))
                  (submissions-reviewed review.status(?submission, reviewed))
                  (decisions-reached decision(?submission, ?decision))
                  (authors-notified and(paper(?submission, ?paper)
                                         authors(?paper ?author)
                                         notified(?author, yes)))
 *control: (BEFORE submissions-solicited has-submissions)
            (BEFORE has-submissions submissions-reviewed)
            (BEFORE submissions-reviewed decisions-reached)
            (BEFORE decisions-reached authors-notified)
            (REPEAT submissions-reviewed for ?submission in ?subms)
            (REPEAT decisions-reached for ?submission in ?subms)
            (REPEAT authors-notified for ?submission in ?subms)
 *constraints: current.meeting(?program-chair, ?meeting)
}

```

```

{ACTIVITY invite.papers.for.presentation
 *comment: "An alternate way to get papers -- invite from authors."
 *agents:?(program-chair program.chair)
 *goal: papers.chosen.for.presentation?(meeting technical.meeting), done)
 *input-goal-vars: ?meeting
 *decomposition: (solicit-submissions
                  submissions.received(?program-chair,?subms))
                  (submissions-invited decision(?submission, invited))
 *control: (BEFORE solicit-submissions submissions-invited)
            (REPEAT submissions-invited for ?submission in ?subms)
 *constraints: current.meeting(?program-chair, ?meeting)
}

```

Figure 39: Possible activities to achieve (a papers-chosen-for-presentation (?meeting, done))

POLYMER 2.0

KEE Desktop 2 - Lisp Listener

CONFERENCE (Select-Papers-From-Submissions-Scenario-2<001>)

Subsystems

Domains	Planner	Display
---------	---------	---------

<From: POLYMER>
 Select activity to accomplish
 (A PAPERS.CHOSEN.FOR.PRESENTATION OF PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS DONE)
 in activity *Select-Papers-From-Submissions-Scenario-2*.
 JUDGE.PAPERS.FOR.PRESENTATION
 INVITE.PAPERS.FOR.PRESENTATION

Working Window

Plan network

```

        graph TD
            A["<*S-P-F-S-S-2*>START<001>"] --- B["<*S-P-F-S-S-2*>TOP-LEVEL-GOAL<001>"]
            B --- C["<*S-P-F-S-S-2*>FINISH<001>"]
            
```

Planner messages

10: -----
 11: NEXT NODE: <*SELECT-PAPERS-FROM-SUBMISSIONS-SCENARIO-2*>TOP-LEVEL-GOAL<001>
 12: -----
 13: PROCESSING GOAL NODE: <*SELECT-PAPERS-FROM-SUBMISSIONS-SCENARIO-2*>TOP-LEVEL-GOAL<001>

Next possible nodes

Plan node information	Selected activity
GOAL Node (1): <*SELECT-PAPERS-FROM-SUBMISSIONS-SCENARIO-2*>TOP-LEVEL-GOAL<001> Primary agent: BROVERMAN Its status is: UNSEEN Start time: [10:00:00,] Finish-time time: [, 12:30:00] Its goal is: (A PAPERS.CHOSEN.FOR.PRESENTATION OF PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS DONE) Possible activities to accomplish this are: JUDGE.PAPERS.FOR.PRESENTATION with (?<judge.papers.for.presentation>meeting = Planning.In.Uncertain.Environments;) INVITE.PAPERS.FOR.PRESENTATION with (?<invite.papers.for.presentation>meeting = Planning.In.Uncertain.Environments;)	Current bindings <hr/> Performed actions <hr/> Status <p style="text-align: center;">WAITING</p>

Figure 40: Initial goal posed to planner in Scenario 1

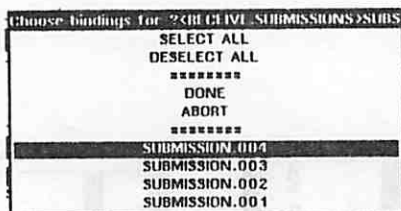


Figure 41: Choosing submission.004 from among possible submissions

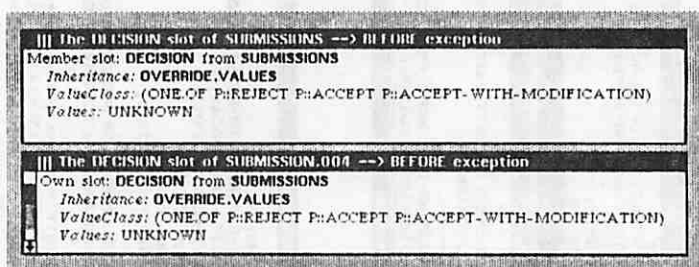


Figure 42: The decision slot of the submission and submission.004 units prior to constraint relaxation

viously in Section 6.4.5, since it did not occur during user interaction. An inconsistent world has already resulted from automated planning actions and must be resolved in a different fashion³.

³This particular exception could have been avoided if a mechanism existed for an exhaustive consistency check of the initial domain description prior to execution. During this check it may have been noted that an activity tries to assert an illegal value (invited) for the property (decision) of an object (submission), and the techniques described here could be applied at load time. However, often there are variables in the original descriptions and the value that is to be asserted during planning may not be established until some arbitrary point during the planning and execution process. In addition, the cost of consistency checking may be great, and the description of domains may proceed incrementally. Therefore, this general class of exception at run-time can still arise and must be handled.

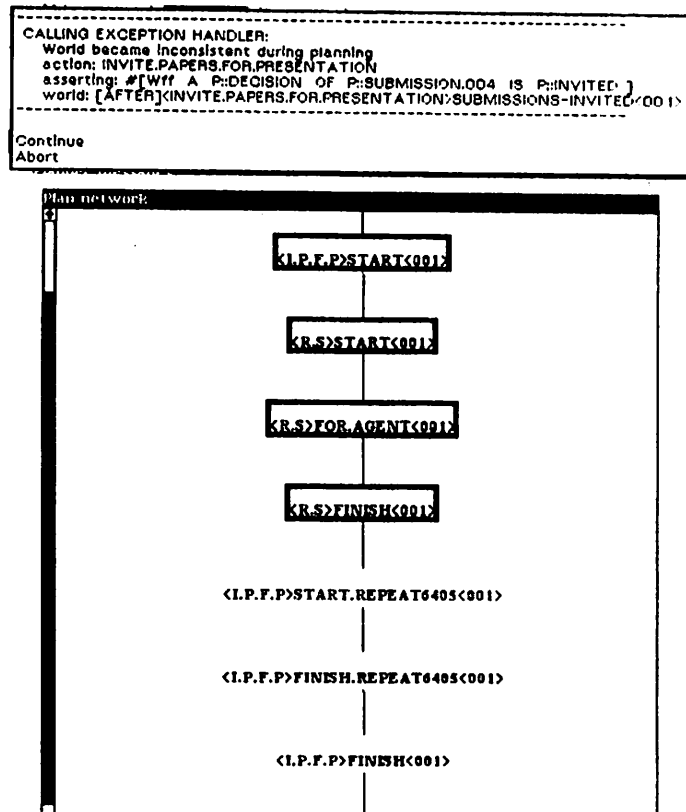


Figure 43: Invoking the exception handler, with the current inconsistent plan network

The explanation basis designated for this type of exception focuses on possible imperfections in the domain. Two establishment strategies are known to provide evidence for this explanation basis, and are shown in Figure 7.1.1. The first of these, *incomplete-knowledge-base*, assumes that the knowledge base is incomplete, and will propose ways to extend the domain to satisfy the constraint. Of the two *inconsistent-state-specs* relevant for this strategy, *iss134* is applicable (since inconsistent worlds have been detected), while *iss133* is not (since we are not at a point of user interaction).

The second strategy for the imperfect-domain explanation basis, *incorrect-static-knowledge-base*, assumes that part of the underlying description of the domain is incorrect and requires modification. Specifically, the single *inconsistent-state-spec* applicable here (*iss136*) will focus on ways to relax the violated constraint in order to eliminate the inconsistency. We illustrate the methods used for both of these strategies in the discussion which follows.

Constraint relaxation: Generalizing a constraint

We first show a method in which the violated constraint may be relaxed in order to accommodate the offending value in the assertion. SPANDEX has detected the violation of a static domain constraint, specifically the *valueclass constraint*⁴ on the decision field of the *submission.004* object. The *incorrect-domain-triggered-by-static-constraint-violation?* function that has been invoked as an establishment function for *iss134* gathers evidence about the violation;

⁴Valueclass constraints in KEE are a set of general constraints that specify how to determine the validity of values that are put into slots of objects. The different types of valueclass constraints available in the implementation were discussed in Section 5.4.

***Rationale*: IMPERFECT.DOMAIN**

***Establishment.strategies*:**

INCOMPLETE.KNOWLEDGE.BASE

***Inconsistent-state-spec*: ISS133**

***Indicators*:**

IND: Establishing-fn: INCOMPLETE-DOMAIN-TRIGGERED-BY-RESOURCE-SELECTION?
When applicable?: (VIOLATION-DETECTED-DURING-USER-INTERACTION?)

***Inconsistent-state-spec*: ISS134**

***Indicators*:**

**IND: Establishing-fn: INCOMPLETE-DOMAIN-TRIGGERED-BY-STATIC-
CONSTRAINT-VIOLATION?**
When applicable?: (INCONSISTENT-WORLDS-DETECTED?)

***Amendments*:**

ADD.TO.KB.TO.SATISFY.CONSTRAINTS

Implement: (ADD-TO-KB-TO-SATISFY-CONSTRAINTS)

INCORRECT.STATIC.KNOWLEDGE.BASE

***Inconsistent-state-spec*: ISS136**

***Indicators*:**

**IND: Establishing-fn: INCORRECT-DOMAIN-TRIGGERED-BY-STATIC-
CONSTRAINT-VIOLATION?**
When applicable?: (INCONSISTENT-WORLDS-DETECTED?)

***Amendments*:**

MODIFY.STATIC.CONSTRAINTS.TO.FIT.KB

Implement: (MODIFY-STATIC-CONSTRAINTS-TO-FIT-KB)

Figure 44: Strategies and indicators for the Imperfect-domain explanation basis

e.g., the assertion which triggered the violation and the constraint which was violated. The strategy then proposes to replace the valueclass specification with a more general version of the constraint, to account for the violating assertion. In order to do this, SPANDEX determines the object in the domain object hierarchy that first specified the violated constraint. Since submission.004 inherited the constraint from the more general class object submissions, it is in the submissions object that the constraint should be modified. Thus, submissions is established as the source-unit to fix (see the fix in Figure 46).

Now that the violated constraint and its source have been identified, a determination is made as to whether it is possible to generalize the constraint, and if so, how. The amendment modify-static-constraints-to-fix-kb is invoked to propose and implement a way to relax the violated constraint to accommodate the new value invited. In general, the type of relaxation proposed depends upon the form of the violated constraint. In this particular case, the constraint is a simple list of allowable values (reject, accept, accept-with-modification), so the action of the amendment is to add the new value invited to the list. A new, more general constraint results, and the original constraint is replaced with the generalized version (see Figure 45). The implemented explanation record which reflects the reasoning just described is shown in Figure 46.

Another representation. To further illustrate how SPANDEX can generalize violated constraints to eliminate inconsistency, consider another representation for the valueclass specification of the decision field of a submission. Suppose that each type of decision was actually represented as an abstract object in the domain. The members of the more general class objects possible-decisions, possible-decisions-on-conference-papers and possible-decisions-

<pre> The DECISION slot of SUBMISSIONS --> ALL exception Member slot: DECISION from SUBMISSIONS Inheritance: OVERRIDE.VALUES ValueClass: (ONE OF P:INVITED P:REJECT P:ACCEPT P:ACCEPT-WITH-MODIFICATION) Values: UNKNOWN </pre>
<pre> The DECISION slot of SUBMISSION.004 --> ALL exception Own slot: DECISION from SUBMISSION.004 Inheritance: OVERRIDE.VALUES ValueClass: (ONE OF P:INVITED P:REJECT P:ACCEPT P:ACCEPT-WITH-MODIFICATION) Values: P:INVITED </pre>

Figure 45: The decision slot of the submission and submission.004 units after valueclass modification

<pre> (Output) Current explanation in domain CONFERENCE Unit: INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD 158 in knowledge base SPANDEX Created by BROVERMAN on 6-21-90 17:59:12 Modified by BROVERMAN on 6-21-90 17:59:12 Member Of: INCORRECT.STATIC.KNOWLEDGE.BASE </pre>
<pre> Own slot: AMENDMENTS from INCORRECT.STATIC.KNOWLEDGE.BASE Values: MODIFY.STATIC.CONSTRAINTS.TO.FIT.KB </pre>
<pre> Own slot: ENGLISH.EXPLANATION from INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD 158 Values: "Possible Rationale: IMPERFECT.DOMAIN. The evidence is : (((The restriction (ONE OF REJECT ACCEPT ACCEPT-WITH-MODIFICATION) on field DECISION of object SUBMISSIONS (ancestor of SUBMISSION.004) is too strict. Replace with (ONE OF INVITED REJECT ACCEPT ACCEPT-WITH-MODIFICATION) to accomodate assertion #[Wff A P:DECISION OF P:SUBMISSION.004 IS P:INVITED])))" </pre>
<pre> Own slot: EXPLANATION.BASIS from INCORRECT.STATIC.KNOWLEDGE.BASE Values: S:IMPERFECT.DOMAIN </pre>
<pre> Own slot: FIX from INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD 158 Values: Type: VALUECLASS-REPLACE Inconsistent-world: [AFTER]<INVITE.PAPERS.FOR.PRESENTATION>SUBMISSIONS-INVITED<001> Assertion: #[Wff A P:DECISION OF P:SUBMISSION.004 IS P:INVITED] Unit: SUBMISSION.004 Source-unit: SUBMISSIONS Slot: DECISION Violating-value: INVITED Violated-spec: (ONE OF REJECT ACCEPT ACCEPT-WITH-MODIFICATION) New-valueclass-spec: (ONE OF INVITED REJECT ACCEPT ACCEPT-WITH-MODIFICATION) Inconsistency-explainer: Antecedents: ((A DECISION OF SUBMISSION.004 IS INVITED)) Consequent: FALSE Violation: VALUECLASS-VIOLATION (((ONE OF REJECT ACCEPT ACCEPT-WITH-MODIFICATION))) </pre>
<pre> Own slot: INCONSISTENT.STATE.SPEC.IN.FOCUS from INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD 158 Values: ISS 136 </pre>

Figure 46: Selected explanation to generalize constraint by adding a disjunct

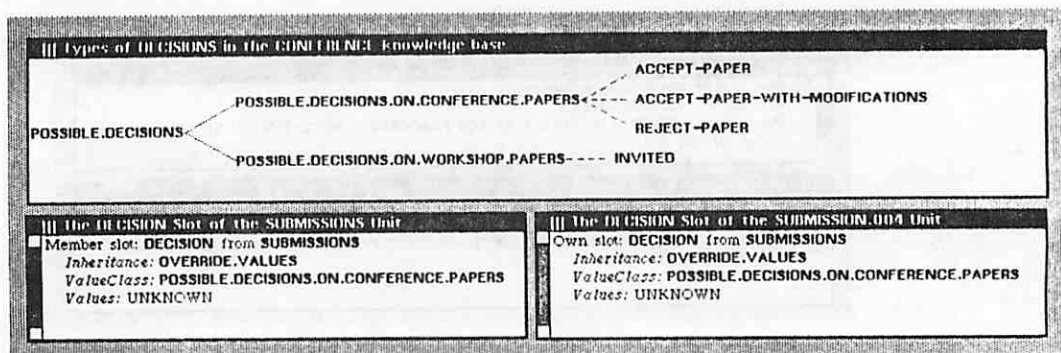


Figure 47: The possible-decisions in the knowledge base and the decision slots of the submission and submission.004 units prior to valueclass modification

on-workshop-papers are shown in Figure 47, along with the current valueclass restrictions on the decisions slot of the submissions and submission.004 objects.

Now, consider replaying the above scenario. This time, when the value invited is asserted, SPANDEX must use a different method to generalize the constraint because of the shift in representation. In general, the method which constructs generalized constraints must take notice of the type of constraint being modified, and the underlying representation of the objects employed in the constraint specification. In the previous example, the generalization required a simple addition of a disjunctive value. In the present case, since the original valueclass specification indicates that the value must be a member of a specified class object possible-decisions-for-conference-papers, the constraint relaxation algorithm determines the least general class object which subsumes the original specification as well as the new value⁵. The resulting class object

⁵It is interesting to consider the case in which a subsuming class object did not already exist. Possibilities to consider would be the dynamic restructuring of the knowledge base, with the create of new object classes "on the fly."

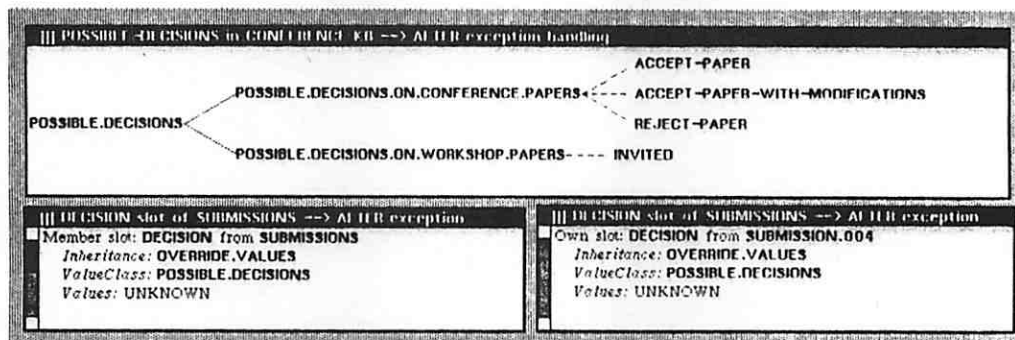


Figure 48: The decision slot of the submission and submission.004 units after valueclass modification

is possible-decisions (refer back to Figure 47). Therefore, when the amendment modify-static-constraints-to-fit-kb is implemented, the original valueclass constraint is replaced with this more general specification. In other words, the decision of a paper now can be any object which is a member of the class possible-decisions, which includes invited. The modification of the constraint is shown in Figure 48, and Figure 49 shows the chosen explanation record, with details about the selected fix.

Creating objects and modifying the taxonomy

In this section, we show an alternative resolution of the same exception which focuses on modifying the object which caused the violation. This approach differs from the previously described methods in that it involves extending the domain model in order to satisfy the violated constraint. For this scenario, however, we assume a different initial knowledge base in order to demonstrate the acquisition of new knowledge. Figure 50 shows how possible decisions on conference papers are once again represented as abstract

```

||| (Output) Current explanation in domain CONFERENCE
Unit: INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD160 in knowledge base SPANDEX
Created by BROVERMAN on 6-21-90 17:27:22
Modified by BROVERMAN on 6-21-90 17:27:22
Member Of: INCORRECT.STATIC.KNOWLEDGE.BASE

Own slot: AMENDMENTS from INCORRECT.STATIC.KNOWLEDGE.BASE
Values: MODIFY.STATIC.CONSTRAINTS.TO.FIT.KB

Own slot: ENGLISH.EXPLANATION from INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD160
Values: "Possible Rationale: IMPERFECT.DOMAIN. The evidence is :
(((The restriction #[Unit: P::POSSIBLE.DECISIONS.ON.CONFERENCE.PAPERS P::CONFERENCE]
on field DECISION of object SUBMISSIONS (ancestor of SUBMISSION.004) is too strict.
Replace with #[Unit: P::POSSIBLE.DECISIONS P::CONFERENCE]
to accomodate assertion #[Wff A P::DECISION OF P::SUBMISSION.004 IS P::INVITED ])))"

Own slot: EXPLANATION.BASIS from INCORRECT.STATIC.KNOWLEDGE.BASE
Values: S:IMPERFECT.DOMAIN

Own slot: FIX from INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD160
Values: Type: VALUECLASS-REPLACE
Inconsistent-world: [AFTER]<INVITE.PAPERS.FOR.PRESENTATION>SUBMISSIONS-INVITED<001>
Assertion: #[Wff A P::DECISION OF P::SUBMISSION.004 IS P::INVITED ]
Unit: SUBMISSION.004
Source-unit: SUBMISSIONS
Slot: DECISION
Violating-value: INVITED
Violated-spec: POSSIBLE.DECISIONS.ON.CONFERENCE.PAPERS
New-valueclass-spec: POSSIBLE.DECISIONS
Inconsistency-explainer: Antecedents: ((A DECISION OF SUBMISSION.004 IS INVITED))
Consequent: FALSE
Violation: VALUECLASS-VIOLATION
((POSSIBLE.DECISIONS.ON.CONFERENCE.PAPERS In CONFERENCE))

Own slot: INCONSISTENT.STATE.SPEC.IN.FOCUS from INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD160
Values: ISS138

```

Figure 49: Selected explanation to generalize constraint by replacing with a more general class object

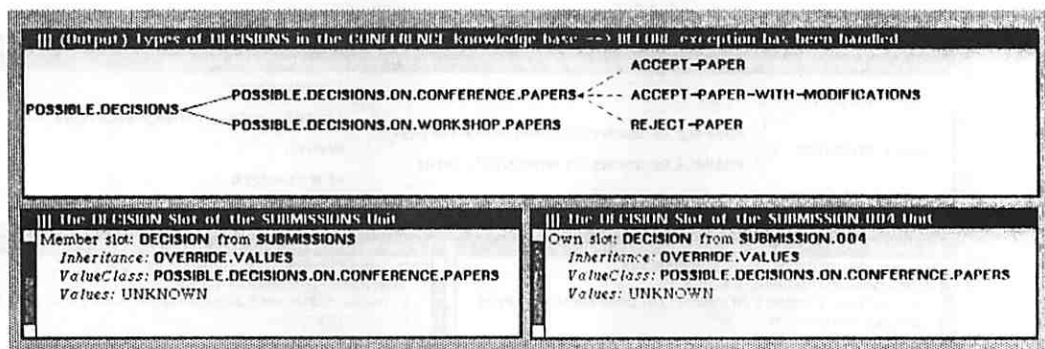


Figure 50: The decision slot of the submission unit and possible-decisions prior to valueclass modification

objects, but the only valid initial values are accept-paper, accept-paper-with-modifications, and reject-paper. Note that the possible decisions for a conference paper do not initially include the value invited. Again, an exception occurs when the planner tries to assert (The decision of submission.004 is invited) during plan expansion, since invited is not a valid decision.

The strategy incomplete-knowledge-base is invoked. The sole inconsistent-state-spec for this strategy, iss133, is applicable since inconsistent worlds have been detected. The strategy then looks for ways, if any, that information can be added to the assumed incomplete domain specification in order to satisfy the constraints. One of the ways in which this can be done is to see if the violating value can be coerced to become a member of the violated class specification. In this case, the value invited is not yet known to be an object in the Conference knowledge base, so SPANDEX is able to create a new object by that name and insert a taxonomic link establishing it to be a new member of the possible-decisions-on-conference-papers class object. The resulting state of the knowledge base after this change has been implemented by the add-

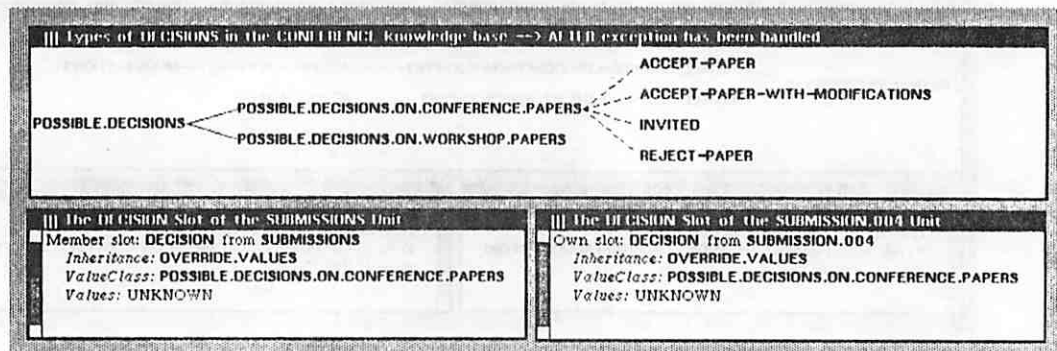


Figure 51: The decision slot of the submission unit and possible-decisions after knowledge base has been extended to satisfy violated constraint

to-kb-to-satisfy-constraints amendment is shown in Figure 51. The selected explanation record which contains details of the proposed and implemented fix is shown in Figure 52.

Figure 53 shows the contents of the SPANDEX execution record after one of the possible explanations has been selected and implemented. Figures 54 and 55 illustrate the interaction with the user in selecting among complete explanations and proposed amendments to complete the exception handling cycle.

As a result of the successful handling of this exception, the plan can be resumed, obviating any need for replanning⁶. In addition, additional knowledge

⁶Due to a bug in the implementation of the KEEworlds mechanism, the current implementation of this scenario is not actually able to continue past the point where SPANDEX determines and implements the necessary modifications. Changes to valueclass constraints in KEE are made to *background* (as opposed to *alternative worlds*) knowledge, and currently the global change is not propagated correctly into the worlds associated with each of the plan nodes. Therefore, the traces of the original valueclass violation cannot currently be fully retracted from the worlds mechanism in order to allow the resumption of planning and execution. However, the successful run that would result in the absence of this imple-

```

[[ (Output) Current explanation in domain CONFERENCE
Unit: INCOMPLETE.KNOWLEDGE.BASE.RECORD 148 in knowledge base SPANDEX
Created by BROVERMAN on 6-21-90 17:27:22
Modified by BROVERMAN on 6-21-90 17:47:01
Member Of: INCOMPLETE.KNOWLEDGE.BASE

Own slot: AMENDMENTS from INCOMPLETE.KNOWLEDGE.BASE
Values: ADD.TO.KB.TO.SATISFY.CONSTRAINTS

Own slot: ENGLISH.EXPLANATION from INCOMPLETE.KNOWLEDGE.BASE.RECORD 148
Values: "Possible rationale: IMPERFECT.DOMAIN. The evidence is:
(((The constraint #[Unit: P::POSSIBLE.DECISIONS.ON.CONFERENCE.PAPERS P::CONFERENCE],
on slot DECISION of object SUBMISSION.004 will not permit the value INVITED
being asserted by #[With A P::DECISION OF P::SUBMISSION.004 IS P::INVITED ],
Coerce INVITED to be of type:
#[Unit: P::POSSIBLE.DECISIONS.ON.CONFERENCE.PAPERS P::CONFERENCE] to satisfy the constraint.)))"

Own slot: EXPLANATION.BASIS from INCOMPLETE.KNOWLEDGE.BASE
Values: S:IMPERFECT.DOMAIN

Own slot: FIX from INCOMPLETE.KNOWLEDGE.BASE.RECORD 148
Values: Type: VALUECLASS-COERCE
Inconsistent-world: [AFTER]<INVITE.PAPERS.FOR.PRESENTATION>SUBMISSIONS-INVITED<001>
Assertion: #[With A P::DECISION OF P::SUBMISSION.004 IS P::INVITED ]
Unit: SUBMISSION.004
Slot: DECISION
Violating-value: INVITED
Violated-spec: POSSIBLE.DECISIONS.ON.CONFERENCE.PAPERS
Coerce-violating-value-to-class: POSSIBLE.DECISIONS.ON.CONFERENCE.PAPERS
Inconsistency-explainer: Antecedents: ((A DECISION OF SUBMISSION.004 IS INVITED))
Consequent: FALSE
Violation: VALUECLASS-VIOLATION
((POSSIBLE.DECISIONS.ON.CONFERENCE.PAPERS in CONFERENCE))

Own slot: INCONSISTENT.STATE.SPEC.IN.FOCUS from INCOMPLETE.KNOWLEDGE.BASE.RECORD 148
Values: ISS134

```

Figure 52: Selected explanation to modify the knowledge base in order to satisfy a violated constraint


```

||| The SPANDEX execution record after resolving inconsistent-world-during-planning exception
Unit: *EXECUTION.RECORD* in knowledge base SPANDEX
Created by BROVERMAN on 3-6-89 13:07:56
Modified by BROVERMAN on 6-20-90 23:11:01
Member Of: ENTITIES in GENERICSUNITS

Maintains information about the current execution cycle.

Own slot: ASSERTION from *EXECUTION.RECORD*
Values: #[Wff A P::DECISION OF P::SUBMISSION.004 IS P::INVITED ]

Own slot: COMPLETE.EXPLANATIONS from *EXECUTION.RECORD*
Values: INCOMPLETE.KNOWLEDGE.BASE.RECORD307, INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD309

Own slot: EXCEPTION.TYPE from *EXECUTION.RECORD*
Values: S::INCONSISTENT-WORLD-DURING-PLANNING

Own slot: INCONSISTENT.WORLD from *EXECUTION.RECORD*
Values: [AFTER]'INVITE.PAPERS.FOR.PRESENTATION>SUBMISSIONS-INVITED<001>

Own slot: OVERALL.EXECUTION.RESULT from *EXECUTION.RECORD*
COMMENT: "Indicates success or failure of execution and exception handler."
Values: P::RESOLVED-EXCEPTION

Own slot: POSSIBLE.EXPLANATIONS from *EXECUTION.RECORD*
COMMENT: "Instantiations of rationale.evidence.records for this exception, having processed inconsistencies and indicators."
Values: INCOMPLETE.KNOWLEDGE.BASE.RECORD308, INCOMPLETE.KNOWLEDGE.BASE.RECORD307,
INCORRECT.DYNAMIC.KNOWLEDGE.BASE.RECORD308, INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD309

Own slot: SELECTED.AMENDMENT from *EXECUTION.RECORD*
COMMENT: "The amendment (instance of amendment) which was chosen interactively by the user, or by the system."
Values: S::MODIFY.STATIC.CONSTRAINTS.TO.FIT.KB

Own slot: SELECTED.EXPLANATION from *EXECUTION.RECORD*
COMMENT: "The explanation (instance of a rationale.establishment.strategy) that was chosen interactively by the user, or by the system."
Values: INCORRECT.STATIC.KNOWLEDGE.BASE.RECORD309

```

Figure 53: Contents of the SPANDEX execution.record after handling an exception during planning

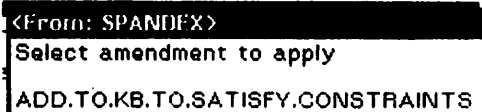
```

<From: SPANDEX>
Select explanation to implement

INCOMPLETE.KNOWLEDGE.BASE.RECORD43
INCORRECT.DYNAMIC.KNOWLEDGE.BASE.RECORD45

```

Figure 54: User selects from possible SPANDEX explanations to resolve the exception

A screenshot of a SPANDEX interface. It shows a black header bar with the text "<From: SPANDEX>". Below the header is a white rectangular area with a black border containing the text "Select amendment to apply" and "ADD.TO.KB.TO.SATISFY.CONSTRAINTS".

```
<From: SPANDEX>
Select amendment to apply
ADD.TO.KB.TO.SATISFY.CONSTRAINTS
```

Figure 55: User selects from possible SPANDEX amendments to resolve the exception

about how submissions can be admitted into a conference (e.g., the knowledge that submissions may be invited, in addition to being accepted and rejected via the reviewing process) has been acquired via exception handling. Since the domain knowledge base has been extended, an execution of the same scenario in the future will be executed without incident; this exception can no longer occur in the context of the refined domain description.

7.1.2 Scenario 2

This scenario illustrates how a new activity can be acquired when an agent's response indicates that an unknown activity has been performed, and then shows how the shortcut rationale can be substantiated by showing that the accomplished action subsumes a portion of the original plan⁷.

Suppose a conference organizer is setting up an academic conference. This task involves many interrelated steps, such as getting approval to hold the conference, soliciting and refereeing submissions, and arranging logistics such as housing and meals. As part of planning for the conference, a call for papers

mentation bug can be simulated by reinitializing and replaying the scenario after SPANDEX background modifications have been made. The replay of the scenario will run through successfully once the *worlds* have been reinitialized.

⁷This is the identical scenario described in Section 1.5 but includes some more detail.

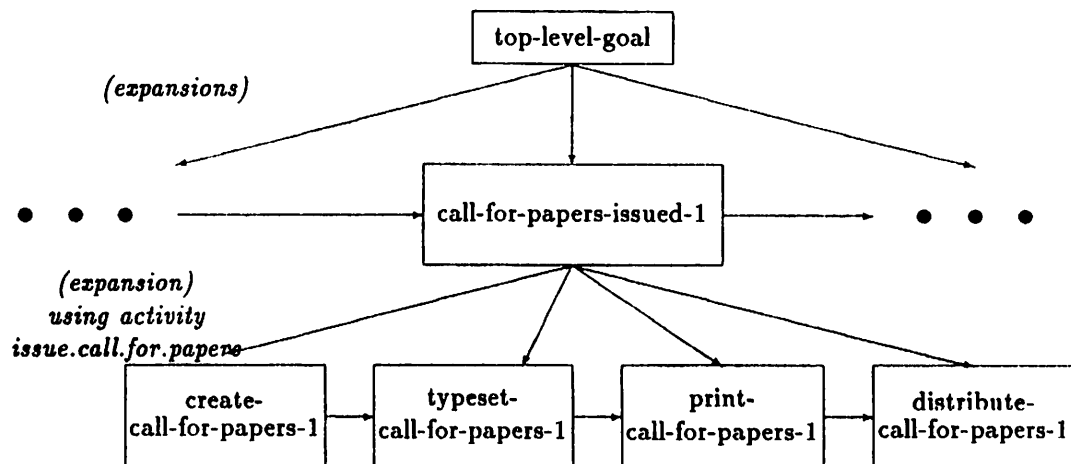


Figure 56: Partial plan outline for issuing a call for papers

is usually issued by the program chair. The way this is commonly done is to prepare the actual content of the call for papers, have it typeset, then printed, and finally, distributed. Since the knowledge base contains a single activity *issue.call.for.papers* which embodies this task knowledge, the planner selects that activity to achieve the goal *call-for-papers-issued*. Figure 56 depicts part of the plan outline after the activity selection.

The program chair *Larry.Lefkowitz.PhD* next creates the content of the actual *call.for.papers* as anticipated. Figure 57 shows a portion of the actual plan network (the leaf nodes of the plan outline) that the planner has constructed. He is then asked by the system to request that the typesetter execute the *typeset.call.for.papers.1* action (Figure 58). However, the organization which is sponsoring the conference has recently begun to offer a contract service to handle the various stages of producing and distributing a *call.for.papers*, and the program chair would like to make use of this new service. Since this is not

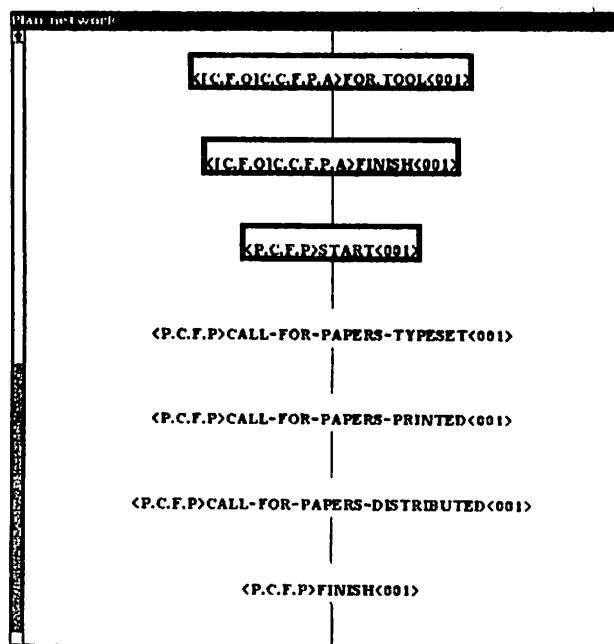


Figure 57: The POLYMER plan network for issuing a call for papers
 one of the known possibilities presented to the organizer, Larry.Lefkowitz.PhD responds that he would like to perform another activity instead.

The system recognizes that an exception has occurred (agent-response-inconsistent-with-executable-expectation). An interactive session is established in order to resolve the exception. First, the system issues a request to

```

JOES.TYPESETTERS: Please perform TYPESET.DOCUMENT
To achieve (A STATUS OF CALL.FOR.PAPERS<001> IS TYPESET)
OK
I would like to do something different.
  
```

Figure 58: Asking the user to perform a task

```

Indicate the activity performed instead
User assertion
SEND.PROCEEDINGS.TO.SPONSOR
SETTLE.EXPENSES
ISSUE.FINAL.PROGRAM
ISSUE.ADVANCE.PROGRAM
DEVISE.SYMPOSIUM.SCHEDULE
REVIEWER.MAKE.RECOMMENDATION
REVIEW.PAPER
GET.REVIEWERS
GET.PAPER.REVIEWED
RECEIVE.SUBMISSIONS
INVITE.A.PAPER.AD
DISTRIBUTE.DOCUMENT
PRINT.DOCUMENT
TYPESET.DOCUMENT
NOTIFY.AUTHOR
PROGRAM.COMMITTEE.DECIDES.ON.PAPER
INVITE.A.PAPER
PROCESS.CALL.FOR.PAPERS
ISSUE.CALL.FOR.PAPERS
JUDGE.PAPERS.FOR.PRESENTATION
INVITE.PAPERS.FOR.PRESENTATION
REPORT.ON.CONFERENCE
DESIGNATE.UNIVERSITY.SYMPOSIUM.SITE
DESIGNATE.SYMPOSIUM.SITE
RECEIVE.APPROVED.PROPOSAL
SEND.PROPOSAL
DESIGNATE.SYMPOSIUM.DATES
CHECK.FOR.DATE.OVERLAPS
COMPLETE.TMRF
SUBMIT.CALENDAR.ENTRY.FORM
GET.APPROVAL.FOR.SYMPOSIUM
POSTPROCESS.A.SYMPOSIUM
HOLD.CONFERENCE
CHOOSE.PROGRAM.COMMITTEE
PLAN.A.SYMPOSIUM
STAGE.A.SYMPOSIUM
** ==>Other activity -- (create) **

```

Figure 59: The user chooses from among alternative activities

Larry.Lefkowitz.PhD to indicate the performed activity from a menu which includes all known activities, as well as an "Other Activity" choice (Figure 59).

He selects "Other Activity", which the system recognizes as a cue to acquire a new activity description from the agent. Larry.Lefkowitz.PhD uses the DACRON graphical interface [43] to specify the new activity contract.out.call.for.papers, as shown in Figure 60. This interface uses an iconic specification language to allow the user to express that the goal of this new activity is (a ?call-for-papers of ?technical-meeting is issued), with various effects (the one shown is that the status of the call-for-papers is distributed after this activity takes place). The new description is assimilated into the knowledge base for future use.

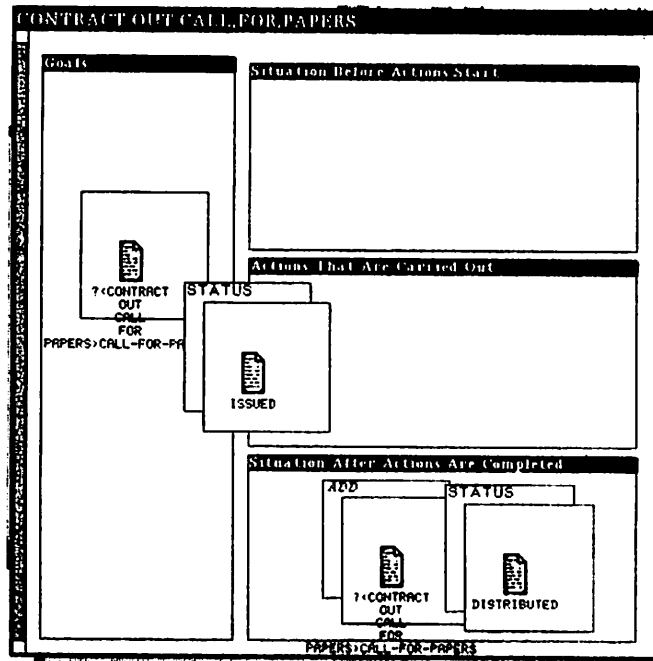


Figure 60: The newly acquired activity CONTRACT-OUT-CALL-FOR-PAPERS

III (Output) The "EXECUTION.RECORD" Unit in SPANDEX Knowledge Base	
Unit: "EXECUTION.RECORD" in knowledge base SPANDEX	
Created by BROVERMAN on 3-6-89 13:07:56	
Modified by BROVERMAN on 6-27-90 23:16:42	
Member Of: ENTITIES in GENERICUNITS	
Maintains information about the current execution cycle.	
Own slot:	ACCOMPLISHED.ACTION from "EXECUTION.RECORD"
Values:	P::ACM.CONTRACT.OUT.CALL.FOR.PAPERS
Own slot:	ACCOMPLISHED.STATE from "EXECUTION.RECORD"
Values:	#[Wff A P::STATUS OF P::CALL.FOR.PAPERS<002> IS P::ISSUED]
Own slot:	COMPLETE.EXPLANATIONS from "EXECUTION.RECORD"
Values:	SUBSUMES.CURRENT.STEP.RECORD341
Own slot:	EXCEPTION.TYPE from "EXECUTION.RECORD"
Values:	S::AGENT-RESPONSE-INCONSISTENT-WITH-AGENT-EXECUTABLE-EXPECTATION
Own slot:	MATCH.RESULT from "EXECUTION.RECORD"
Values:	#(S::MATCH-RECORD :WORLD-STATES 6 :ACTIVITIES 4 :ACTIVITY-GOALS 5 :ACHIEVE D-STATE-AND-PERFORMED-ACTIVITY-GOAL 1)
Own slot:	OVERALL.EXECUTION.RESULT from "EXECUTION.RECORD"
Values:	P::RESOLVED-EXCEPTION
Own slot:	SELECTED.AMENDMENT from "EXECUTION.RECORD"
Values:	S::REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT
Own slot:	SELECTED.EXPLANATION from "EXECUTION.RECORD"
Values:	SUBSUMES.CURRENT.STEP.RECORD341
Own slot:	SUGGESTED.ACTION from "EXECUTION.RECORD"
Values:	P::TYPESET.DOCUMENT
Own slot:	SUGGESTED.NODE from "EXECUTION.RECORD"
Values:	<TYPESET.DOCUMENT>FOR.AGENT<001> in CONFERENCE
Own slot:	SUGGESTED.STATE from "EXECUTION.RECORD"
Values:	#[Wff A P::STATUS OF P::CALL.FOR.PAPERS<002> IS P::TYPESET]

Figure 61: The SPANDEX execution record

The task for the exception analyst is to now determine how this newly specified activity relates to the current dynamic model of the task. The SPANDEX execution.record summarizes the exception (see Figure 61).

Two complete explanations result from the application of rationale establishment strategies relevant to this exception type. Both explanations are valid, but the one chosen interactively through negotiation with the user is shown in Figure 62. The chosen explanation record states that since the activity contract.out.call.for.papers has been verified to achieve the same goal as

```

[[ (Output) The SUBSUMES.CURRENT.STEP.RECORD341 Unit in SPANDEX Knowledge Base
Unit: SUBSUMES.CURRENT.STEP.RECORD341 in knowledge base SPANDEX
Created by BROVERMAN on 6-27-90 23:15:22
Modified by BROVERMAN on 6-27-90 23:15:35
Member Of: SUBSUMES.CURRENT.STEP

Own slot: AMENDMENTS from SUBSUMES.CURRENT.STEP
Values: REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT

Own slot: COMPARISON.NODE from SUBSUMES.CURRENT.STEP.RECORD341
Values: <ISSUE.CALL.FOR.PAPERS>CALL-FOR-PAPERS-PROCESSED-AND-ISSUED<001> (0: CONFERENCE

Own slot: ENGLISH.EXPLANATION from SUBSUMES.CURRENT.STEP.RECORD341
Values: "Possible Rationale: SHORTCUT. The evidence is :
(((The performed action ACM.CONTRACT.OUT.CALL.FOR.PAPERS
achieved a goal which subsumes the expected action TYPESET.DOCUMENT.)))"

Own slot: EXPLANATION.BASIS from SUBSUMES.CURRENT.STEP
Values: S:SHORTCUT

Own slot: INCONSISTENT.STATE.SPEC.IN.FOCUS from SUBSUMES.CURRENT.STEP.RECORD341
Values: IS322

Own slot: MATCH.RESULT from SUBSUMES.CURRENT.STEP.RECORD341
Values: #S(MATCH-RECORD:WORLD-STATES 1:ACTIVITIES 3:ACTIVITY-GOALS 1:ACHIEVED-STA
TE-AND-PERFORMED-ACTIVITY-GOAL 1)

```

Figure 62: Chosen explanation

the activity *issue.call.for.papers*⁸, this unexpected action may be a substitution for the abstract goal node that derived the *issue.call.for.papers* expansion (refer back to Figure 56). In addition, this substitution subsumes the expected action *typeset.call.for.papers.1*.

An amendment is then proposed for the explanation which specifies the changes that must be made to the current representation of the plan in order to restore consistency. The implementation of this explanation involves replacing the wedge of the plan network subsumed by the more abstract parent node (*call-for-papers-issued-1*) with the unexpected action (*contract.out.call.for.papers.1*). As a side effect, the unexecuted nodes in the expansion of *call-for-papers-issued-1* are deactivated from the planner's predictions.

⁸This comparison result is stored in the *match.result* field of the explanation (see Figure 62) and was calculated by the exception classifier.

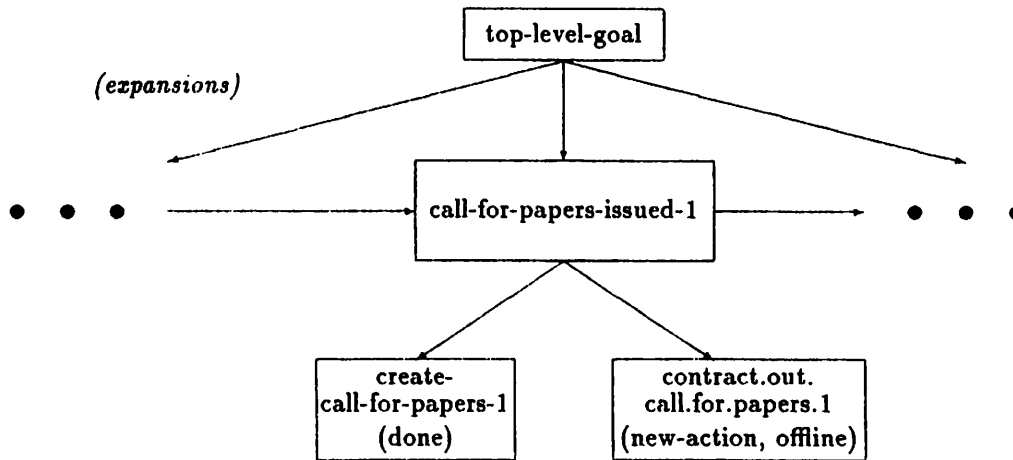


Figure 63: Partial plan outline after exception is resolved

A portion of the plan outline which now represents the hierarchical decomposition of the task is depicted in Figure 63, and a portion of the actual plan network which results from the implementation of the amendment just described is shown in Figure 64.

An abbreviated version⁹ of the planner messages generated during this scenario follows:

```

1: Instantiating SELECT-PAPERS-FROM-SUBMISSIONS-SCENARIO-2 scenario.
2: -----
3: Scenario SELECT-PAPERS-FROM-SUBMISSIONS-SCENARIO-2<001> selected.
4: -----
5: PROCESSING GOAL NODE:
   <*SELECT-PAPERS-FROM-SUBMISSIONS-SCENARIO-2*>TOP-LEVEL-GOAL<001>
6: -----
7: Selected activity: JUDGE.PAPERS.FOR.PRESENTATION
8: for goal (A PAPERS.CHOSEN.FOR.PRESENTATION OF
   PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS DONE)
9: -----
10: PROCESSING GOAL NODE:
    <JUDGE.PAPERS.FOR.PRESENTATION>SUBMISSIONS-SOLICITED<001>
11: -----

```

⁹We have removed various other messages from the planner that are not of particular interest here; e.g. the information about bindings and the processing of structural nodes, etc.

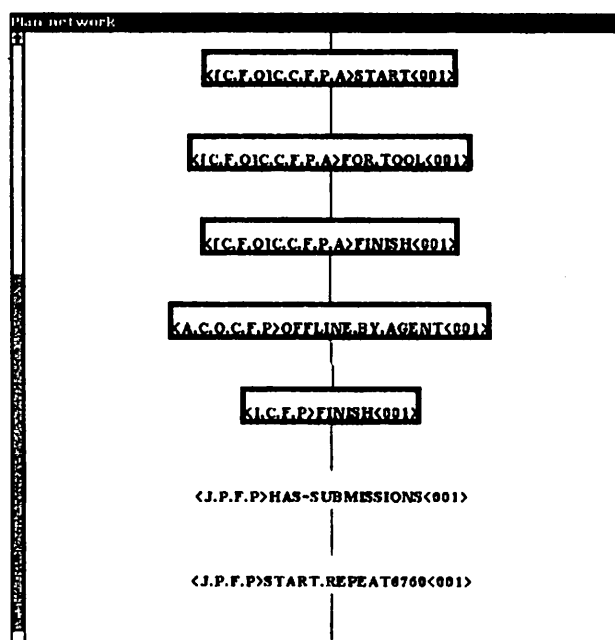


Figure 64: Partial view of POLYMER plan network after exception is resolved

```

9: Selected activity: ISSUE.CALL.FOR.PAPERS
10: for goal (AN ISSUED.CALL.FOR.PAPERS OF
    PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS ?<JUDGE.PAPERS.FOR.PRESENTATION>CALL)
12: -----
11: PROCESSING GOAL NODE: <ISSUE.CALL.FOR.PAPERS>CALL-FOR-PAPERS-CREATED<001>
12: -----
13: Selected activity: [CREATE.FORM.OBJECTS]CREATE.CALL.FOR.PAPERS.ACTIVITY
14: for goal (A STATUS OF ?<ISSUE.CALL.FOR.PAPERS>CALL-FOR-PAPERS IS CREATED)
16: -----
55: PROCESSING TOOL.EXECUTABLE NODE:
    <[CREATE.FORM.OBJECTS]CREATE.CALL.FOR.PAPERS.ACTIVITY>FOR.TOOL<001>
56: -----
57: About to execute [CREATE.FORM.OBJECTS]CREATE.CALL.FOR.PAPERS
59: -----
68: PROCESSING GOAL NODE:
    <ISSUE.CALL.FOR.PAPERS>CALL-FOR-PAPERS-PROCESSED-AND-ISSUED<001>
69: -----
70: Selected activity: PROCESS.CALL.FOR.PAPERS
71: for goal (A STATUS OF CALL.FOR.PAPERS<004> IS ISSUED)
73: -----
82: PROCESSING GOAL NODE: <PROCESS.CALL.FOR.PAPERS>CALL-FOR-PAPERS-TYPESET<001>
83: -----
84: Selected activity: TYPESET.DOCUMENT
85: for goal (A STATUS OF CALL.FOR.PAPERS<004> IS TYPESET)
87: -----
96: PROCESSING AGENT.EXECUTABLE NODE: <TYPESET.DOCUMENT>FOR.AGENT<001>
97: -----

--> *** HERE IS WHERE THE EXCEPTION OCCURS!!!
    *** Agent performs ACM.CONTRACT.OUT.CALL.FOR.PAPERS instead.
    *** HANDLING FOLLOWS.***

98: SPANDEX: user chose the REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT
    amendment to implement the chosen explanation SUBSUMES.CURRENT.STEP.RECORD421.
    using rationale SHORTCUT.
99: -----
100: SPANDEX: excising unstarted portions of wedge subsumed by expanded goal node
    <ISSUE.CALL.FOR.PAPERS>CALL-FOR-PAPERS-PROCESSED-AND-ISSUED<001>.
101: -----
102: SPANDEX: removing node <PROCESS.CALL.FOR.PAPERS>CALL-FOR-PAPERS-TYPESET<001>
    from the planning history.
103: -----
104: SPANDEX: removing node <PROCESS.CALL.FOR.PAPERS>CALL-FOR-PAPERS-PRINTED<001>
    from the current network.
105: -----
106: SPANDEX: removing node
    <PROCESS.CALL.FOR.PAPERS>CALL-FOR-PAPERS-DISTRIBUTED<001>
    from the current network.

```

```

107: -----
108: SPANDEX: removing node <TYPESET.DOCUMENT>FOR.AGENT<001> from the
      current network.
109: -----
110: SPANDEX: removing node <PROCESS.CALL.FOR.PAPERS>START<001> from
      the current network.
111: -----
112: SPANDEX: removing node <PROCESS.CALL.FOR.PAPERS>FINISH<001> from
      the current network.
113: -----
114: SPANDEX: removing node <TYPESET.DOCUMENT>START<001> from
      the current network.
115: -----
116: SPANDEX: removing node <TYPESET.DOCUMENT>FINISH<001> from the current network.
117: -----
118: Reexpanding goal (A STATUS OF CALL.FOR.PAPERS<004> IS ISSUED)
      with: ACM.CONTRACT.OUT.CALL.FOR.PAPERS
121: -----"
122: SPANDEX:... and including offline action
      <ACM.CONTRACT.OUT.CALL.FOR.PAPERS>OFFLINE.BY.AGENT<001>.
123: -----"
130: NEXT NODE: <JUDGE.PAPERS.FOR.PRESENTATION>HAS-SUBMISSIONS<001>

```

A variation. An exception that is resolved by the replacement of a plan wedge with an offline action can greatly reduce the complexity of the plan, by eliminating many detailed steps. Consider for example a case in which the conference chair is using the planner to generate a detailed plan of action for staging a conference, but indicates that he has done all of the preliminary organization "offline" beforehand. The plan network shown in Figure 65 becomes much simpler (Figure 67) when the chosen explanation (Figure 68) is implemented upon the detection of the exception (Figure 66).

7.1.3 Scenario 3

We have shown how the successful handling of an agent-response-inconsistent-with-executable-expectation exception can result in the acquisition

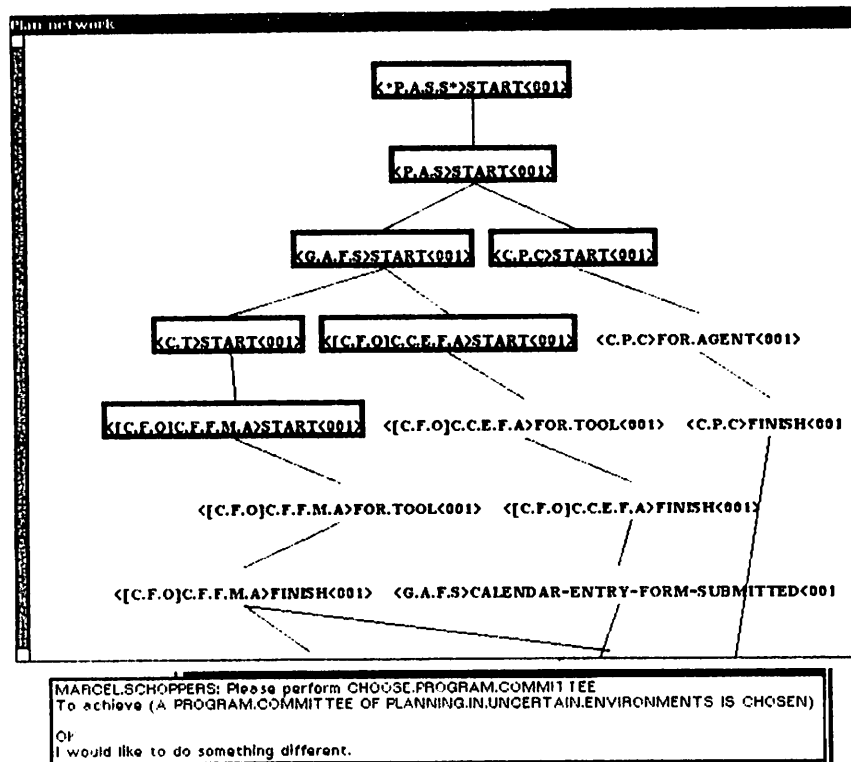


Figure 65: Plan network before exception, showing interaction with user

```

-----
CALLING EXCEPTION HANDLER:
MISMATCH in EXPECTATIONS:

agent: #[Unit: P::MARCEL.SCHOPPERS P::CONFERENCE]
plan node: CHOOSE.PROGRAM.COMMITTEE.FOR.AGENT<001>
requested goal: (A PROGRAM.COMMITTEE OF PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS CHOSEN)
using activity: CHOOSE.PROGRAM.COMMITTEE
accomplished goal: (A COMPLETION.STATUS OF PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS PLANNED)
performed activity: PLAN.A.SYMPOSIUM
-----
Press the space bar to remove this message:
  
```

Figure 66: Calling the exception handler

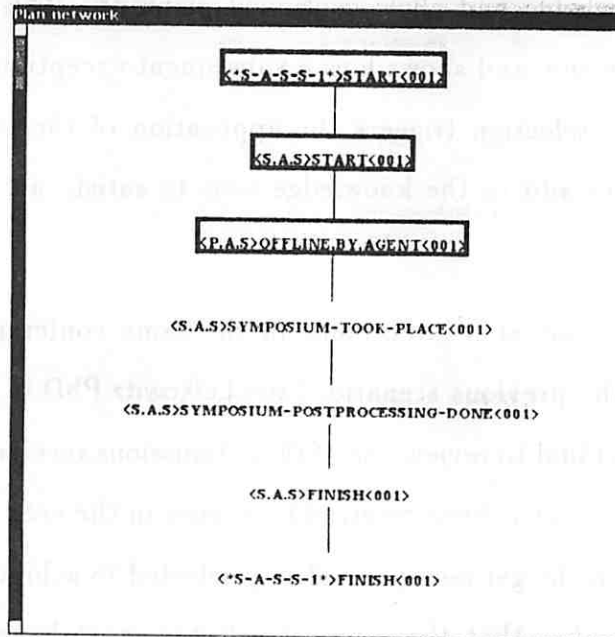


Figure 67: The plan network after wedge subsumed by symposium-is-planned is excised

III (Output) The SUBSUMES.CURRENT.STEP.RECORD100 Unit in SPANDEX Knowledge Base	
Unit:	SUBSUMES.CURRENT.STEP.RECORD100 in Knowledge base SPANDEX
Created by:	BROVERMAN on 6-27-90 19:21:22
Modified by:	BROVERMAN on 6-27-90 19:22:12
Member Of:	SUBSUMES.CURRENT.STEP
Own slot:	AMENDMENTS from SUBSUMES.CURRENT.STEP
Values:	REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT
Own slot:	COMPARISON.NODE from SUBSUMES.CURRENT.STEP.RECORD100
Values:	<STAGE.A.SYMPOSIUM>SYMPOSIUM-IS-PLANNED<001> in CONFERENCE
Own slot:	ENGLISH.EXPLANATION from SUBSUMES.CURRENT.STEP.RECORD100
Values:	"Possible Rationale: SHORTCUT. The evidence is : (((The performed action PLAN.A.SYMPOSIUM achieved a goal which subsumes the expected action CHOOSE.PROGRAM.COMMITTEE.)))"
Own slot:	EXPLANATION.BASIS from SUBSUMES.CURRENT.STEP
Values:	S:SHORTCUT
Own slot:	INCONSISTENT.STATE.SPEC.IN.FOCUS from SUBSUMES.CURRENT.STEP.RECORD100
Values:	ISS22
Own slot:	MATCH.RESULT from SUBSUMES.CURRENT.STEP.RECORD100
Values:	#S(S:MATCH-RECORD:WORLD-STATES 1:ACTIVITIES 4:ACTIVITY-GOALS 1:ACHIEVED-STATE-AND-PERFORMED-ACTIVITY-GOAL 1)

Figure 68: The selected explanation record for exception in Scenario 2

of new knowledge and allow continued planning. This scenario is based on the previous one, and shows how a subsequent exception that is detected during resource selection triggers the application of the constraint partitioning techniques to add to the knowledge base to satisfy an unsatisfied constraint set.

Suppose that at a later point in the same conference planning task described in the previous scenario, Larry.Lefkowitz.PhD is asked to select a particular individual to review one of the submissions received. The paper in question (paper.001) has been received for review in the area of machine.learning. A constraint on the get-reviewers activity selected to achieve the reviewer-selected subgoal specifies that the reviewer selected must have an area of expertise which matches the area of the paper to be reviewed (see the agent-executable node generated from this activity in Figure 69).

Figure 70 illustrates a portion of the current knowledge base, showing the paper being reviewed as well as the available reviewers and their areas.of.expertise. An examination of the knowledge base, along with an analysis of the posted constraints, imply that only three of the shown reviewers are eligible to be selected.

When the program chair is instructed to perform the activity select.reviewer, those reviewers which are known to satisfy the specified constraints are presented for selection (Figure 71). Larry.Lefkowitz.PhD, however, would like to designate Paul.Utgoff as the reviewer of this paper, and although this is not one of the listed choices, Larry.Lefkowitz.PhD chooses Override suggestions from the menu to specify the alternate selection. The system makes note of the new

```

||| (Output) The <GET REVIEWERS FOR AGENT<001> Unit in CONFERENCE Knowledge Base
Unit: <GET REVIEWERS>FOR.AGENT<001> in knowledge base CONFERENCE
Created by BROVERMAN on 7-3-90 18:46:58
Modified by BROVERMAN on 7-3-90 18:47:00
Member Of: <GET REVIEWERS>FOR.AGENT, INSTANCES in POLYMER

Own slot: AGENT from <GET REVIEWERS>FOR.AGENT<001>
Values: JAMES.HENDLER

Own slot: CONDITIONS from <GET REVIEWERS>FOR.AGENT<001>
Values: #[Wff P::SUBMISSION.001 IS IN CLASS P::SUBMISSIONS ],
#[Wff P::JAMES.HENDLER IS IN CLASS P::PROGRAM.CHAIR ]

Own slot: FROM.ACTIVITY from <GET REVIEWERS>FOR.AGENT
Values: GET.REVIEWERS

Own slot: FROM.NODE from <GET REVIEWERS>FOR.AGENT<001>
Values: <GET.PAPER.REVIEWED>REVIEWERS-SELECTED<001>

Own slot: GOAL from <GET REVIEWERS>FOR.AGENT<001>
Values: #[Wff A P::REVIEWERS OF P::SUBMISSION.001 IS P::?<GET REVIEWERS>REVIEWERS ]

Own slot: IF.BOUND.CONDITIONS from <GET REVIEWERS>FOR.AGENT<001>
Values: #[Wff P::?<GET REVIEWERS>REVIEWERS IS IN CLASS P::REVIEWERS ],
#[Wff A P::PAPER OF P::SUBMISSION.001 IS P::?<GET REVIEWERS>PAPER ],
#[Wff A P::KEYWORDS OF P::?<GET REVIEWERS>PAPER IS P::?<GET REVIEWERS>SUBJECT-AREA ],
#[Wff AN P::AREAS.OF.EXPERTISE OF P::?<GET REVIEWERS>REVIEWERS IS P::?<GET REVIEWERS>SUBJECT-AREA ]

```

Figure 69: Conditions on the selection of a reviewer, specified in executable node

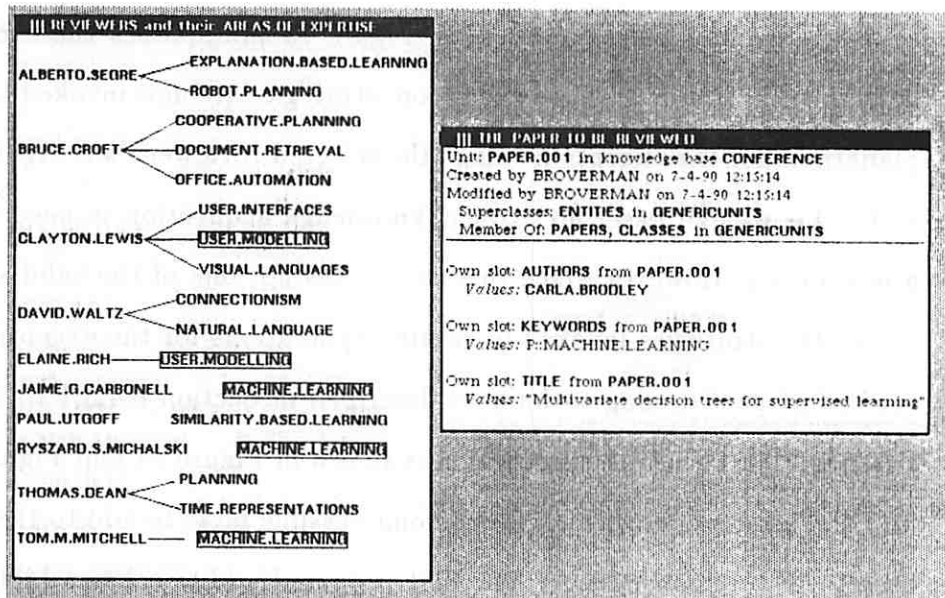
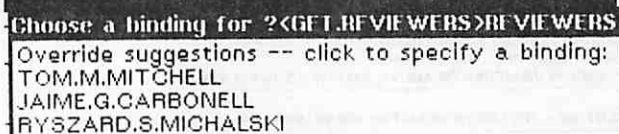


Figure 70: The paper to review and available reviewers

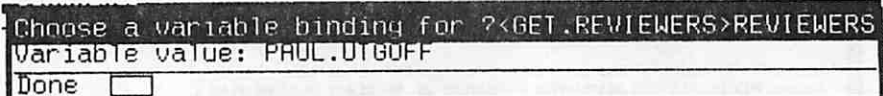


```

Choose a binding for ?<GET.REVIEWERS>REVIEWERS
Override suggestions -- click to specify a binding:
TOM.M.MITCHELL
JAIME.G.CARBONELL
RYSZARD.S.MICHALSKI

```

Figure 71: Choosing among qualified reviewers



```

Choose a variable binding for ?<GET.REVIEWERS>REVIEWERS
Variable value: PAUL.UTGOFF
Done 

```

Figure 72: Specifying a new reviewer not in suggestions

resource selection (Figure 72), and SPANDEX is invoked upon the detection of this resource-choice-inconsistent-with-expectations exception.

The execution.record pictured in Figure 73 summarizes the exception. As with the case of the previous exception, strategies are now invoked to create explanations for the exception. Since the selected reviewer Paul.Utgoff is already in the knowledge base, no explicit knowledge acquisition is needed to define a new entity. However, since this choice was not one of the valid suggestions, one of the strategies invoked to create explanations for the exception uses the constraint partitioning techniques described in Section 6.4.5.1 to restore consistency. The resulting explanation is shown in Figure 74 and Figure 75 shows how the user is asked which fact among possible facts to add to the knowledge base to restore consistency. The first proposed addition has a higher priority, due to the heuristic that prefers to “shoe-horn” the controversial choice into

```

[[ (Output) The 'EXECUTION.RECORD' unit in SPANDEX knowledge base
Unit: 'EXECUTION.RECORD' in knowledge base SPANDEX
Created by BROVERMAN on 3-6-89 13:07:56
Modified by BROVERMAN on 7-3-90 0:16:54
Member Of: ENTITIES in GENERCOUNTS

Maintains information about the current execution cycle.

Own slot: ACCOMPLISHED.ACTION from 'EXECUTION.RECORD'
Values: P:GET.REVIEWERS

Own slot: ACCOMPLISHED.STATE from 'EXECUTION.RECORD'
Values: #[Wff A P:REVIEWERS OF P:SUBMISSION.001 IS P:PAUL.UTODIFF ]

Own slot: BINDING.TYPE from 'EXECUTION.RECORD'
Values: OVERRIDE

Own slot: COMPLETE.EXPLANATIONS from 'EXECUTION.RECORD'
Values: INCOMPLETE.KNOWLEDGE.BASE.RECORDS 1,
INCORRECT.DYNAMIC.KNOWLEDGE.BASE.RECORDS 3

Own slot: EXCEPTION.TYPE from 'EXECUTION.RECORD'
Values: S:RESOURCE-CHOICE-INCONSISTENT-WITH-EXPECTATIONS

Own slot: OVERALL.EXECUTION.RESULT from 'EXECUTION.RECORD'
Values: P:RESOLVED-EXCEPTION

Own slot: RETURNED.BINDINGS from 'EXECUTION.RECORD'
Values: ((#[Var P:KGET.REVIEWERS:REVIEWERS] . #[Wff P:PAUL.UTODIFF ]))

Own slot: SELECTED.AMENDMENT from 'EXECUTION.RECORD'
Values: S:ADD.TO.KB.TO.SATISFY.CONSTRAINTS

Own slot: SELECTED.EXPLANATION from 'EXECUTION.RECORD'
Values: INCOMPLETE.KNOWLEDGE.BASE.RECORDS 1

```

Figure 73: The SPANDEX execution.record

the current plan, i.e. modify the resource, rather than modify another existing object not in focus.

An additional example showing how an inconsistent-resource-choice is resolved was given in detail in the section which described the constraint partitioning algorithm, and begins on page 192.

7.1.4 Scenario 4

This scenario illustrates an agent-response-inconsistent-with-executable-expectation in which the reorder explanation basis is used. The plan steps are rearranged in order to incorporate an action which is performed out of or-

```

||| (Output) Current explanation in domain CONFERENCE
Unit: INCOMPLETE.KNOWLEDGE.BASE.RECORD51 in knowledge base SPANDEX
Created by BROVERMAN on 7-3-90 0:03:16
Modified by BROVERMAN on 7-3-90 0:03:29
Member Of: INCOMPLETE.KNOWLEDGE.BASE

Own slot: AMENDMENTS from INCOMPLETE.KNOWLEDGE.BASE
Values: ADD.TO.KB.TO.SATISFY.CONSTRAINTS

Own slot: ENGLISH.EXPLANATION from INCOMPLETE.KNOWLEDGE.BASE.RECORD51
Values: "Possible Rationale: IMPERFECT.DOMAIN. The evidence is :
(((The resource ((#[Var P::GET.REVIEWERS>REVIEWERS] . #[Wff P::PAUL.UTGOFF ]))
chosen by the user did not meet specified constraints because the domain is incomplete.)))

Own slot: EXPLANATION.BASIS from INCOMPLETE.KNOWLEDGE.BASE
Values: S:IMPERFECT.DOMAIN

Own slot: INCONSISTENT.STATE.SPEC.IN.FOCUS from INCOMPLETE.KNOWLEDGE.BASE.RECORD51
Values: ISS37

Own slot: POSSIBLE.CHANGES.TO.KB from INCOMPLETE.KNOWLEDGE.BASE.RECORD51
Values: #[Wff AN P::AREAS.OF.EXPERTISE OF P::PAUL.UTGOFF IS P::MACHINE.LEARNING ],
#[Wff A P::KEYWORDS OF P::PAPER.001 IS P::SIMILARITY.BASED.LEARNING ]

Own slot: SELECTED.CHANGE.TO.KB from INCOMPLETE.KNOWLEDGE.BASE.RECORD51
Values: #[Wff AN P::AREAS.OF.EXPERTISE OF P::PAUL.UTGOFF IS P::MACHINE.LEARNING ]

```

Figure 74: The chosen explanation record

```

<From: SPANDEX>
In order to allow the binding choice of PAULUTGOFF
--select an additional fact to assert to restore consistency:

Priority: 1 (AN AREAS.OF.EXPERTISE OF PAULUTGOFF IS MACHINE.LEARNING)
Priority: 2 (A KEYWORDS OF PAPER.001 IS SIMILARITY.BASED.LEARNING)

```

Figure 75: Choosing among new facts to add to restore consistency

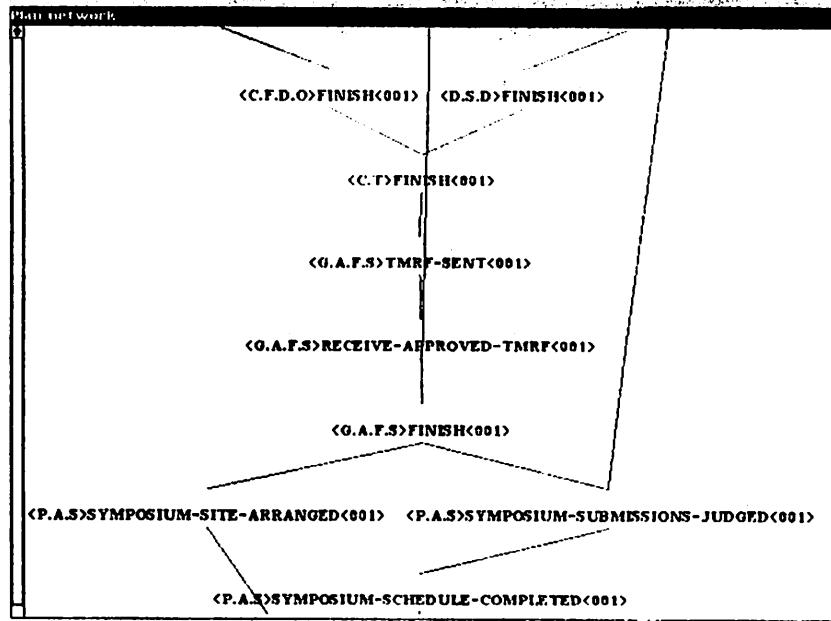


Figure 76: Scenario4: network before exception

der. Intervening steps remain in the plan. This scenario is presented in brief, showing the planner messages, before and after snapshots of the plan network, the selected explanation and the execution record (Figures 78 and 79).

1-202: <Planner expands plan to stage a symposium...>

203: PROCESSING AGENT.EXECUTABLE NODE: <DESIGNATE.SYMPOSIUM.DATES>FOR.AGENT<001>
204: -----

!!! ==> Exception! User designates the site of the symposium instead <==== !!!

205: SPANDEX: user chose the REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT amendment to implement the chosen explanation ACCOMPLISH.LATER.GOAL.STEP.RECORD59 using rationale REORDER.

206: -----

207: SPANDEX: including offline-action DESIGNATE.SYMPOSIUM.SITE to achieve unexpanded goal node <PLAN.A.SYMPOSIUM>SYMPOSIUM-SITE-ARRANGED<001>

208: -----

213: SPANDEX: Reordering <DESIGNATE.SYMPOSIUM.SITE>OFFLINE.BY.AGENT<001> which was accomplished offline and

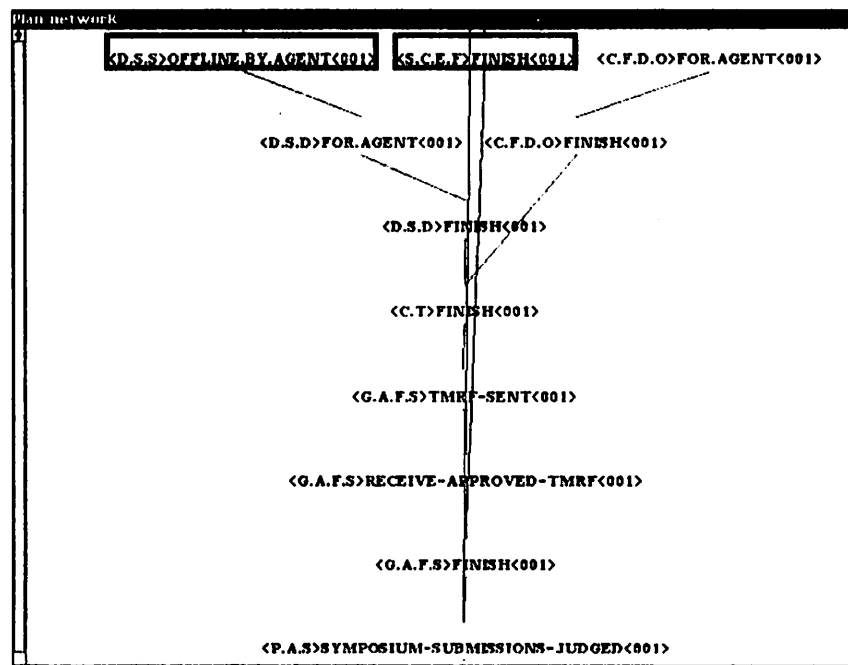


Figure 77: Scenario4: network after exception

214: ... occurred prior to expected node <DESIGNATE.SYMPOSIUM.DATES>FOR.AGENT<001>.
 215: -----
 222: NEXT NODE: <DESIGNATE.SYMPOSIUM.DATES>FOR.AGENT<001>

Variation. Suppose that instead, the user selected the explanation accomplish-later-goal-step-skip-intervening-steps-record259, which is based on the shortcut rationale. The implementation of this explanation involves removing the steps that come after (and including) the expected action, but before the accomplished action. The resulting plan network would be as shown in Figure 80, and the planner messages which follow shows what has been done to implement this alternative explanation.

208: -----
 209: PROCESSING AGENT.EXECUTABLE NODE: <DESIGNATE.SYMPOSIUM.DATES>FOR.AGENT<001>

```

III (Output) The ACCOMPLISH.LATER.GOAL.STEP.RECORD59 Unit in SPANDEX Knowledge Base
Unit: ACCOMPLISH.LATER.GOAL.STEP.RECORD59 in knowledge base SPANDEX
Created by BROVERMAN on 7-6-90 20:53:39
Modified by BROVERMAN on 7-6-90 20:56:21
Member Of: ACCOMPLISH.LATER.GOAL.STEP

Own slot: AMENDMENTS from ACCOMPLISH.LATER.GOAL.STEP
Values: REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT

Own slot: COMPARISON.NODE from ACCOMPLISH.LATER.GOAL.STEP.RECORD59
Values: <PLAN.A.SYMPOSIUM>SYMPOSIUM-SITE-ARRANGED<001> in CONFERENCE

Own slot: ENGLISH.EXPLANATION from ACCOMPLISH.LATER.GOAL.STEP.RECORD59
Values: "Possible Rationale: REORDER. The evidence is:
(((Accomplished-state ((A GEOGRAPHIC.SETTING OF PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS AMHERST)
AND (AN ACCOMODATIONS OF PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS CAMPUS.CENTER)) matches the
later goal ((A GEOGRAPHIC.SETTING OF PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS <PLAN.A.SYMPOSIUM-
SITE) AND (AN ACCOMODATIONS OF PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS <PLAN.A.SYMPOSIUM-H
OTEL)) (of node <PLAN.A.SYMPOSIUM>SYMPOSIUM-SITE-ARRANGED<001>)))"

Own slot: EXPLANATION.BASIS from ACCOMPLISH.LATER.GOAL.STEP
Values: S:REORDER

Own slot: ISS.RECORDS from ACCOMPLISH.LATER.GOAL.STEP.RECORD59
Values: ISS.RECORD 184

Own slot: MATCH.RESULT from ACCOMPLISH.LATER.GOAL.STEP.RECORD59
Values: #S(S:MATCH-RECORD:WORLD-STATES 2 :ACTIVITIES S:UNSPECIFIED-ACT-COMP :ACTIVITY-GOALS S:UNSP
ECIFIED-ACT-COMP :ACHIEVED-STATE-AND-PERFORMED-ACTIVITY-GOAL 1)

```

Figure 78: Scenario4: chosen explanation

```

210: -----
211: SPANDEX: user chose the REPLACE.ACTIVITY.REMOVING.INTERVENING.STEPS
      amendment to implement the chosen explanation
      ACCOMPLISH.LATER.GOAL.STEP.SKIP.INTERVENING.STEPS.RECORD259.
      using rationale SHORTCUT.
212: -----
213: SPANDEX: removing nodes from planner expectations which did not occur
214: prior to the detected unexpected action which has been determined to
      achieve <PLAN.A.SYMPOSIUM>SYMPOSIUM-SITE-ARRANGED<001>.
215: -----
216: SPANDEX: removing node <GET.APPROVAL.FOR.SYMPOSIUM>TMRP-SENT<001> from
      the current network.
217: -----
218: SPANDEX: removing node <GET.APPROVAL.FOR.SYMPOSIUM>RECEIVE-APPROVED-TMRP<001>
      from the current network.
219: -----
220: SPANDEX: removing node <COMPLETE.TMRP>DATE-OVERLAPS-CHECKED<001> from
      the planning history.
221: -----
222: SPANDEX: removing node <CHECK.FOR.DATE.OVERLAPS>FOR.AGENT<001> from
      the current network.
223: -----
224: SPANDEX: removing node <CHECK.FOR.DATE.OVERLAPS>START<001> from
      the current network.

```

```

||| (Output) The 'EXECUTION.RECORD' Unit in SPANDEX Knowledge Base
Unit: 'EXECUTION.RECORD' in knowledge base SPANDEX
Created by BROVERMAN on 3-6-89 13:07:56
Modified by BROVERMAN on 7-7-90 13:23:31
Member Of: ENTITIES in GENERCUNITS

Maintains information about the current execution cycle.

Own slot: ACCOMPLISHED.ACTION from 'EXECUTION.RECORD'
Values: P:DESIGNATE.SYMPOSIUM.SITE

Own slot: ACCOMPLISHED.STATE from 'EXECUTION.RECORD'
Values: #[Wff (A P::GEOGRAPHIC.SETTING OF P::PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS P::AMHERST) A
ND (AN P::ACCOMODATIONS OF P::PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS P::CAMPUS.CENTER) ]

Own slot: COMPLETE.EXPLANATIONS from 'EXECUTION.RECORD'
Values: ACCOMPLISH.LATER.GOAL.STEP.SKIP.INTLIVENING.STEPS.RECORD87,
ACCOMPLISH.LATER.GOAL.STEP.RECORD59,
ADDITIONAL.STEP.TO.ACHIEVE.CURRENT.GOAL.IN.PROGRESS.RECORD51,
ADDITIONAL.STEP.TO.ACHIEVE.CURRENT.GOAL.IN.PROGRESS.RECORD52,
ADDITIONAL.STEP.TO.ACHIEVE.CURRENT.GOAL.IN.PROGRESS.RECORD53,
ADDITIONAL.STEP.TO.ACHIEVE.CURRENT.GOAL.IN.PROGRESS.RECORD64

Own slot: EXCEPTION.TYPE from 'EXECUTION.RECORD'
Values: S:AGENT-RESPONSE-INCONSISTENT-WITH-AGENT-EXECUTABLE-EXPECTATION

Own slot: OVERALL.EXECUTION.RESULT from 'EXECUTION.RECORD'
Values: P:RESOLVED-EXCEPTION

Own slot: SELECTED.EXPLANATION from 'EXECUTION.RECORD'
Values: ACCOMPLISH.LATER.GOAL.STEP.RECORD59

Own slot: SUGGESTED.ACTION from 'EXECUTION.RECORD'
Values: P:DESIGNATE.SYMPOSIUM.DATES

Own slot: SUGGESTED.NODE from 'EXECUTION.RECORD'
Values: <DESIGNATE.SYMPOSIUM.DATES>FOR.AGENT<001> In CONFERENCE

Own slot: SUGGESTED.STATE from 'EXECUTION.RECORD'
Values: #[Wff (A P::MEETING OF P::TMRF<001> IS P::PLANNING.IN.UNCERTAIN.ENVIRONMENTS) AND (A P:
:PROPOSED.MEETING.START.DATE OF P::TMRF<001> IS P::?<DESIGNATE.SYMPOSIUM.DATES>START-DAY
) AND (A P::PROPOSED.MEETING.FINISH.DATE OF P::TMRF<001> IS P::?<DESIGNATE.SYMPOSIUM.DATE
S>FINISH-DAY) ]

```

Figure 79: Scenario4: the execution record

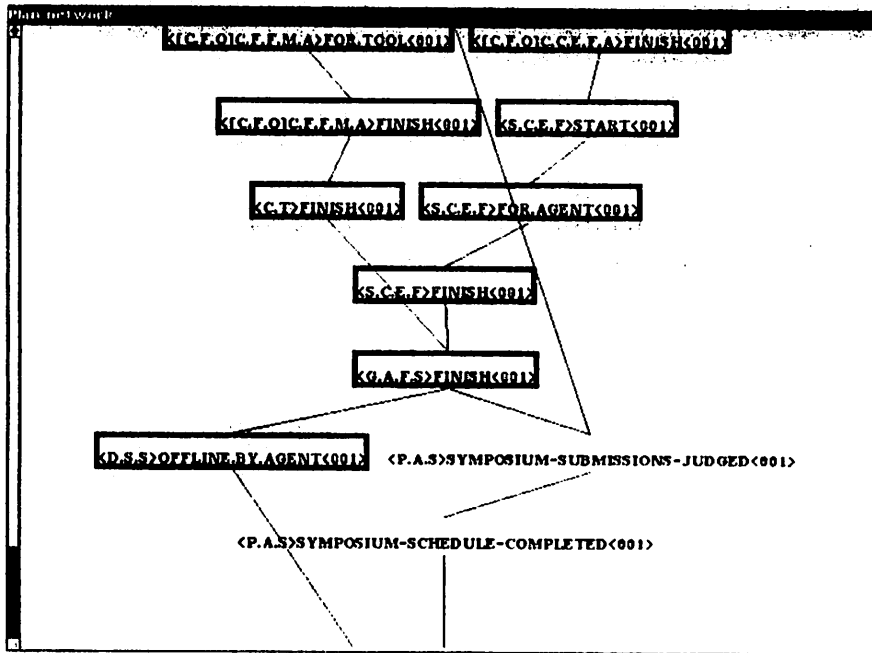


Figure 80: Scenario4: the network after removing intervening steps

225: -----
 226: SPANDEX: removing node <CHECK.FOR.DATE.OVERLAPS>FINISH<001> from
 the current network.
 227: -----
 228: SPANDEX: removing node <COMPLETE.TMRF>DATES-SELECTED<001> from
 the planning history.
 229: -----
 230: SPANDEX: removing node <DESIGNATE.SYMPOSIUM.DATES>FOR.AGENT<001> from
 the current network.
 231: -----
 232: SPANDEX: removing node <DESIGNATE.SYMPOSIUM.DATES>START<001> from
 the current network.
 233: -----
 234: SPANDEX: removing node <DESIGNATE.SYMPOSIUM.DATES>FINISH<001> from
 the current network.
 235: -----
 239: SPANDEX: including offline-action DESIGNATE.SYMPOSIUM.SITE to achieve
 unexpanded goal node <PLAN.A.SYMPOSIUM>SYMPOSIUM-SITE-ARRANGED<001>
 240: -----
 255: NEXT NODE: <PLAN.A.SYMPOSIUM>SYMPOSIUM-SUBMISSIONS-JUDGED<001>)

7.1.5 Scenario 5

This scenario illustrates the handling of an exception where the accomplished action is a possible first step in the achievement of a later unexpanded goal. We illustrate again with planner messages, before and after snapshots of the plan network, and the resulting explanation and execution records. The scenario begins in the context where all the planning is done for staging a symposium, and the actual registration and sessions are about to be held. Instead of conducting registration, the agent elects to send the proceedings to the symposium sponsor; this is not expected.

There are two amendments possible for this explanation. One involves interleaving the later complex activity with the current expectations; i.e. move the first step which has been accomplished to the current execution point, but

leave the expectations of the remainder of the steps for the activity in their original position (i.e. after current expectations). The second amendment involves regressing the entire complex activity which subsumes the performed action to the current execution point. We illustrate what would happen in both cases¹⁰.

Interleaving a later activity

In this section we illustrate system actions when the user chooses an amendment dictating that the later (started) activity should be interleaved with current expectations.

```

1--211: -- >>> planner expands plan for staging a symposium, does all the
           planning and now is ready to hold the actual registration and sessions
212: -----
213: PROCESSING AGENT.EXECUTABLE NODE: <CONDUCT.REGISTRATION>FOR.AGENT<001>
214: -----

!!! -->> EXCEPTION occurs here: user elects to SEND.PROCEEDINGS.TO.SPONSOR instead of
           conducting the registration                                <<----- !!!

215: SPANDEX: user chose the
           INTERLEAVE.LATER.COMPLEX.ACTIVITY.WITH.CURRENT.ACTIVITY.AND.EXPAND.TO.INCLUDE
           amendment to implement the chosen explanation
           PART.OF.LATER.STEP.RECORD616. using rationale REORDER.
216: -----
225: SPANDEX: Interleaving expansion from goal
           <STAGE.A.SYMPOSIUM>SYMPOSIUM-POSTPROCESSING-DONE<001> since its first step:
           SEND.PROCEEDINGS.TO.SPONSOR occurred offline before the expected node
           <CONDUCT.REGISTRATION>FOR.AGENT<001>
216: -----
234: NEXT NODE: <CONDUCT.REGISTRATION>FOR.AGENT<001>

```

¹⁰In actuality, the latter amendment would not be applied in this situation since explicit causal information in this plan requires the conference to be held before a meeting report can be filled out (a subsequent step in the expansion of the activity for the subsuming goal).

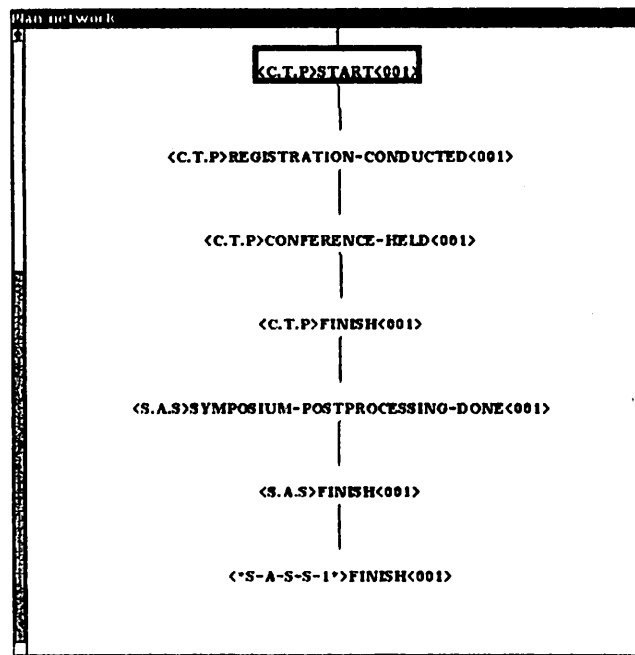


Figure 81: Scenario5: the network before exception

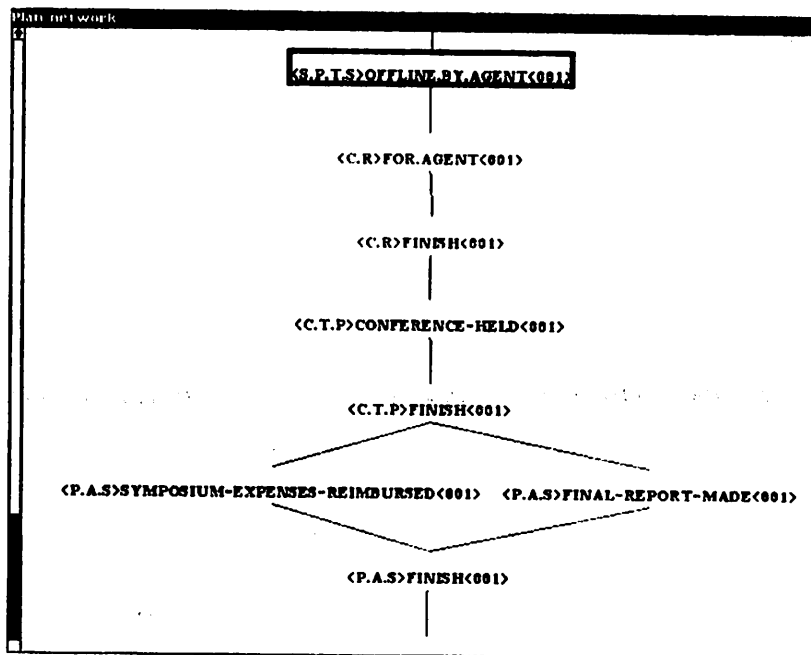


Figure 82: Scenario5: network after exception

[] (Output) The PART OF LATER STEP RECORD 16 Unit in SPANDEX Knowledge Base	
Unit:	PART.OF.LATER.STEP.RECORD16 in knowledge base SPANDEX
Created by:	BROVERMAN on 7-7-90 15:45:15
Modified by:	BROVERMAN on 7-7-90 15:46:32
Member Of:	PART.OF.LATER.STEP
Own slot:	AMENDMENTS from PART.OF.LATER.STEP
Values:	INTERLEAVE.LATER.COMPLEX.ACTIVITY.WITH.CURRENT.ACTIVITY.AND.EXPAND.TO.INCLUDE, MOVE.LATER.COMPLEX.ACTIVITY.TO.CURRENT.EXECUTION.POINT.AND.EXPAND.TO.INCLUDE
Own slot:	COMPARISON.NODE from PART.OF.LATER.STEP.RECORD16
Values:	(STAGE.A.SYMPOSIUM)SYMPOSIUM-POSTPROCESSING-DONE(001) in CONFERENCE
Own slot:	ENGLISH.EXPLANATION from PART.OF.LATER.STEP.RECORD16
Values:	"Possible Rationale: REORDER. The evidence is : (((The achieved state (A PROCEEDINGS.SENT.TO.SPONSOR OF PLANNING.IN.UNCERTAIN.ENVIROMEN TS IS YES) accomplishes a first step in an expansion of the later goal (A POST-CONFERENCE.PAPERWORK OF PLANNING.IN.UNCERTAIN.ENVIROMENTS IS DONE))))"
Own slot:	EXPLANATION.BASIS from PART.OF.LATER.STEP
Values:	S:REORDER
Own slot:	ISS.RECORDS from PART.OF.LATER.STEP.RECORD16
Values:	ISS.RECORD838
Own slot:	MATCH.RESULT from PART.OF.LATER.STEP.RECORD16
Values:	#S(S:MATCH-RECORD:WORLD-STATES 7 :ACTIVITIES S:UNSPECIFIED-ACT-COMP :ACTIVITY-GOALS S :UNSPECIFIED-ACT-COMP :ACHIEVED-STATE-AND-PERFORMED-ACTIVITY-GOAL 1)

Figure 83: Scenario5: chosen explanation

Regressing an entire activity to the current execution point

Here we show how the system would regress the entire activity to take place before current expectations.

```

1--211: -- >>> planner expands plan for staging a symposium, does all the
           planning and now is ready to hold the actual registration and sessions
212: -----
213: PROCESSING AGENT.EXECUTABLE NODE: <CONDUCT.REGISTRATION>FOR.AGENT<001>
214: -----

!!! -->> EXCEPTION occurs here: user elects to SEND.PROCEEDINGS.TO.SPONSOR
           instead of conducting the registration          <<-----    !!!
209: SPANDEX: user chose the
           MOVE.LATER.COMPLEX.ACTIVITY.TO.CURRENT.EXECUTION.POINT.AND.
           EXPAND.TO.INCLUDE
           amendment to implement the chosen explanation
           PART.OF.LATER.STEP.RECORD892. using rationale REORDER.
210: -----
211: SPANDEX: Reordering <STAGE.A.SYMPOSIUM>SYMPOSIUM-POSTPROCESSING-DONE<001>
           since its accomplishment has begun offline
           ... and will be achieved in its entirety prior to expected node
           <CONDUCT.REGISTRATION>FOR.AGENT<001>.
221: -----
224: NEXT NODE: <POSTPROCESS.A.SYMPOSIUM>SYMPOSIUM-EXPENSES-REIMBURSED<001>

```

7.1.6 Scenario 6

This scenario illustrates how the unexpected performance of an activity is handled when the unanticipated activity is a specialization of an expected activity, according the definition specified in Section 6.3.4. It makes use of the rationale Alternative, and shows how facts can be added to the knowledge base as assumptions in order to permit the more specialized activity to take place.

In this scenario, a paper is being reviewed, and the program committee is about to consider all the reviews in order to make a decision. Instead

```

III (Output) The "EXECUTION RECORD" Unit in SPANDEX Knowledge Base
Unit: "EXECUTION.RECORD" in knowledge base SPANDEX
Created by BROVERMAN on 3-6-89 13:07:56
Modified by BROVERMAN on 7-7-90 16:36:28
Member Of: ENTITIES in GENERCOUNTS

Maintains information about the current execution cycle.

Own slot: ACCOMPLISHED.ACTION from "EXECUTION.RECORD"
Values: P:SEND.PROCEEDINGS.TO.SPONSOR

Own slot: ACCOMPLISHED.STATE from "EXECUTION.RECORD"
Values: #[Wff A P:PROCEEDINGS.SENT.TO.SPONSOR OF P:PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS P::YES ]

Own slot: COMPLETE.EXPLANATIONS from "EXECUTION.RECORD"
Values: PART.OF.LATER.STEP.SKIP.INTERVENING.STEPS.RECORD804, PART.OF.LATER.STEP.RECORD892,
ADDITIONAL.STEP.TO.ACHIEVE.CURRENT.GOAL.IN.PROGRESS.RECORD883,
ADDITIONAL.STEP.TO.ACHIEVE.CURRENT.GOAL.IN.PROGRESS.RECORD884

Own slot: EXCEPTION.TYPE from "EXECUTION.RECORD"
Values: S:AGENT-RESPONSE-INCONSISTENT-WITH-AGENT-EXECUTABLE-EXPECTATION

Own slot: OVERALL.EXECUTION.RESULT from "EXECUTION.RECORD"
Values: P:RESOLVED-EXCEPTION

Own slot: SELECTED.AMENDMENT from "EXECUTION.RECORD"
Values: S:MOVE.LATER.COMPLEX.ACTIVITY.TO.CURRENT.EXECUTION.POINT.AND.EXPAND.TO.INCLUDE

Own slot: SELECTED.EXPLANATION from "EXECUTION.RECORD"
Values: PART.OF.LATER.STEP.RECORD892

Own slot: SUGGESTED.ACTION from "EXECUTION.RECORD"
Values: P:CONDUCT.REGISTRATION

Own slot: SUGGESTED.NODE from "EXECUTION.RECORD"
Values: <CONDUCT.REGISTRATION>FOR.AGENT<001> in CONFERENCE

Own slot: SUGGESTED.STATE from "EXECUTION.RECORD"
Values: #[Wff A P:REGISTRATION.COMPLETED OF P:PLANNING.IN.UNCERTAIN.ENVIRONMENTS IS P::YES ]

```

Figure 84: Scenario5: the execution record

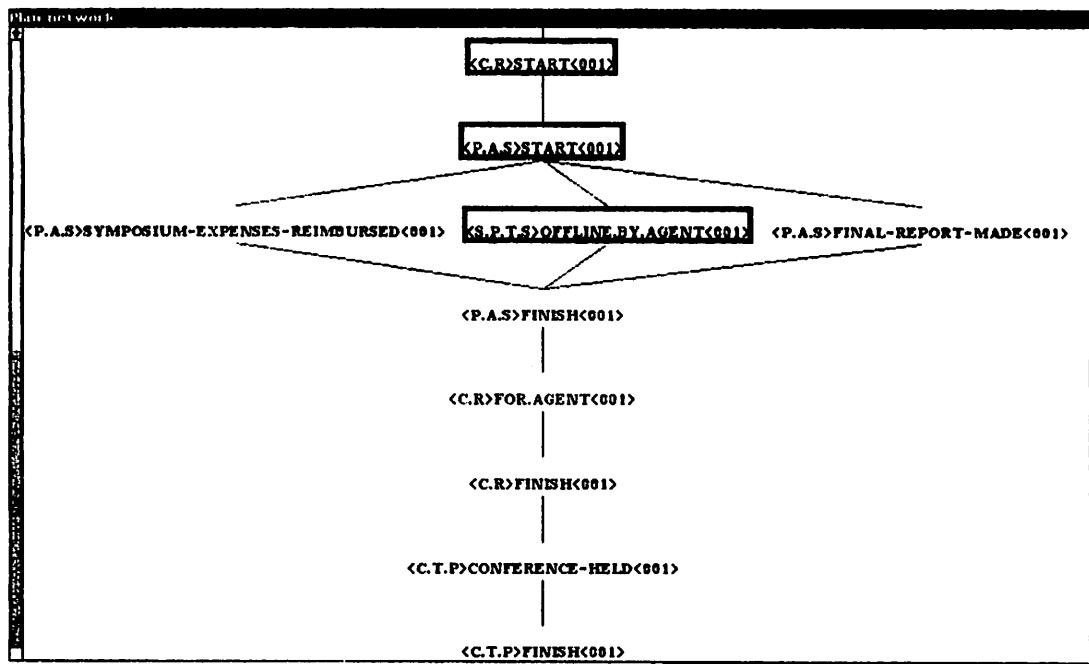


Figure 85: Scenario5: the network after regressing entire activity and expanding

```
{ADVERTISEMENT program.committee.decides.on.paper
 *comment: "Program committee decides whether to accept the paper."
 *agents: ?program-chair
 *input-goal-vars: ?submission
 *goal: and(program.committee.decision.made(?submission,t),
            decision(?submission, ?decision))
 *constraints: member(?submission, submissions)
                member(?decision, possible.decisions)
}
```

```
{ADVERTISEMENT paper.rejected.without.review
 *comment: "Program committee rejects paper because of word count."
 *agents: ?program-chair
 *preconditions: word.count(?submission, over.limit)
 *input-goal-vars: ?submission
 *goal: and(program.committee.decision.made(?submission, t),
            decision(?submission, reject-paper))
 *constraints: member(?submission, submissions)
}
```

Figure 86: Activity specialization

of going ahead with this activity, the activity `paper.rejected.without.review` is invoked. This activity is usually only permitted when the word count exceeds the maximum. In this case, the `word.count` of the paper was specified by the author as `within.limit`, so this activity was not originally suggested. The two activities being discussed are shown in Figure 86.

When the execution monitor detects that an unpredicted activity has taken place, an analysis of the exception is subsequently performed. The exception classifier notes that the accomplished activity has a more specialized precon-

dition as well as more specialized variables in the goal (e.g. a constant as opposed to a variable)¹¹.

The resulting complete explanation is shown in Figure 87. Note also that the values in the match record stored in the match.result slot of the explanations indicate the following comparisons were made between the expected and performed activities (refer back to the values of the dimensions described in Section 6.2):

- World states: Classification 1.2: the achieved and expected world states unify, but are not an exact match.
- Activities: Classification 2.2: the accomplished activity is a specialization of the expected activity.
- Activity-goals: Classification 3.3: Activity goals unify (no exact match) with consistent constraint sets.
- Achieved-state-and-performed-activity-goal: Classification 4.1: the performed activity and the achieved goal are internally consistent.

Since the exception analyst deduced that this activity is a specialization of the expected activity, the precondition is assumed to be true. In particular, it is assumed that the count is indeed over the limit (probably pointed out by one of the reviewers). This fact is changed in the knowledge base, and the performance of this unexpected activity can now be consistently incorporated into the plan.

The planner keeps a record of the actions of the exception handler, as follows:

1-403: >>> Papers are received for review, program committee

¹¹The notion of activity specialization was defined in Section 6.3.4.

```

[[ (Output) The ACCOMPLISH CURRENT STEP RECORD 190 that is SPANDEX Knowledge Base
Unit: ACCOMPLISH.CURRENT.STEP.RECORD.190 in knowledge base SPANDEX
Created by BROVERMAN on 7-14-90 16:16:19
Modified by BROVERMAN on 7-14-90 16:16:21
Member Of: ACCOMPLISH.CURRENT.STEP

Own slot: AMENDMENTS from ACCOMPLISH.CURRENT.STEP
Values: REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT

Own slot: COMPARISON.NODE from ACCOMPLISH.CURRENT.STEP.RECORD.190
Values: <PROGRAM.COMMITTEE.DECIDES.ON.PAPER>FOR.AGENT<001> in CONFERENCE

Own slot: ENGLISH.EXPLANATION from ACCOMPLISH.CURRENT.STEP.RECORD.190
Values: "Possible Rationale: ALTERNATIVE. The evidence is :
(** The performed action PAPER.REJECTED.WITHOUT.REVIEW is a specialization of
PROGRAM.COMMITTEE.DECIDES.ON.PAPER, since it has a more specialized precondition:
#[W/ff A P:WORD.COUNT OF P:(<PAPER.REJECTED.WITHOUT.REVIEW>SUBMISSION IS P:OVER.LIMIT )
versus
NO PRECONDITIONS.
Assume the more specialized precondition holds.
Also has more specialized goal vars -- assume the constraints to hold.
** The accomplished state
((A PROGRAM.COMMITTEE.DECISION.MADE OF SUBMISSION.004 IS T) AND (A DECISION OF SUBMISSION.004 IS REJECT-PAP
ER))
unifies with the suggested state
((A PROGRAM.COMMITTEE.DECISION.MADE OF SUBMISSION.004 IS T) AND (A DECISION OF SUBMISSION.004 IS ?PROGRAM
.COMMITTEE.DECIDES.ON.PAPER>DECISION))."

Own slot: EXPLANATION.BASIS from ACCOMPLISH.CURRENT.STEP
Values: S:ALTERNATIVE

Own slot: INCONSISTENT.STATE.SPEC.IN.FOCUS from ACCOMPLISH.CURRENT.STEP.RECORD.190
Values: ISS149

Own slot: MATCH.RESULT from ACCOMPLISH.CURRENT.STEP.RECORD.190
Values: #S(S:MATCH-RECORD :WORLD-STATES 2 :ACTIVITIES 2 :ACTIVITY-GOALS 3 :ACHIEVED-STATE-AND-PERFORMED-ACTIVI
TY-GOAL 1)

```

Figure 87: Scenario6: A complete explanation illustrating activity specialization

meets to go over the reviews and make decisions....

<<<<

```

-----
410: PROCESSING GOAL NODE:
      <JUDGE.PAPERS.FOR.PRESENTATION>DECISIONS-REACHED<002>
411: -----
412: Selected activity: PROGRAM.COMMITTEE.DECIDES.ON.PAPER for goal
      ((A PROGRAM.COMMITTEE.DECISION.MADE OF SUBMISSION.004 IS T) AND
      (A DECISION OF SUBMISSION.004 IS ?<JUDGE.PAPERS.FOR.PRESENTATION>DECISION))
415: -----
424: PROCESSING AGENT.EXECUTABLE NODE:
      <PROGRAM.COMMITTEE.DECIDES.ON.PAPER>FOR.AGENT<002>
425: -----

!!!! >>> Exception occurs here; agent says to perform
      PAPER.REJECTED.WITHOUT.REVIEW instead.
      Must establish unknown precondition
      that the word count is over the limit. <<<< !!!!!

426: SPANDEX: user chose the
      REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT
      amendment to implement the chosen explanation
      ACCOMPLISH.CURRENT.STEP.RECORD80. using rationale ALTERNATIVE.
427: -----
428: Asserting precondition
      #[Wff A P::WORD.COUNT OF P::SUBMISSION.004 IS P::OVER.LIMIT ]
      of more specialized activity PAPER.REJECTED.WITHOUT.REVIEW which occurred.
      .. and is taking the place of predicted activity
      PROGRAM.COMMITTEE.DECIDES.ON.PAPER
429: -----
430: SPANDEX: removing node <PROGRAM.COMMITTEE.DECIDES.ON.PAPER>FOR.AGENT<002>
      from the current network.
431: -----
432: SPANDEX: removing node <PROGRAM.COMMITTEE.DECIDES.ON.PAPER>START<002>
      from the current network.
433: -----
434: SPANDEX: removing node <PROGRAM.COMMITTEE.DECIDES.ON.PAPER>FINISH<002>
      from the current network.
435: -----
436: Replace with activity: PAPER.REJECTED.WITHOUT.REVIEW
437: for goal
      ((A PROGRAM.COMMITTEE.DECISION.MADE OF SUBMISSION.004 IS T) AND
      (A DECISION OF SUBMISSION.004 IS ?<JUDGE.PAPERS.FOR.PRESENTATION>DECISION))
438: with bindings ((?<judge.papers.for.presentation>decision = Reject-Paper;
      ?<paper.rejected.without.review>submission = Submission.004;
      ?<paper.rejected.without.review>word-count = 4500; )))

```

```

||| (Output) The "EXECUTION RECORD" Unit in SPANDEX Knowledge Base
Unit: *EXECUTION.RECORD* in knowledge base SPANDEX
Created by BROVERMAN on 3-6-89 13:07:56
Modified by BROVERMAN on 7-14-90 16:16:39
Member Of: ENTITIES in GENERICSUNITS

Maintains information about the current execution cycle.

Own slot: ACCOMPLISHED.ACTION from *EXECUTION.RECORD*
Values: P:PAPER.REJECTED.WITHOUT.REVIEW

Own slot: ACCOMPLISHED.STATE from *EXECUTION.RECORD*
Values: #[Wff (A P::PROGRAM.COMMITTEE.DECISION.MADE OF P::SUBMISSION.004 IS T) AND (A P::DECISION OF P::SUBMISSION.004 IS P
::REJECT-PAPER) ]

Own slot: COMPLETE.EXPLANATIONS from *EXECUTION.RECORD*
Values: ACCOMPLISH.CURRENT.STEP.RECORD 190, ACCOMPLISH.CURRENT.STEP.RECORD 198

Own slot: EXCEPTION.TYPE from *EXECUTION.RECORD*
Values: S:AGENT-RESPONSE-INCONSISTENT-WITH-AGENT-EXECUTABLE-EXPECTATION

Own slot: MATCH.RESULT from *EXECUTION.RECORD*
Values: #(S:MATCH-RECORD :WORLD-STATES 2 :ACTIVITIES 2 :ACTIVITY-GOALS 3 :ACHIEVED-STATE-AND-PERFORMED-ACTIVIT
Y-GOAL 1)

Own slot: OVERALL.EXECUTION.RESULT from *EXECUTION.RECORD*
Values: P:RESOLVED-EXCEPTION

Own slot: SELECTED.AMENDMENT from *EXECUTION.RECORD*
Values: S:REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT

Own slot: SELECTED.EXPLANATION from *EXECUTION.RECORD*
Values: ACCOMPLISH.CURRENT.STEP.RECORD 190

Own slot: SUGGESTED.ACTION from *EXECUTION.RECORD*
Values: P:PROGRAM.COMMITTEE.DECIDES.ON.PAPER

Own slot: SUGGESTED.NODE from *EXECUTION.RECORD*
Values: <PROGRAM.COMMITTEE.DECIDES.ON.PAPER>FOR.AGENT<001> in CONFERENCE

Own slot: SUGGESTED.STATE from *EXECUTION.RECORD*
Values: #[Wff (A P::PROGRAM.COMMITTEE.DECISION.MADE OF P::SUBMISSION.004 IS T) AND (A P::DECISION OF P::SUBMISSION.004 IS P
::T<PROGRAM.COMMITTEE.DECIDES.ON.PAPER>DECISION) ]

```

Figure 88: Scenario6: The SPANDEX execution record

Finally, the contents of the execution.record after the handling of the exception is shown in Figure 88.

7.2 Cost of explanation generation

The cost of operations involved in generating explanations for an exception is derived from: (1) the number of rationales which are applicable to a

particular exception type, and (2) the number and complexity of the explanation establishment strategies associated with each of those rationales. In the current prototype of the SPANDEX system, there are eight rationales (including the imperfect-domain explanation basis) which are indexed according to exception types. In some cases, all rationales are applicable (e.g., when the agent response is inconsistent with an executable expectation); in others (e.g. inconsistent-world-during-planning) a single rationale is applicable. The cost of explanation generation for encountering an arbitrary exception can be determined as follows:

We define *evidence.cost* to be the cost of performing a query of the indicator and inconsistency specifications for a single inconsistent state specification (ISS, as described in Section 6.3.3). We then define the cost of applying a single explanation establishment strategy to be

$$\sum_{i=1}^n \textit{evidence.cost}_i$$

for the n ISSes associated with that strategy, and denote this cost as *strategy.cost*. The cost for processing a single rationale is denoted by *rationale.cost*, and is defined to be:

$$\sum_{i=1}^n \textit{strategy.cost}_i$$

for the n strategies associates with that rationale. Finally, the total cost for producing explanations for an exception is denoted by *explanation.cost* and, for the n rationales applicable to the exception type, is defined to be:

$$\sum_{i=1}^n \text{rationale.cost}_i$$

The number of comparison nodes which need to be examined also contributes to the cost of explanation generation. For example, for the shortcut rationale, one of the strategies is to try and match one of the ancestor nodes of the expected action. As the depth and breadth of a hierarchical plan outline can be arbitrary large, and the number of nodes grows exponentially, the brute-force method of explanation generation used in the current prototype of the system is clearly not tractable for an arbitrary network. We have provided heuristics of locality, completeness, and cost to control the search, although we have not needed to employ these heuristics in the use of our prototype since the demonstration of our prototype in our sample domain has never produced a plan outline exceeding 6 levels or containing greater than 60 nodes.

In the scenarios presented in this chapter, the system required between 1-20 seconds to generate a set of plausible explanations. Performance would obviously degrade in applications or scenarios in which a large number of levels are generated during the expansion of the top-level goal node to executable actions, producing exponentially large numbers of comparison nodes. Other factors that would slow the system down would be the introduction of new rationales, indicators or establishment strategies. If performance became a significant problem in the domain of application, further heuristic measures could be developed and applied to the selection of rationales for initial processing.

7.3 Analysis of Survey Results

In order to validate the applicability of the exception classes outlined in this thesis, we designed and distributed a questionnaire to elicit feedback from human conference organizers and individuals who had attended several conferences. The questionnaire is included in full in Appendix F, and describes the typical way of staging a conference¹². A set of attached questions were aimed at eliciting examples of how the actual planning and running of an academic conference can deviate from the general guidelines described.

The questionnaire was distributed to 12 individuals, of whom 9 were faculty members and 3 were advanced graduate students. Five of the surveys were carried out interactively (feedback to the questions was verbal, tape-recorded, and the protocols subsequently analyzed), while the other seven surveys were completed "off-line."

Results

The questions in the first part of the questionnaire were aimed at identifying particular examples of the types of procedural exceptions that can be handled by SPANDEX, with a "catch-all" question to elicit examples of exceptions which may not fall into these classes. Of the 12 surveys completed, 128 examples of exceptions were identified. Of these, 58 exceptions were mentioned only once; the remaining 70 were repeated mentions (by different individuals) of the original 58. All but 7 of these exceptions (3 unique exceptions) were covered by the categories described in this thesis. Figure 89 breaks down the

¹²Based on guidelines from the ACM.

Rationale	Total exceptions identified	Unique exceptions
Alternative	11	8
Alternative resource	45	15
Shortcut	23	11
Reorder	26	12
Reordering shortcut	2	2
Redundant action	14	7
Unclassified	7	3
Total	128	58

Figure 89: Breakdown of identified exceptions by rationale

identified exceptions by the rationales they would be associated with, resulting from recognition by the rationale establishment strategies¹³. We then follow the tabulation with a brief characterization of the actual exceptions that were described. A note is made of cases in which multiple individuals suggested the same thing.

1. Alternative:

- (a) Alternate ways to accomplish submissions-solicited: post in electronic digest, invite people directly to submit (4).
- (b) Establish other ways of subsidizing conference other than registration fees (e.g. get government funding).
- (c) Delegate refereeing into sub-committees rather than having the program chair do all the selection of referees (2).
- (d) Procedure for call for papers varies from publication to publication.
- (e) ACM can do the mailings or you can buy a mailing list.
- (f) A banquet might be skipped in favor of a sponsored reception.

¹³We have omitted examples which would be classified under the New-action rationale from the tabulation since many of the new actions mentioned are a consequence of the inability to represent the plan in full detail for the purposes of this survey. A subset of the actual new actions mentioned (which could be overlooked in a domain expert's original description of the domain as well) are described in the itemization of examples that follow.

(g) Alternate arrangements for transportation might be necessary.

2. Alternative resource:

- (a) No reviewer on program committee with expertise; pick someone (3) else
- (b) Choose a student of someone on program committee as reviewer (5)
- (c) Choose university dorms rather than hotel for housing (1)
- (d) Choose a different site based on climate or ease of access rather than meeting hall size (3)

3. Shortcut:

- (a) Approval process may be skipped. Sometimes this is automatic, some conferences are pre-approved (yearly) (3).
- (b) Advance program may be skipped (3).
- (c) May not get final copies back from authors (2).
- (d) Formal program committee meeting is sometimes skipped (3).
- (e) Final versions are rarely edited (4).
- (f) ACM provides contract services for sub-parts of plan.
- (g) Can skip step of getting explicit agreement from reviewer to review (3).
- (h) Technical sessions can be skipped.
- (i) Invited talk might not happen.
- (j) Approval process not necessary (if there is no sponsoring organization).
- (k) Program committee might do all the reviewing instead of farming out to reviewers.

4. Reorder:

- (a) Approval often comes late in the game (3).
- (b) Usual procedure is to notify accepted, then rejected, but not always in this order.
- (c) Submissions sometimes arrive late.
- (d) Conference proceedings can arrive after conference starts (4).
- (e) Advance program can be done before submissions judged.

- (f) Refereeing can be interleaved with the receipt of submissions (6).
 - (g) Sessions can start before registration ends (4).
 - (h) Registration (pre) can be done earlier (2).
 - (i) Could receive submissions before the call goes out.
 - (j) Can enlist reviewers before papers received.
 - (k) Decisions sometimes get lost (resent) or sent late.
 - (l) Submission can be "accepted" before it's received, if invited.
5. Reordering shortcut:
- (a) Can decide on paper before all reviews have been received.
 - (b) Can mark reviewer as designated before receiving explicit agreement.
6. New action (just a few examples are given here, there were many more):
- (a) Invite some papers as well as referee others.
 - (b) Get keynote speakers.
 - (c) Set up panels.
 - (d) Make book or journal publications out of selected papers (3).
 - (e) Choose program committee.
7. Redundant action:
- (a) May have to iterate over conference approval process (3).
 - (b) Iterate over reviewer selection process to get enough good reviews (3).
 - (c) May need to send out another call if response is low or to change due dates, make corrections, etc. (2).
 - (d) May need to repeat select-site if something goes wrong (2).
 - (e) May "batch out" author notifications in bunches.
 - (f) May need to iterate over conference date selection process (2).
 - (g) A paper may be resubmitted (if modified or lost).

Relaxation	Number of times mentioned
Disjunct addition	15
Conjunct elimination	2
Conjunct to disjunct conversion	2
Constraint elimination	27
Not handled by current methods	6

Figure 90: Breakdown of identified constraint relaxations

The second part of the questionnaire presented an assortment of constraints and asked individuals to respond with alternative specifications for those constraints that they felt were unreasonable. The individuals surveyed were able to propose relaxations for nearly all of them. A breakdown of the responses show that the proposed generalizations fall into the categories shown in Figure 90¹⁴:

A number of the proposed constraint respecifications could not be represented by the generalization techniques discussed in Section 6.4.5.3. For example, some individuals suggested that the constraint on hotel selection (the third constraint given in the survey) should specify "a cluster of hotels with a combined capacity that exceeds the conference registration." This is an altogether different constraint and could not be constructed by one of the above extension techniques.

Another example was a proposed relaxation of the "students cannot be reviewers" constraint. One proposal was that a chosen reviewer can be "a

¹⁴Note: There were no examples of range extension since none of the sample constraints involved ordinal ranges. No examples of taxonomic generalization or taxonomic expansion were given either, since the survey did not specify the domain at the level of the representation; this relaxation is very specific to the implemented representation of the classes involved in the original constraint.

student if that student is supervised by someone on the program committee.” This would be changing $\text{and}(\text{qualified}(\text{?x}, \text{?keyword-area}), \text{not}(\text{student}(\text{?x})))$ to $\text{or}(\text{qualified}(\text{?x}, \text{?keyword-area}), \text{and}(\text{student}(\text{?x}), \text{supervised-by}(\text{?x}, \text{?y}), \text{qualified}(\text{?y}, \text{?keyword-area})))$. Although this modification represents the addition of a disjunct, the disjunct added is a complex statement and not just another simple value. Techniques to acquire and construct more complex generalizations of constraints are needed.

The results described indicate that exceptions and constraint violations indeed would arise during the planning and execution of a complex task such as conference organization, and that people are able to point out several instances by “debugging” a sample task description. Furthermore, most of the exceptions identified could be represented and identified by the techniques described in this thesis.

Problems with the survey design

It is a difficult task to get people to spend a significant amount of time completing a survey, especially when it requires much thought. There was a wide range in the degree of attention which was given to the completion of the survey; many responses had the flavor “If I had more time I could think of some answers... etc.” It seemed difficult for people to fully analyze the task breakdown because of the amount of detail. There was an obvious preference towards focusing on the constraints that were listed since the grain size of information involved was smaller and those questions seemed easier to answer. For example, two of the temporal constraints which appear in the work breakdown structure were specified redundantly in the listing of con-

straints. In many cases, individuals suggested ways to relax the specification in the constraint section of the survey, but did not think to offer the identical specification when asked about it in the context of the task breakdown structure.

In addition, although an attempt was made to make the task breakdown general, too much variation in an overly general specification confounded the analysis of the response. Since there is a great deal of variation in the planning process among different types and sizes of conferences, many responses were directed at pointing out the differences, rather than focusing on exceptional behaviors. For example, many answers followed the form: "now if it were a large conference I would say that this is typical, but for a smaller workshop this step would not be necessary, etc."

Limitations highlighted by the survey

Although many of the exceptional cases pointed out by the individuals who participated in this survey would be covered by the taxonomy and approach described in this thesis, the coverage is by no means complete. The shortcomings are in the following areas:

We cannot tighten "underconstrained" specifications. When the inaccuracies in the task description are due to an underconstrained task representation, the system will not detect the missing constraints. For example, we cannot detect step orderings that are "too loose." Specifically, several individuals mentioned that the conference site should be established before the call for papers is sent out; if this is not represented in the domain descrip-

tion through a causal link, the system will have no information with which to enforce this ordering.

The semantics of temporal constraints in a loosely structured domain is not clear. In the blocks world, one simply cannot put a block on top of another without picking it up first. But in a domain such as conference organization, even though some steps should happen first, they often don't, and the plan may still be able to execute with the desired result in spite of the unanticipated change. In such domains, we really need to represent different levels of criticalities of constraints; i.e., constraints that absolutely cannot be relaxed, those that need to be reachieved if relaxed, or those which are simply used as guidelines, but are flexible.

We can categorize many exceptions, but do not always "understand" them. The techniques associated with the set of rationales used by the system are able to recognize and resolve many types of exceptional behavior that have been identified in this survey. However, just recognizing that an exceptional event allows a shortcut does not capture the reasons why such a thing might be allowed. Knowledge like: a different hotel might be booked "if there was a fire," or that criteria for submission acceptance might change "if the acceptance rate is too high" constitutes common sense or general policy knowledge that is not represented and hard to capture.

We cannot propose entirely different task breakdowns. Some of the feedback from the questionnaire indicated that experts would have broken down the task structure differently than the original domain description. For example, a number of individuals said that the reviewing process could be broken down into a different set of substeps altogether. We would not be able

to recognize such a reconfiguration at run-time unless it were a *variation* of what is already in place, at the same granularity as the original description. In addition, certain types of variations in iteration would not be recognized either. For example, if the original specification of a sub-task breakdown for an activity states that the three subgoals are to iterated over n times as a group (e.g. $(abc)^n$), an exceptional way to accomplish the decomposition that could occur as $(a^n b^n c^n)$ would not be recognized.

Many exceptions are in response to external events. In answering the questions in this survey, people tend to focus primarily on external events that cause problems. However, the response to these events may be purposeful behavior that attempts to rectify the situation. For example, a repeated action might be detected when a hotel is booked after that action has already been accomplished. The reason why this might be done is that there was a fire in the original hotel, causing an agent to take the additional step. The rationale (redundant action) was not the primary motivating force here, but the strategy which explains the role of the action in the plan and the corresponding amendments are nonetheless effective in restoring consistency to the planning and execution of the task.

CHAPTER 8

SUMMARY AND FUTURE DIRECTIONS

In this final chapter, we review the problem which we have addressed, and give a summary of the approach described in this thesis. We outline what we believe to be the unique contributions of the work presented here, as well as point out some of the limitations of the research which has been conducted thus far. Finally, we suggest directions in which this work might be continued.

8.1 Restatement of the Problem

The real world often does not operate in a "typical" and easily specified fashion. Therefore, planning systems which attempt to incorporate static models of complex domains cannot possibly anticipate all the possible ways of performing a task goal. A goal that is largely overlooked in current planning systems is to be able to handle unexpected contingencies as they arise in a planning framework. Although some domain-independent replanning approaches exist [74], in general, current planners are too brittle in the face of unexpected events.

In addition, the assumption adopted by many planning systems that an

autonomous non-rational agent will be the executor of a course of action produced by a planner is too restrictive. The need to relax the assumption of agent autonomy is supported by recent trends in planning research. In many settings total autonomy is not desired, and the human agents involved must still retain a large measure of control. The reason for such a configuration often involves the measure of risk involved; mistakes may be costly and the human must always be the one to make the final decisions. Problem settings in which human interaction is a factor also generally require the interleaving of execution with planning. In these instances, the behavior of human agents should be analyzed critically, especially when it is inconsistent with system expectations. Agents who interact with the system are knowledgeable and behave purposefully, and can thus be viewed as important and accessible sources of knowledge for refining an incompletely specified domain.

When "intelligent" computer support is used to guide human actions (as in an interactive planning system), we are faced with a situation which involves the interaction of disparate knowledge sources: the static knowledge base of the system, and the knowledge possessed by the human agent(s). The system's static knowledge base is explicit and can be manipulated to generate predictions, but is inherently incomplete and incorrect. Human agents have effectively unlimited knowledge bases upon which their actions are based, but this knowledge is dispersed and not explicitly accessible. Measures are needed that will result in an incremental expansion of the initially specified knowledge base so as to better approximate the knowledge of human agents.

Agents perform purposeful actions; their behavior is seldom random. Previous systems have not addressed the notion of attempting to view seemingly

"erroneous" agent actions as contributory events within an evolving planning framework. Recovery measures such as those of [4, 32, 73] and other transformational approaches [60, 59, 49] have been proposed to effectively replan around arbitrary unanticipated domain state changes. But such replanning is primarily a reactionary approach that does not result in improved subsequent system performance. Unexpected user actions can not be undone; and can represent purposeful behavior, improperly categorized as errorful due to an imperfect domain. Current planners do not incorporate constructive measures in order to establish contributory relationships between purposeful user actions and an ongoing plan developed from an imperfect domain.

8.2 Summary of the thesis

In response to the problem characteristics discussed above, in this thesis, we have provided the description of a flexible planning system SPANDEX to help human agents perform tasks. The system goes beyond reactionary approaches when encountering unexpected events or states by looking for ways in which the exceptional occurrence may contribute in a positive way to the ongoing plan.

We focus on the unique role of human agents, and advocate a constructive approach that can extend the boundaries of the initial knowledge base. The natural intelligence and familiarity of humans with the application domain implies that their actions, even when inconsistent with system expectations, are generally purposeful. Human-generated exceptional occurrences, along with information justifying their occurrence, should be incorporated as contribu-

tory events rather than "undone" using traditional replanning techniques. We claim that human-generated events that initially appear to be points of failure may instead represent valid events that appear to be faulty due to the incompleteness and incorrectness of the initial model. Plausible inference about how such actions relate in a contributory fashion to overall plan goals can establish the legitimacy of such actions, and in addition, domain information can be acquired by the system.

The fundamental idea behind the approach presented here is to invoke a type of plausible reasoning as exceptions occur during the use of the system. When an exceptional event occurs, it is viewed as an opportunity to refine the domain model, and a "conduit" is established for the exchange and incorporation of information to justify the exception. The domain model, a theory of human procedural behavior, and information elicited from the user, are applied to transform the current plan into a valid alternative. The primary goal is to provide an explanation which will allow the continuation of planning and execution. As an important side-effect, the explanation process results in new domain knowledge about how tasks can be performed in the domain and the domain objects which are manipulated. By taking this "constructive" approach during exception handling, two important goals are attained: domain knowledge is incrementally refined, and replanning can be avoided or reduced.

In this thesis, we described our methodology for handling points of failure, which we call *exceptions*, as they arise during interactive planning, and showed how it can result in an augmented knowledge base. We presented a theory of human procedural behavior which we use as the basis for understanding how exceptional agent behavior during plan execution can represent new

knowledge about the domain. We also described a framework for interactive planning, and defined the types of exceptions that can be detected within this framework. A general architecture was presented which has been designed and implemented to perform exception handling as described. We showed how the plausible *rationales* identified by our theory are used as the basis for explanation. Corrective measures are proposed in the form of *amendments*, which represent new knowledge to incorporate into the static and dynamic models of the domain. We demonstrated the implementation of these ideas in the SPANDEX system, showing several successfully resolved exceptions from the domain of conference planning. Finally, we presented results from a survey conducted among experienced conference organizers that provided support for the exception types and explanation strategies developed in this thesis.

8.3 Contributions

We consider the contributions of this thesis to fall into five major areas:

8.3.1 Interactive Planning as a Total Man-machine System

In this thesis, we have adopted a unique position in considering the role of human agents as an important component of a planning system. Very few systems which do hierarchical planning consider the human element and the implications of having human agents in the loop. The novel approach presented here considers human agents that interact with a computerized support system to be sources of knowledge that can be exploited when handling exceptions that

arise. We have analyzed studies of human procedural deviations to provide and implement a theory of *rationales* as the basis for plausible explanations of exceptional behavior on the part of human agents.

8.3.2 Expanded Plan Flaw Taxonomy, and a Constructive Approach Towards Their Resolution

As part of this research, we have identified a set of plan flaws that can arise in an interactive planning framework. By pinpointing the sources of interaction with the user, and the possible discrepancies which can result, we have converged on a categorization that subsumes other taxonomies in the planning literature. We also provide measures for the detection and correction of arbitrary constraint violations that can occur during planning. We are therefore able to detect and correct more plan flaws than previous systems.

In addition, we take a constructive rather than reactionary approach in the resolution of discrepancies that arise during planning and execution. Most other systems use the detection of such flaws to prune the search space, or invoke replanning measures to redo parts of the plan. The techniques which we apply in response to plan exceptions look for positive contributions of the unexpected event, and ways in which the plan might subsequently be shortened or revised. The addition of domain knowledge or the inference of unmonitored activity may be necessary in order to complete an explanation.

8.3.3 Graceful Degradation, and the Avoidance of Replanning

One of the primary aims of this research is to provide a knowledge-based, hierarchical, interactive planner with mechanisms to make it more robust when internal or external failure points are detected. In addition to the invocation of standard replanning measures in response to the set of previously identified failure points (e.g. protection interval violations, goal not achieved, phantoms not maintained, etc., which result from arbitrary state changes caused by the external world failure), we resolve exceptions which fall into two primary classes: (a) when the planner is unable to continue planning due to an inconsistency which has been introduced into the world model, and (b) when the result of an interaction with the user is an event which does not conform to the planner's current expectations.

In addition, we provide techniques to guide the planning and execution away from potential failure points. These techniques use constraints to generate valid resource choices for bindings to be employed during task expansion and execution, as well as exploit constraints more fully during activity selection.

The result of all of these measures is more guidance towards correct plan generation and execution, as well as minimal interruption and replanning in response to exceptions that may be encountered along the way. The subsequent continuation of planning after an exception is handled is a significant improvement over the behavior of the system without the exception handler; in the original system configuration an exception would result in a failed plan.

8.3.4 Constraint Relaxation

Other planning systems use the violation of constraints as a way to prune the search space. For example, while SIPE does do general replanning, Wilkins mentions explicitly that no attempt is ever made to relax violated constraints. The ISIS system, on the other hand, does represent predetermined ways to relax specific constraints when trying to devise a job schedule, but is not embedded within a planning context. In addition, in ISIS the relaxations are predefined.

When user actions are interleaved with planner actions, a user action which was inconsistent with the plan cannot be "pruned," since it has already occurred (i.e., you cannot "unring" a bell). Therefore, different measures are needed to incorporate the action into the plan. The approach described in this thesis is unique in that user-generated exceptions that occur during planning and execution dynamically trigger *constraint relaxations* in order to accommodate an exceptional event.

8.3.5 Extending the Domain Model for Improved Performance

The work described in this thesis provides measures to help bridge the gap between system and agent knowledge. A situation to strive for when using knowledge-based systems in interactive and complex domains is one in which the system's model of the domain approximates the domain model of the intelligent agents who interact with the system. We have demonstrated in this thesis an approach that shows how a limited initial domain model can be debugged and enhanced as a result of exception handling, with knowledge acquisition being a side-effect. For example, the technique of *constraint partitioning*

presented in Chapter 6.4.5.1 shows how an inconsistent set of constraints can be made consistent by determining and adding a minimal amount of domain knowledge. Unexpected occurrences during planning and plan execution can be used as a focal point for incremental refinement and extension of the domain model.

This work is a demonstration of how a system can acquire missing domain knowledge in a directed fashion, and suggests ways in which alternative ways to complete task goals can be learned. This newly acquired knowledge is then available for use during future planning activities.

8.4 Directions for Future Research

An initial prototype of an exception handler for an interactive planner has been described in this thesis. In Section 6.6, we discussed specific limitations of the current implementation; in this section, we discuss more general limitations of the prototype and directions in which this research might be continued.

The specification of real world domains is difficult

Our experience with trying to specify a real-world domain (e.g.; conference-planning) in terms of the formalism we have provided have shown it to be difficult. There are no rules by which to guide the knowledge base designer. For example, it is not always clear where the best place is to specify constraints on activity variables if the variables are to be used by activities that will be used to achieve sub-parts of the plan at more detailed levels. In general it seems

that such constraints should be placed on the highest-level activity possible so as to reduce the search space as quickly as possible. On the other hand, this technique, while narrowing the search, widens the possibilities for exceptions to occur at lower levels. An area for future work might be to empirically come up with a set of guidelines for activity specification.

The language for activity specification is also limited and would be easier to use if the expressibility were improved. For example, it is currently not possible to refer to objects that might be used in a later substep using a type of local variable facility, since no such facility is provided. This limitation became obvious when a knowledge base designer using our prototype attempted to write a description for make revisions to a software system. He found that he needed to establish a dummy substep just to set up the modules variable which would be iterated over by the edit substep, which was awkward. One possible solution to this problem would be to augment the current activity description language with a *computed-predicate* facility such as that used by Huff in [34].

In addition, the current implementation does not do any constraint checking of the initial static domain description for inconsistencies. Since the specification of a complex domain is fraught with interrelated detail and thus inherently liable to produce inconsistent dynamic plan instantiations, some form of semantic preprocessing is warranted; such measures could reduce the number of inconsistent situations that can arise during normal planning and execution. To do this effectively, the techniques may prove more than trivial; but an argument can be made for pushing as much semantic consistency checking as possible into the static phases of the system. Also, any changes which are implemented as amendments should be made to existing domain

knowledge in such a way that the knowledge base remains consistent. Similar measures will need to be employed to satisfy both of these aims, and suggested directions may be found by looking at work on knowledge assimilation [37].

Determining when to make permanent changes, or: when does the exception become the rule?

In general, it is not clear how permanent the changes should be that are specified by an amendment. There are different levels possible: changes can be added in the *start world* of the overall task (modifications would be erased at the end of this task instantiation), they can be added in *world* attached to the node currently being processed by the planner (meaning that the models of states which come prior to the current execution point may not become invalid), or changes can be made permanently to the static knowledge base after the task instantiation has concluded so that they persist. Decisions about the level at which to make the changes amounts to a determination of when we think we are making allowances for special cases versus actually debugging the knowledge base. Determining how much context is relevant to remember about the exception is not a simple issue, and can probably not be done in any automated fashion. Currently, the default is to rely upon the user to provide guidance as to whether the changes that are necessary to restore plan consistency should be made permanent or not.

Along the same lines, we need to develop strategies for knowing when and how to generalize an explanation. As part of this, it must be determined when static knowledge should be changed (as opposed to the dynamic plan network

instantiations) and which features should be abstracted from the explanation for generalization.

Reducing system intrusiveness

One of the pervasive problems with the entire approach presented here is the large amount of user interaction that is required to choose and implement explanations of exceptions as they arise. This is a shortcoming and is not a desirable long-term solution since the user may not want to be involved with many interactions at this level. In order for the user to make an informed decision about whether an explanation is reasonable, they are presented with significant amounts of detailed information that they may find intrusive and tedious to sort through. Future work could concentrate on how better to present this information to the user and how to use a sophisticated set of heuristics to automatically choose among explanations. This might include a better interface for browsing among possible explanations.

What other kinds of knowledge are necessary?

Although the approach presented here is often able to show *how* an exceptional event has commonalities with an element of the current plan context, it is not always possible to determine *why* the deviation should be allowed. We adopt hypotheses based on studies of human procedural behavior when possible (e.g. human agents prefer to take shortcuts if possible, etc.), but this set of behavioral explanations is limited and cannot account for all cases. More fundamental domain knowledge that goes beyond "tweakings" of exist-

ing knowledge may be necessary to really explain or excuse an exception. For example, in [9] Borgida gives an example of when to allow a violation of the constraint that says that "an employee may not earn more than their supervisor." Conditions that are given to excuse this exception are if the employee: (a) works overtime, (b) supervisor works part time, or (c) a temporary relationship exists between the employee and their supervisor. Where does this "deeper" knowledge come from? In this case, the user is relied upon to say *why* the exception should be allowed; this information could not be drawn from the knowledge base automatically.

When is an error really an error?

The approach outlined thus far generally assumes that inconsistent behavior on the part of an agent is generally purposeful. However, we also know that humans are prone to making errors, and a more extensive approach would consider models of errorful behavior as well. Error detectors and correctors can be constructed based on the models of human error that were described in Chapter 3.

Being able to handle arbitrary state changes at arbitrary points during the planning and execution of a task

The exception handling facility currently is only able to detect exceptions which occur as a result of execution, resource selection, or as a direct result of a normal planning action. To provide greater flexibility and facilitate the modeling of what-if scenarios and the user of the planner as a simulation tool,

we could extend the model to allow the user to arbitrarily specify state-changes at any point during planning.

Multiple agents

Other directions in which this work could be continued include extending the exception handling algorithms to handle multiple agents. We have initially considered the analysis of an exception in the context of a single overall task, involving a single agent. It is much more complex to consider the analysis of the exceptional behavior in the context of multiple ongoing tasks which are being multiplexed among multiple agents. The definition of wedges of reference that was given in Chapter 6.3.2 could be extended to focus the search on explanations concerning multiple agents.

8.5 Conclusion

We have developed and implemented a system which successfully resolves exceptions that arise during interactive hierarchical nonlinear planning. The key elements of our approach are:

- an analysis and categorization of the types of exceptions that can occur within an interactive planning framework which interleaves execution with planning;
- an understanding about what motivates users to perform actions which are different from system expectations;

- a rich integrated representation for domain activities and objects;
- mechanisms which exploit the domain representation in order to provide constructive guidance to the user and planner;
- a set of algorithms for controlled exploration and modification of the domain representation to construct and implement meaningful explanations for exceptional behavior;
- techniques for constraint relaxation, knowledge base refinement, and plan network modification that restore consistency to an exceptional planning situation in response to a validated explanation.

We have used a corpus of implemented scenarios to demonstrate that the system is able to successfully resolve instances of the types of exceptional agent and planner behavior presented in this thesis. This demonstration, along with the results from user studies, shows that this approach produces a planning system which is less brittle than previous planners which adopt a simpler replanning approach when encountering exceptional occurrences. The SPAN-DEX system focuses on ways in which exceptions can contribute in a positive way to the ongoing plan. It can also be considered more efficient, since by adopting constructive interpretations of exceptions, extensive replanning can be avoided.

APPENDIX A

POLYMER DOMAIN THEORY

A.1 POLYMER Domain Theory

activity	:=	Activity <i>activity-name</i> activity-clause*
activity-clause	:=	goal-clause # [preconditions-clause] # [effects-clause] # [decomposition-clause] # [rationale-clause] # [control-clause] #[agents-clause] # [constraints-clause]
goal-clause	:=	Goal world-predicate
preconditions-clause	:=	Preconditions world-state*
effects-clause	:=	Effects effect-spec*
decomposition-clause	:=	Decomposition step*
rationale-clause	:=	Plan-rationale enables-relation*
control-clause	:=	Control control-construct*
agents-clause	:=	Agents agent-spec*
constraints-clause	:=	Constraints world-state*
step	:=	step-spec [done-by agent-spec]
step-spec	:=	(goal <i>step-name</i> world-state) (activity <i>step-name</i> { <i>activity-name</i> (one-of <i>activity-name</i> *)}) (action <i>step-name</i> <i>action-name</i> parameter-list [world-state])
agent-spec	:=	world-state kb-entity
parameter-list	:=	({variable, }* variable)
control-construct	:=	before{ <i>step-name</i> , }+ <i>step-name</i> if world-state then step-or-net [else step-or-net] optional step-or-net star step-or-net plus step-or-net repeat step-or-net repeat-bounds [iterate-when world-state]
repeat-bounds	:=	while world-state until world-state times <i>integer</i> for variable in ({value, }* value) with variable suchthat world-state
step-or-net	:=	<i>step-name</i> (<i>step-name</i> to <i>step-name</i>)
enables-relation	:=	enables({ <i>step-name</i> , }+ <i>step-name</i>)
effect-spec	:=	(effect-action world-state)
effect-action	:=	set add delete
world-state	:=	predicate (kb-term, kb-term) not(world-state) and({world-state, }* world-state) or({world-state, }* world-state)

kb-term := predicate (kb-term) | *kb-entity* | value | variable
predicate := system-predicate | *kb-predicate*
system-predicate := member | subclass | equal
value := *string* | *number* | symbol-not-a-var
variable := unconstrained-variable | constrained-variable
unconstrained-variable := any symbol whose first character is a "?"
constrained-variable := ?(symbol-not-a-var {world-state | *kb-entity*})
symbol-not-a-var := any symbol whose first character is not a "?"

A.2 Plan Network Definitions

A *plan network* consists of a partially ordered set of *plan nodes*. The plan network structure contains pointers to first and last nodes in the network and a pointer to the activity from which the network was derived. "Successor" links within the nodes define the partial order.

```

plan-network := Plan-network network-name
              start-node # finish-node # from-activity
start-node  := (start plan-node)
finish-node := (finish plan-node)
from-activity := (from-activity activity-name)

```

All plan nodes contain certain generic structural information. The nodes are specialized into four classes:

- *Goal Nodes*: Represent a world state to be achieved;
- *Tool Executable Nodes*: Represent an action which can be performed by a tool in the application domain;
- *Agent Executable Nodes*: Represent an activity which can be carried out by an agent without further decomposition by the planner; and
- *Structural Nodes*: Used by the planner for bookkeeping purposes.

The information contained in the nodes is as follows:

```

plan-node := Plan-node node-name node-info
node-info := goal-node | tool-executable-node |
              agent-executable-node | structural-node
goal-node := generic-node # (type goal) # (goal world-state) #
              (possible-activities activities-and-bindings) #
              (selected-activity activity-name) #
              (expansion-network plan-network)
tool-executable-node := generic-node # (type tool-executable) #
                       (goal [world-state]) #
                       (action fn-name) #
                       (arglist arg-name*)
agent-executable-node := generic-node # (type tool-executable) #
                       (goal [world-state])
structural-node := generic-node # (type structural) #
                  (loop-info loop-controller)

```

```

generic-node := (predecessors (node-name*)) #
               (successors (node-name*)) #
               (from-node node-name) #
               (from-activity activity-name) #
               (before-world world-name) #
               (after-world world-name) #
               (start-time time-range) # (finish-time time-range) #
               (split-type {and | or}) # (join-type {and | or}) #
               (level number) # (step-name step-name) #
               (status { unseen | expanded | pending | phantom |
                        complete | looping }) #
               (repeat-from (node-name*)) #
               (repeat-after (node-name*)) #
               (conditions world-state*) #
               (if-bound-conditions world-state*) #
               (agent agent-spec) # (effects effect-spec*)

activities-and-bindings := (activity-and-bindings*)
activity-and-bindings := (activity-name bindings-list)
bindings-list := ((variable kb-term)*)
loop-controller := Loop-controller controller-name loop-controller-info
loop-controller-info := (initial-list (kb-term*)) #
                       (initial-value kb-term) #
                       (current-list (kb-term*)) # (current-value kb-term) #
                       (variable variable)

time-range := (time-spec, time-spec)
time-spec := number [time-unit]
time-unit := seconds | minutes | hours | days | weeks | years

```

A P P E N D I X B

ALGORITHMS FOR SUGGESTIONS COMPUTATION

B.1 The algorithm for calculation of suggestions for INTERSECTION valueclass constraint

```
FUNCTION intersect-value-class-specs (valueclass-specs)
  "Calculate and combine value-sets containing
   known-positive-examples, known-negative-examples,
   and a measure of enumerability."
  For each valueclass-spec in valueclass-specs
    collect value-sets into all-value-sets such that
      value-set := (known-positive-examples, known-negative-examples,
                    enumerability);
  { result1 := intersect-value-class-sets ((first all-value-sets)
                                           (second all-value-sets));
    result2 := intersect-value-class-sets (result1,
                                           (third all-value-sets));
    result3 := intersect-value-class-sets (result2,
                                           (fourth all-value-sets));
    ....
    until we have run out of elements in all-value-sets}
  Return nth result.
```

B.2 Calculating the Intersection of Two Value Sets

```
FUNCTION intersect-value-class-sets (set1 set2)

  "Takes two possible value sets which are each calculated from a
   value-class restriction, and combines them, assuming an AND
   between the two value sets. Returns 3 values: known positive
   examples, known negative examples, and an indication of
   enumerability."
```

```

IF (set1-positive-examples are enumerable) and
  (set2-positive-examples are enumerable)
THEN { known-positive-examples := intersection
      (set1-positive-examples,
       set2-positive-examples);
      known-negative-examples := union
      (union
       (set-difference
        (set1-positive-examples,
         set2-positive-examples),
        set-difference
        (set2-positive-examples,
         set1-positive-examples)),
       union
        (set1-negative-examples,
         set2-negative-examples));
      enumerability := complete}
ELSE IF (set1-positive-examples are enumerable)
;;(only first set is enumerable, filter using known negatives
  from the partial set)
  THEN {known-positive-examples := set-difference
      (set1-positive-examples,
       set2-negative-examples);
      known-negative-examples := union
      (set1-negative-examples,
       set2-negative-examples);
      enumerability := complete}
ELSE IF (set2-positive-examples are enumerable)
;;(only second set is enumerable, filter using known negatives
  from the partial set)
  THEN { known-positive-examples := set-difference
      (set2-positive-examples,
       set1-negative-examples);
      known-negative-examples := union
      (set1-negative-examples,
       set2-negative-examples);
      enumerability := complete}

```

```

ELSE IF (positive-examples are known for both set1 and set2)
  ;(both are partial sets)
  THEN {known-positive-examples := set-difference
        (intersect
         (set1-positive-examples,
          set2-positive-examples),
         union
         (set1-negative-examples,
          set2-negative-examples));
        known-negative-examples := union
        (set1-negative-values,
         set2-negative-examples);
        enumerability := partial}
ELSE IF (positive-examples are known for set1 only)
  THEN {known-positive-examples := set-difference
        (set1-positive-examples,
         union
         (set1-negative-examples,
          set2-negative-examples));
        known-negative-examples := union
        (set1-negative-values,
         set2-negative-examples);
        enumerability := partial}
ELSE IF (positive-examples are known for set2 only)
  THEN {known-positive-examples := set-difference
        (set2-positive-examples,
         union
         (set1-negative-examples,
          set2-negative-examples));
        known-negative-examples := union
        (set1-negative-values,
         set2-negative-examples);
        enumerability := partial}
ELSE ;;if we have gotten here, no known positive examples are known
  ;; for either set
  {known-positive-examples := the empty set;
   known-negative-examples := union(set1-negative-examples,
                                     set2-negative-examples);
   enumerability := partial};

```

APPENDIX C
SPANDEX STRUCTURES

***Rationale*:** IMPERFECT.DOMAIN

***Establishment.strategies*:**

INCOMPLETE.KNOWLEDGE.BASE

***Inconsistent-state-spec*:** ISS37

***Inconsistencies*:** none

***Indicators*:**

IND: Establishing-fn: INCOMPLETE-DOMAIN?

When applicable?: Always.

***Amendments*:**

ADD.TO.KB.TO.SATISFY.CONSTRAINTS

Implement: (ADD-TO-KB-TO-SATISFY-CONSTRAINTS)

INCORRECT.KNOWLEDGE.BASE

***Inconsistent-state-spec*:** ISS38

***Inconsistencies*:** none

***Indicators*:**

IND: Establishing-fn: INCORRECT-DOMAIN?

When applicable?: Always.

***Amendments*:**

MODIFY.CONSTRAINTS.TO.FIT.KB

Implement: (MODIFY-CONSTRAINTS-TO-FIT-KB)

***Rationale*:** ALTERNATIVE

***Establishment.strategies*:**

ACCOMPLISH.CURRENT.STEP

***Inconsistent-state-spec*:** ISS3

***Inconsistencies*:** none

***Indicators*:**

IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-SUGGESTED-STATE?

When applicable?: (ACHIEVED-STATE-KNOWN?)

IND: Establishing-fn: ACCOMPLISHED-ACTION-PRECONDITION-
SPECIALIZATION-OF-SUGGESTED-NODE?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

***Inconsistent-state-spec*: ISS4**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-SUGGESTED-STATE?
 When applicable?: (ACHIEVED-STATE-KNOWN?)
 IND: Establishing-fn: ACCOMPLISHED-ACTION-GOAL-VARS-SPECIALIZATION-OF-SUGGESTED-NODE?
 When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

***Inconsistent-state-spec*: ISS5**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-BEING-COMPARED?
 When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

***Inconsistent-state-spec*: ISS6**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: SUPPLIED-PARAMETER-SAME-CLASS-AS-EXPECTED-PARAMETER?
 When applicable?: (ACHIEVED-STATE-KNOWN? PARAMETER-INCONSISTENCY?)

***Inconsistent-state-spec*: ISS7**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-SUGGESTED-STATE?
 When applicable?: (ACHIEVED-STATE-KNOWN?)

***Amendments*:**
 REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT
 Implement: (REPLACE-ACTIVITY)
 Implementation.args: (REMOVE-INTERVENING-NODES? NIL)

ACCOMPLISH.CONCURRENT.STEP

***Inconsistent-state-spec*: ISS8**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-UNSELECTED-READY-ACTION-STATE?
 When applicable?: (ACHIEVED-STATE-KNOWN?)
 IND: Establishing-fn: ACCOMPLISHED-ACTION-MATCHES-UNSELECTED-READY-ACTION?
 When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

***Inconsistent-state-spec*: ISS9**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-UNSELECTED-READY-ACTION-STATE?
 When applicable?: (ACHIEVED-STATE-KNOWN?)
 IND: Establishing-fn: ACCOMPLISHED-ACTION-PRECONDITION-SPECIALIZATION-OF-UNSELECTED-READY-ACTION?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

Inconsistent-state-spec: ISS10

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-UNSELECTED-READY-ACTION-STATE?

When applicable?: (ACHIEVED-STATE-KNOWN?)

IND: Establishing-fn: ACCOMPLISHED-ACTION-GOAL-VARS-SPECIALIZATION-OF-UNSELECTED-READY-ACTION?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

Inconsistent-state-spec: ISS11

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-BEING-COMPARED?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

Inconsistent-state-spec: ISS12

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-UNSELECTED-READY-ACTION-STATE?

When applicable?: (ACHIEVED-STATE-KNOWN?)

Amendments:

REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT

Implement: (REPLACE-ACTIVITY)

Implementation.args: (REMOVE-INTERVENING-NODES? NIL)

***Rationale*: SHORTCUT**

***Establishment.strategies*:**

ACCOMPLISH.LATER.GOAL.STEP.SKIP.INTERVENING.STEPS

Inconsistent-state-spec: ISS19

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-LATER-GOAL-NODE?

When applicable?: (ACHIEVED-STATE-KNOWN? GOAL-COMP-NODE?)

Inconsistent-state-spec: ISS20

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-BEING-COMPARED?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

Amendments:

REPLACE.ACTIVITY.REMOVING.INTERVENING.STEPS

Implement: (REPLACE-ACTIVITY)

Implementation.args: (REMOVE-INTERVENING-NODES? T)

SUBSUMES.CURRENT.STEP

***Inconsistent-state-spec*: ISS21**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-BEING-COMPARED?
 When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

***Inconsistent-state-spec*: ISS22**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-GOAL-SUBSUMING-SUGGESTED-ACTION?
 When applicable?: (ACHIEVED-STATE-KNOWN?)

***Amendments*:**
 REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT
 Implement: (REPLACE-ACTIVITY)
 Implementation.args: (REMOVE-INTERVENING-NODES? NIL)
 SUBSUMES.CONCURRENT.STEP

***Inconsistent-state-spec*: ISS23**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-BEING-COMPARED?
 When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

***Inconsistent-state-spec*: ISS24**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-GOAL-SUBSUMING-UNSELECTED-READY-ACTION?
 When applicable?: (ACHIEVED-STATE-KNOWN?)

***Amendments*:**
 REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT
 Implement: (REPLACE-ACTIVITY)
 Implementation.args: (REMOVE-INTERVENING-NODES? NIL)
 ACCOMPLISH.LATER.ACTION.STEP.SKIP.INTERVENING.STEPS

***Inconsistent-state-spec*: ISS17**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-BEING-COMPARED?
 When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

***Inconsistent-state-spec*: ISS18**
***Inconsistencies*: none**
***Indicators*:**
 IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-LATER-GOAL-NODE?
 When applicable?: (ACHIEVED-STATE-KNOWN? GOAL-COMP-NODE?)

***Amendments*:**
 REPLACE.ACTIVITY.REMOVING.INTERVENING.STEPS
 Implement: (REPLACE-ACTIVITY)

Implementation.args: (REMOVE-INTERVENING-NODES? T)
SUBSUMES.LATER.STEP.SKIP.INTERVENING.STEPS
Inconsistent-state-spec: ISS27
Inconsistencies: none
Indicators:
IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-BEING-COMPARED?
When applicable?: (PERFORMED-ACTIVITY-KNOWN?)
Inconsistent-state-spec: ISS28
Inconsistencies: none
Indicators:
IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-GOAL-SUBSUMING-
LATER-NODE?
When applicable?: (ACHIEVED-STATE-KNOWN?)
Amendments:
REPLACE.ACTIVITY.REMOVING.INTERVENING.STEPS
Implement: (REPLACE-ACTIVITY)
Implementation.args: (REMOVE-INTERVENING-NODES? T)
PART.OF.LATER.STEP.SKIP.INTERVENING.STEPS
Inconsistent-state-spec: ISS31
Inconsistencies: none
Indicators:
IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-
BEING-COMPARED?
When applicable?: (PERFORMED-ACTIVITY-KNOWN?)
Inconsistent-state-spec: ISS32
Inconsistencies: none
Indicators:
IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-GOAL-IN-
EXPANSION-OF-LATER-NODE?
When applicable?: (ACHIEVED-STATE-KNOWN?)
Amendments:
INCLUDE.IN.EXPANSION.OF.LATER.COMPLEX.ACTIVITY.REMOVING.
INTERVENING.STEPS
Implement: (EXPAND-LATER-COMPLEX-ACTIVITY-TO-INCLUDE)
Implementation.args: (REMOVE-INTERVENING-NODES? T REORDER? NIL
INTERLEAVE? NIL)

Rationale: REORDER

Establishment.strategies:
ACCOMPLISH.LATER.GOAL.STEP
Inconsistent-state-spec: ISS15
Inconsistencies: none
Indicators:
IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-LATER-GOAL-NODE?
When applicable?: (ACHIEVED-STATE-KNOWN? GOAL-COMP-NODE?)
Inconsistent-state-spec: ISS16

```

*Inconsistencies*: none
*Indicators*:
  IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-
    BEING-COMPARED?
    When applicable?: (PERFORMED-ACTIVITY-KNOWN?)
*Amendments*:
  REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT
  Implement: (REPLACE-ACTIVITY)
  Implementation.args: (REMOVE-INTERVENING-NODES? NIL)
ACCOMPLISH.LATER.ACTION.STEP
*Inconsistent-state-spec*: ISS13
  *Inconsistencies*: none
  *Indicators*:
    IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-
      BEING-COMPARED?
      When applicable?: (PERFORMED-ACTIVITY-KNOWN?)
*Inconsistent-state-spec*: ISS14
  *Inconsistencies*: none
  *Indicators*:
    IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-LATER-GOAL-NODE?
      When applicable?: (ACHIEVED-STATE-KNOWN? GOAL-COMP-NODE?)
*Amendments*:
  REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT
  Implement: (REPLACE-ACTIVITY)
  Implementation.args: (REMOVE-INTERVENING-NODES? NIL)
PART.OF.LATER.STEP
*Inconsistent-state-spec*: ISS29
  *Inconsistencies*: none
  *Indicators*:
    IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-
      BEING-COMPARED?
      When applicable?: (PERFORMED-ACTIVITY-KNOWN?)
*Inconsistent-state-spec*: ISS30
  *Inconsistencies*: none
  *Indicators*:
    IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-GOAL-IN-
      EXPANSION-OF-LATER-NODE?
      When applicable?: (ACHIEVED-STATE-KNOWN?)
*Amendments*:
  INTERLEAVE.LATER.COMPLEX.ACTIVITY.WITH.CURRENT.ACTIVITY.AND.EXPAND.
  TO.INCLUDE
  Implement: (EXPAND-LATER-COMPLEX-ACTIVITY-TO-INCLUDE)
  Implementation.args: (REMOVE-INTERVENING-NODES? NIL REORDER? T
    INTERLEAVE?
    T)
  MOVE.LATER.COMPLEX.ACTIVITY.TO.CURRENT.EXECUTION.POINT.AND.EXPAND.
  TO.INCLUDE
  Implement: (EXPAND-LATER-COMPLEX-ACTIVITY-TO-INCLUDE)

```

Implementation.args: (REMOVE-INTERVENING-NODES? NIL REORDER? T
 INTERLEAVE?
 NIL)

Rationale: REORDERING.SHORTCUT

Establishment.strategies:

SUBSUMES.LATER.STEP

Inconsistent-state-spec: ISS25

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-
 BEING-COMPARED?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

Inconsistent-state-spec: ISS26

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-STATE-MATCHES-GOAL-SUBSUMING-
 LATER-NODE?

When applicable?: (ACHIEVED-STATE-KNOWN?)

Amendments:

REPLACE.ACTIVITY.AND.PLACE.AT.CURRENT.EXECUTION.POINT

Implement: (REPLACE-ACTIVITY)

Implementation.args: (REMOVE-INTERVENING-NODES? NIL)

Rationale: NEW.ACTION

Establishment.strategies:

ADDITIONAL.STEP.TO.ACHIEVE.CURRENT.GOAL.IN.PROGRESS

Inconsistent-state-spec: ISS33

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-ACTION-AND-STATE-CONSISTENT-WITH-
 CURRENT-GOAL?

When applicable?: Always.

Amendments:

ADD.AS.PART.OF.CURRENT.COMPLEX.ACTIVITY

Implement: (ADD-AS-PART-OF-CURRENT-COMPLEX-ACTIVITY)

ADDITIONAL.STEP.TO.ACHIEVE.CONCURRENT.GOAL.IN.PROGRESS

Inconsistent-state-spec: ISS34

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-ACTION-AND-STATE-CONSISTENT-WITH-
 CONCURRENT-GOAL?

When applicable?: Always.

Amendments:

ADD.AS.PART.OF.CONCURRENT.COMPLEX.ACTIVITY
 Implement: (ADD-AS-PART-OF-CONCURRENT-COMPLEX-ACTIVITY)

Rationale: ALTERNATIVE.RESOURCE

Establishment.strategies:

Rationale: REDUNDANT.ACTION

Establishment.strategies:

REPEATED.ACTION.STRING

Inconsistent-state-spec: ISS39

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-
 BEING-COMPARED?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

Inconsistent-state-spec: ISS40

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACHIEVED-STATE-MATCHES-GOAL-SUBSUMING-
 EXECUTED-STRING?

When applicable?: (ACHIEVED-STATE-KNOWN?)

Amendments:

INSERT.REDUNDANT.ACTION

Implement: (INSERT-REDUNDANT-ACTION)

REPEATED.ACTION

Inconsistent-state-spec: ISS35

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACCOMPLISHED-EFFECTS-MATCH-STATE-
 BEING-COMPARED?

When applicable?: (PERFORMED-ACTIVITY-KNOWN?)

Inconsistent state-spec: ISS36

Inconsistencies: none

Indicators:

IND: Establishing-fn: ACHIEVED-STATE-MATCHES-GOAL-OF-EXECUTED-STEP?

When applicable?: (ACHIEVED-STATE-KNOWN?)

Amendments:

INSERT.REDUNDANT.ACTION

Implement: (INSERT-REDUNDANT-ACTION)

A P P E N D I X D

EXAMPLE CONSTRAINT PARTITIONS

Here are the set of actual constraint partitions that were computed for the example in Section 6.4.5.1. For $N = 7$ constraints with *primary-constraint-set-size* (M) set to 1, seven partitions were initially calculated. Three remain viable after filtering. The partition which generated the selected KB addition is Partition 1.

```
=====
Partitions Generated for Node: <SELECT.A.SITE>FOR.AGENT<001>
=====
```

(PARTITION 1:)

Primary-constraints: #[Wff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE
IS ?<SELECT.A.SITE>SEASON]

Primary-bindings: ((?<SELECT.A.SITE>SITE . BOSTON)
(?<SELECT.A.SITE>SEASON . FALL))
(?<SELECT.A.SITE>SITE . BOSTON)
(?<SELECT.A.SITE>SEASON . SPRING))
(?<SELECT.A.SITE>SITE . PITTSBURGH)
(?<SELECT.A.SITE>SEASON . SPRING))
(?<SELECT.A.SITE>SITE . DENVER)
(?<SELECT.A.SITE>SEASON . WINTER))

Secondary-constraints: #[Wff SIGMOD IS IN CLASS CONFERENCE]
#[Wff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS
UNIVERSITY]
#[Wff ?<SELECT.A.SITE>SITE IS IN CLASS CITY]
#[Wff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY
IS ?<SELECT.A.SITE>SITE]
#[Wff A MONTH OF SIGMOD IS
?<SELECT.A.SITE>MONTH]
#[Wff A SEASON OF ?<SELECT.A.SITE>MONTH IS
?<SELECT.A.SITE>SEASON]

Secondary-bindings: ((?<SELECT.A.SITE>SEASON . SPRING)

(?<SELECT.A.SITE>MONTH . APRIL)
 (?<SELECT.A.SITE>SITE . DENVER)
 (?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

=====
 (PARTITION 2:)

Primary-constraints: #[Wff A SEASON OF ?<SELECT.A.SITE>MONTH IS
 ?<SELECT.A.SITE>SEASON]

Primary-bindings: ((?<SELECT.A.SITE>MONTH . DECEMBER)
 (?<SELECT.A.SITE>SEASON . WINTER))
 ((?<SELECT.A.SITE>MONTH . OCTOBER)
 (?<SELECT.A.SITE>SEASON . FALL))
 ((?<SELECT.A.SITE>MONTH . SEPTEMBER)
 (?<SELECT.A.SITE>SEASON . FALL))
 ((?<SELECT.A.SITE>MONTH . AUGUST)
 (?<SELECT.A.SITE>SEASON . SUMMER))
 ((?<SELECT.A.SITE>MONTH . JULY)
 (?<SELECT.A.SITE>SEASON . SUMMER))
 ((?<SELECT.A.SITE>MONTH . JUNE)
 (?<SELECT.A.SITE>SEASON . SPRING))
 ((?<SELECT.A.SITE>MONTH . MAY)
 (?<SELECT.A.SITE>SEASON . SPRING))
 ((?<SELECT.A.SITE>MONTH . APRIL)
 (?<SELECT.A.SITE>SEASON . SPRING))
 ((?<SELECT.A.SITE>MONTH . MARCH)
 (?<SELECT.A.SITE>SEASON . WINTER))
 ((?<SELECT.A.SITE>MONTH . FEBRUARY)
 (?<SELECT.A.SITE>SEASON . WINTER))
 ((?<SELECT.A.SITE>MONTH . JANUARY)
 (?<SELECT.A.SITE>SEASON . WINTER))

Secondary-constraints: #[Wff SIGMOD IS IN CLASS CONFERENCE]
 #[Wff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS
 UNIVERSITY]
 #[Wff ?<SELECT.A.SITE>SITE IS IN CLASS CITY]
 #[Wff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY
 IS ?<SELECT.A.SITE>SITE]
 #[Wff A MONTH OF SIGMOD IS
 ?<SELECT.A.SITE>MONTH]
 #[Wff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE IS
 ?<SELECT.A.SITE>SEASON]

Secondary-bindings: ((?<SELECT.A.SITE>SEASON . WINTER)
 (?<SELECT.A.SITE>MONTH . APRIL)
 (?<SELECT.A.SITE>SITE . DENVER)
 (?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

=====

(PARTITION 3:)

Primary-constraints: #[Wff A MONTH OF SIGMOD IS ?<SELECT.A.SITE>MONTH]

Primary-bindings: ((?<SELECT.A.SITE>MONTH . APRIL))

Secondary-constraints: #[Wff SIGMOD IS IN CLASS CONFERENCE]

#[Wff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS
UNIVERSITY]

#[Wff ?<SELECT.A.SITE>SITE IS IN CLASS CITY]

#[Wff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY IS
?<SELECT.A.SITE>SITE]

#[Wff A SEASON OF ?<SELECT.A.SITE>MONTH IS
?<SELECT.A.SITE>SEASON]

#[Wff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE IS
?<SELECT.A.SITE>SEASON]

Secondary-bindings: ((?<SELECT.A.SITE>MONTH . DECEMBER)

(?<SELECT.A.SITE>SEASON . WINTER)

(?<SELECT.A.SITE>SITE . DENVER)

(?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

((?<SELECT.A.SITE>MONTH . MARCH)

(?<SELECT.A.SITE>SEASON . WINTER)

(?<SELECT.A.SITE>SITE . DENVER)

(?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

((?<SELECT.A.SITE>MONTH . FEBRUARY)

(?<SELECT.A.SITE>SEASON . WINTER)

(?<SELECT.A.SITE>SITE . DENVER)

(?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

((?<SELECT.A.SITE>MONTH . JANUARY)

(?<SELECT.A.SITE>SEASON . WINTER)

(?<SELECT.A.SITE>SITE . DENVER)

(?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

=====
(PARTITION 4:)

Primary-constraints: #[Wff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY IS
?<SELECT.A.SITE>SITE]

Primary-bindings: ((?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO)
(?<SELECT.A.SITE>SITE . DENVER))

((?<SELECT.A.SITE>UNIVERSITY . TUFTS.UNIVERSITY)

(?<SELECT.A.SITE>SITE . BOSTON))

((?<SELECT.A.SITE>UNIVERSITY . CARNEGIE.MELLON.UNIVERSITY)

(?<SELECT.A.SITE>SITE . PITTSBURGH))

Secondary-constraints: #[Wff SIGMOD IS IN CLASS CONFERENCE]

#[Wff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS
UNIVERSITY]

#[Wff ?<SELECT.A.SITE>SITE IS IN CLASS CITY]


```

#[Wff A MONTH OF SIGMOD IS ?<SELECT.A.SITE>MONTH ]
#[Wff A SEASON OF ?<SELECT.A.SITE>MONTH IS
?<SELECT.A.SITE>SEASON ]
#[Wff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE IS
?<SELECT.A.SITE>SEASON ]

```

Secondary-bindings: NIL

=====
(PARTITION 5:)

Primary-constraints: #[Wff ?<SELECT.A.SITE>SITE IS IN CLASS CITY]

Primary-bindings: ((?<SELECT.A.SITE>SITE . DENVER))
((?<SELECT.A.SITE>SITE . PITTSBURGH))
((?<SELECT.A.SITE>SITE . BOSTON))

Secondary-constraints:#[Wff SIGMOD IS IN CLASS CONFERENCE]

#[Wff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS
UNIVERSITY]

#[Wff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY IS
?<SELECT.A.SITE>SITE]

#[Wff A MONTH OF SIGMOD IS ?<SELECT.A.SITE>MONTH]

#[Wff A SEASON OF ?<SELECT.A.SITE>MONTH IS
?<SELECT.A.SITE>SEASON]

#[Wff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE IS
?<SELECT.A.SITE>SEASON]

Secondary-bindings: NIL

=====
(PARTITION 6:)

Primary-constraints: #[Wff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS
UNIVERSITY]

Primary-bindings: ((?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))
((?<SELECT.A.SITE>UNIVERSITY . TUFTS.UNIVERSITY))
((?<SELECT.A.SITE>UNIVERSITY . CARNEGIE.MELLON.UNIVERSITY))

Secondary-constraints: #[Wff SIGMOD IS IN CLASS CONFERENCE]

#[Wff ?<SELECT.A.SITE>SITE IS IN CLASS CITY]

#[Wff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY IS
?<SELECT.A.SITE>SITE]

#[Wff A MONTH OF SIGMOD IS ?<SELECT.A.SITE>MONTH]

#[Wff A SEASON OF ?<SELECT.A.SITE>MONTH IS
?<SELECT.A.SITE>SEASON]

#[Wff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE IS
?<SELECT.A.SITE>SEASON]

Secondary-bindings: NIL

=====

(PARTITION 7:)

Primary-constraints: #[Wff SIGMOD IS IN CLASS CONFERENCE]
 Primary-bindings: NIL
 Secondary-constraints: #[Wff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS
 UNIVERSITY]
 #[Wff ?<SELECT.A.SITE>SITE IS IN CLASS CITY]
 #[Wff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY IS
 ?<SELECT.A.SITE>SITE]
 #[Wff A MONTH OF SIGMOD IS ?<SELECT.A.SITE>MONTH]
 #[Wff A SEASON OF ?<SELECT.A.SITE>MONTH IS
 ?<SELECT.A.SITE>SEASON]
 #[Wff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE IS
 ?<SELECT.A.SITE>SEASON]

Secondary-bindings: NIL

=====
 =====
 (s:heuristically-filter-partitions) ==> PARTITIONS WHICH SURVIVE FILTERING (3)
 =====
 =====

(PARTITION 1:)

Primary-constraints: #[Wff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE
 IS ?<SELECT.A.SITE>SEASON]
 Primary-bindings: ((?<SELECT.A.SITE>SITE . BOSTON)
 (?<SELECT.A.SITE>SEASON . FALL))
 ((?<SELECT.A.SITE>SITE . BOSTON)
 (?<SELECT.A.SITE>SEASON . SPRING))
 ((?<SELECT.A.SITE>SITE . PITTSBURGH)
 (?<SELECT.A.SITE>SEASON . SPRING))
 ((?<SELECT.A.SITE>SITE . DENVER)
 (?<SELECT.A.SITE>SEASON . WINTER))

Secondary-constraints: #[Wff SIGMOD IS IN CLASS CONFERENCE]
 #[Wff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS
 UNIVERSITY]
 #[Wff ?<SELECT.A.SITE>SITE IS IN CLASS CITY]
 #[Wff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY
 IS ?<SELECT.A.SITE>SITE]
 #[Wff A MONTH OF SIGMOD IS
 ?<SELECT.A.SITE>MONTH]
 #[Wff A SEASON OF ?<SELECT.A.SITE>MONTH IS
 ?<SELECT.A.SITE>SEASON]

Secondary-bindings: ((?<SELECT.A.SITE>SEASON . SPRING)

(?<SELECT.A.SITE>MONTH . APRIL)
 (?<SELECT.A.SITE>SITE . DENVER)
 (?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

=====
 (PARTITION 2:)

Primary-constraints: #[Wff A SEASON OF ?<SELECT.A.SITE>MONTH IS
 ?<SELECT.A.SITE>SEASON]

Primary-bindings: ((?<SELECT.A.SITE>MONTH . DECEMBER)
 (?<SELECT.A.SITE>SEASON . WINTER))
 ((?<SELECT.A.SITE>MONTH . OCTOBER)
 (?<SELECT.A.SITE>SEASON . FALL))
 ((?<SELECT.A.SITE>MONTH . SEPTEMBER)
 (?<SELECT.A.SITE>SEASON . FALL))
 ((?<SELECT.A.SITE>MONTH . AUGUST)
 (?<SELECT.A.SITE>SEASON . SUMMER))
 ((?<SELECT.A.SITE>MONTH . JULY)
 (?<SELECT.A.SITE>SEASON . SUMMER))
 ((?<SELECT.A.SITE>MONTH . JUNE)
 (?<SELECT.A.SITE>SEASON . SPRING))
 ((?<SELECT.A.SITE>MONTH . MAY)
 (?<SELECT.A.SITE>SEASON . SPRING))
 ((?<SELECT.A.SITE>MONTH . APRIL)
 (?<SELECT.A.SITE>SEASON . SPRING))
 ((?<SELECT.A.SITE>MONTH . MARCH)
 (?<SELECT.A.SITE>SEASON . WINTER))
 ((?<SELECT.A.SITE>MONTH . FEBRUARY)
 (?<SELECT.A.SITE>SEASON . WINTER))
 ((?<SELECT.A.SITE>MONTH . JANUARY)
 (?<SELECT.A.SITE>SEASON . WINTER))

Secondary-constraints: #[Wff SIGMOD IS IN CLASS CONFERENCE]
 #[Wff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS
 UNIVERSITY]
 #[Wff ?<SELECT.A.SITE>SITE IS IN CLASS CITY]
 #[Wff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY
 IS ?<SELECT.A.SITE>SITE]
 #[Wff A MONTH OF SIGMOD IS
 ?<SELECT.A.SITE>MONTH]
 #[Wff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE IS
 ?<SELECT.A.SITE>SEASON]

Secondary-bindings: ((?<SELECT.A.SITE>SEASON . WINTER)
 (?<SELECT.A.SITE>MONTH . APRIL)
 (?<SELECT.A.SITE>SITE . DENVER)
 (?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

=====

(PARTITION 3:)

Primary-constraints: #[Wff A MONTH OF SIGMOD IS ?<SELECT.A.SITE>MONTH]

Primary-bindings: ((?<SELECT.A.SITE>MONTH . APRIL))

Secondary-constraints: #[Wff SIGMOD IS IN CLASS CONFERENCE]

#[Wff ?<SELECT.A.SITE>UNIVERSITY IS IN CLASS
UNIVERSITY]

#[Wff ?<SELECT.A.SITE>SITE IS IN CLASS CITY]

#[Wff A LOCALE OF ?<SELECT.A.SITE>UNIVERSITY IS
?<SELECT.A.SITE>SITE]

#[Wff A SEASON OF ?<SELECT.A.SITE>MONTH IS
?<SELECT.A.SITE>SEASON]

#[Wff A POPULAR-SEASONS OF ?<SELECT.A.SITE>SITE IS
?<SELECT.A.SITE>SEASON]

Secondary-bindings: ((?<SELECT.A.SITE>MONTH . DECEMBER)

(?<SELECT.A.SITE>SEASON . WINTER)

(?<SELECT.A.SITE>SITE . DENVER)

(?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

((?<SELECT.A.SITE>MONTH . MARCH)

(?<SELECT.A.SITE>SEASON . WINTER)

(?<SELECT.A.SITE>SITE . DENVER)

(?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

((?<SELECT.A.SITE>MONTH . FEBRUARY)

(?<SELECT.A.SITE>SEASON . WINTER)

(?<SELECT.A.SITE>SITE . DENVER)

(?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

((?<SELECT.A.SITE>MONTH . JANUARY)

(?<SELECT.A.SITE>SEASON . WINTER)

(?<SELECT.A.SITE>SITE . DENVER)

(?<SELECT.A.SITE>UNIVERSITY . UNIVERITY.OF.COLORADO))

=====

A P P E N D I X E

CONFERENCE DOMAIN DESCRIPTION

This appendix contains a sample description of the activities and objects pertinent to the domain of conference planning. This is the external description of the domain input to the POLYMER planner and used in the example scenarios described in Chapter 7.

```
;;; Conference procedures and objects

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                               ;;;
;;; CONFERENCE AGENT OBJECTS;;;
;;;                               ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

{OBJECT vice.president.for.conferences
 *subclass-of: human.agents
}

{OBJECT committee.members
 *subclass-of: human.agents
 *responsibilities:
 *current.meeting: VALUE-CLASS: technical.meeting
}

{OBJECT program.chair
 *subclass-of: committee.members
 *responsibilities: program
 *submissions.solicited:
 *submissions.received: VALUE-CLASS: submissions
}
```

```

    FACETS: (explicit.cardinality: multiple)
}

```

```

{OBJECT conference.chair
  *subclass-of: committee.members
  *responsibilities: operations
}

```

```

{OBJECT tutorials.chair
  *subclass-of: committee.members
  *responsibilities: tutorials
}

```

```

{OBJECT exhibits.chair
  *subclass-of: committee.members
  *responsibilities: exhibits
}

```

```

{OBJECT publicity.chair
  *subclass-of: committee.members
  *responsibilities: publicity
}

```

```

{OBJECT registration.chair
  *subclass-of: committee.members
  *responsibilities: registration
}

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; EDIT SUBMISSIONS AGENTS ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

{OBJECT reviewers
  *subclass-of: human.agents
  *areas.of.expertise: VALUE-CLASS:subject
  *papers.asked.to.review VALUE-CLASS: submissions
  *papers.currently.reviewing VALUE-CLASS: submissions
}

```

```

{OBJECT authors
  *subclass-of: human.agents
  *notified: VALUE-CLASS: (one.of yes no)
  *invited.to.submit: VALUE-CLASS: submissions
}

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

;;;          ;;;
;;; CONFERENCE OBJECTS ;;;
;;;          ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

{OBJECT date
  *numeric.value: value-class: number
}

```

```

{OBJECT technical.meeting
  *geographic.setting: VALUE-CLASS: city
  *start.date: VALUE-CLASS: date
  *finish.date: VALUE-CLASS: date
  *accomodations: VALUE-CLASS: hotel
  *number.of.participants: VALUE-CLASS: number
  *completion.status: VALUE-CLASS: (one.of planned in.progress
                                     held successful cancelled)
  *final.report: VALUE-CLASS: meeting.report
  *title: VALUE-CLASS: string
  *elected.conference.chair: VALUE-CLASS: conference.chair
  *papers.chosen.for.presentation:
  *schedule.status:
  *expenses.settled: VALUE-CLASS: (one.of yes no)
  *proceedings.sent.to.sponsor: VALUE-CLASS: (one.of yes no)
  *sponsors: VALUE-CLASS: professional.organization
  *proposal: VALUE-CLASS: TMRF
  *calendar.form: VALUE-CLASS: calendar.entry.form
  *issued.call.for.papers: VALUE-CLASS: call.for.papers
  *advance.program: VALUE-CLASS: advance.program
  *final.program: VALUE-CLASS: final.program
  *post.conference.paperwork:
  *program.committee:
  *conference.status:
  *registration.completed:
}

```

```

{OBJECT organization
}

```

```

{OBJECT professional.organization
  *subclass-of: organization
}

```

```

{OBJECT type.setting.company
  *subclass-of: organization
}

```

```
{OBJECT printing.company
 *subclass-of: organization
 }
```

```
{OBJECT conference
 *subclass-of: technical.meeting
 *number.of.participants: VALUE-CLASS: memberp (lambda (num) (> num 300))
 ;proceedings are usually published
 }
```

```
{OBJECT symposium
 *subclass-of: technical.meeting
 *number.of.participants: VALUE-CLASS: memberp (lambda (num)
 (and (> num 100) (< num 300)))
```

```
{OBJECT workshop
 *subclass-of: technical.meeting
 *number.of.participants: VALUE-CLASS: memberp (lambda (num) (< num 100))
 ;proceedings are usually NOT published
 }
```

```
{OBJECT TMRF
 ;technical.meeting.request.form
 *subclass.of: written.material
 *status: VALUE-CLASS: (one.of sent-for-approval approved rejected received)
 *meeting.name:
 *date.conflicts: VALUE-CLASS: (one.of none some)
 *completed: VALUE-CLASS: (one.of yes no)
 *proposed.meeting.start.date: VALUE-CLASS: date
 *proposed.meeting.finish.date: VALUE-CLASS: date
 *conference.assumptions.info: VALUE-CLASS: conference.assumptions.section
 *meeting: VALUE-CLASS: technical.meeting
 }
```

```
{OBJECT calendar.entry.form
 *meeting: VALUE-CLASS: technical.meeting
 *submitted: VALUE-CLASS: (one.of yes no)
 }
```

```
{OBJECT conference.assumptions.section
 }
```

```
{OBJECT meeting.report
 *subclass.of: written.material
 ;must be submitted within 120 days after the meeting to Chairman of
 ;Conferences Board and to the Conference Coordinator at ACM
```



```

*number.of.attendees:
*paper.quality.assessment:
*presentation.quality.assessment:
*problem.summary:
*financial.report:
*meeting: VALUE-CLASS: technical.meeting
}

```

```

{OBJECT call.for.papers
*subclass.of: written.material
*meeting: VALUE-CLASS: technical.meeting
*start.date: VALUE-CLASS: date
*finish.date: VALUE-CLASS: date
*location:
*sponsors: VALUE-CLASS: (one.of professional.organization)
*key.organizers:
*meeting.scope:
*meeting.purpose:
*types.of.papers.desired:
*due.date: VALUE-CLASS: date
*acceptance.notification.date: VALUE-CLASS: date
*where.to.send:
*status: VALUE-CLASS: (one.of created typeset printed distributed issued)
}

```

```

{OBJECT advance.program
*subclass.of: written.material
*meeting: VALUE-CLASS: technical.meeting
*start.date: VALUE-CLASS: date
*finish.date: VALUE-CLASS: date
*location:
*sponsors: VALUE-CLASS: (one.of professional.organization)
*key.organizers:
*meeting.scope:
*meeting.purpose:
*who.should.attend:
*preliminary.schedule:
    ;authors names and paper titles, dates and times, social activities
*hotel.reservation.form: hotel.reservation.form
    ;the actual form, not an instance
*conference.registration.form: conference.registration.form
*travel.information:
*status: VALUE-CLASS: (one.of typeset printed distributed)
}

```

```

{OBJECT final.program
*subclass.of: written.material
*meeting: VALUE-CLASS: technical.meeting
}

```

```

*welcome.from.chairman:
*keynote.speaker.biography:
*local.information: ;restaurants, etc.
*author.index:
*sponsors: VALUE-CLASS: (one.of professional.organization)
*schedule:
    ;authors names and paper titles, dates and times, social activities
*status: VALUE-CLASS: (one.of typeset printed distributed)
}

```

```

{OBJECT hotel.reservation.form
  *advance.program: VALUE-CLASS: advance.program
}

```

```

{OBJECT conference.registration.form
  *advance.program: VALUE-CLASS: advance.program
}

```

```

{OBJECT site
}

```

```

{OBJECT hotel
  *subclass-of: site
  *metropolitan.area: VALUE-CLASS: city
  *capacity:
  *availability:
}

```

```

{OBJECT city
  *subclass-of: site
  *climate: VALUE-CLASS: (one.of hot cold)
}

```

```

{OBJECT university
  *subclass-of: site
  *locale: VALUE-CLASS: city
}

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; EDIT SUBMISSIONS OBJECTS ;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

{OBJECT papers
  *subclass-of: written.material
  *title: VALUE-CLASS: string
  *authors: VALUE-CLASS: authors
}

```

```
*keywords: VALUE-CLASS: subject
}
```

```
{OBJECT submissions
```

```
*subclass-of: instantiable.entities
*paper: VALUE-CLASS: papers
*submitted.to: VALUE-CLASS: technical.meeting
*decision: VALUE-CLASS: possible.decisions.on.conference.papers
*refereed.by: VALUE-CLASS: program.chair
*considered.reviewers: VALUE-CLASS: reviewers
*invited.reviewers: VALUE-CLASS: reviewers
    INVERSE-SLOT: papers.asks.to.review reviewers
*declined.reviewers: VALUE-CLASS: reviewers
*reviewers: VALUE-CLASS: reviewers
    INVERSE-SLOT: papers.currently.reviewing reviewers
*reviews: VALUE-CLASS: reviews
*review.status: VALUE-CLASS: (one.of received-for-review reviewed)
*word.count:
*program.committee.decision.made:
}
```

```
{OBJECT reviews
```

```
*subclass-of: instantiable.entities
*submission: VALUE-CLASS: submissions
    INVERSE-SLOT: reviews submissions
*reviewer: VALUE-CLASS: reviewers
*recommendation: VALUE-CLASS: possible.decisions
}
```

```
{OBJECT requests
```

```
*subclass-of: messages, instantiable.entities
*type: "request"
*action:
*objects:
}
```

```
{OBJECT replies
```

```
*subclass-of: messages
*type: "reply"
*reply.to: VALUE-CLASS: messages
}
```

```
{OBJECT possible.decisions
}
```

```
{OBJECT possible.decisions.on.conference.papers
*subclass-of: possible.decisions
```

```
}  
  
{OBJECT possible.decisions.on.workshop.papers  
 *subclass-of: possible.decisions  
}  
  
{OBJECT subjects  
}  
  
{OBJECT artificial.intelligence  
 *subclass-of: subjects  
}  
  
{OBJECT natural.language  
 *subclass-of: artificial.intelligence  
}  
  
{OBJECT planning  
 *subclass-of: artificial.intelligence  
}  
  
{OBJECT time.representations  
 *subclass-of: artificial.intelligence  
}  
  
{OBJECT robot.planning  
 *subclass-of: planning  
}  
  
{OBJECT cooperative.planning  
 *subclass-of: planning  
}  
  
{OBJECT human.computer.issues  
 *subclass-of: subjects  
}  
  
{OBJECT user.interfaces  
 *subclass-of: human.computer.issues  
}  
  
{OBJECT user.modelling  
 *subclass-of: human.computer.issues  
}  
  
{OBJECT visual.languages  
 *subclass-of: human.computer.issues  
}
```

```
{OBJECT machine.learning
 *subclass-of: artificial.intelligence
 }
```

```
{OBJECT knowledge.acquisition
 *subclass-of: artificial.intelligence
 }
```

```
{OBJECT knowledge.representation
 *subclass-of: artificial.intelligence
 }
```

```
{OBJECT expertise.transfer
 *subclass-of: knowledge.acquisition
 }
```

```
{OBJECT computer.vision
 *subclass-of: artificial.intelligence
 }
```

```
{OBJECT explanation.based.learning
 *subclass-of: machine.learning
 }
```

```
{OBJECT similarity.based.learning
 *subclass-of: machine.learning
 }
```

```
{OBJECT connectionism
 *subclass-of: subjects
 }
```

```
{OBJECT office.automation
 *subclass-of: subjects
 }
```

```
{OBJECT document.retrieval
 *subclass-of: subjects
 }
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; CONFERENCE ACTIVITIES ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
{ACTIVITY stage.a.symposium
 *agents:?(chairperson conference.chair)
 *comment: "Top level activity to set up and hold a symposium."
```

```

*goal: completion.status(?meeting symposium), successful)
*input-goal-vars: ?meeting
*decomposition: (symposium-is-planned
                 completion.status(?meeting, planned))
                 (symposium-took-place completion.status(?meeting, held))
                 (symposium-postprocessing-done
                  post.conference.paperwork(?meeting, done))
*plan-rationale: enables(symposium-is-planned, symposium-took-place)
                 enables(symposium-took-place,
                          symposium-postprocessing-done)
*constraints: elected.conference.chair(?meeting, ?chairperson)
}

{ACTIVITY plan.a.symposium
 *agents: ?(chair conference.chair)
 *comment: "Plan a symposium."
 *goal: completion.status(?meeting, planned)
 *input-goal-vars: ?meeting
 *decomposition: (symposium-proposal-approved and(proposal(?meeting,
                                                       ?(proposal TMRF))
                                                       status(?proposal,
                                                       approved)))
                 (program-committee-chosen program.committee(?meeting, chosen))
                 (symposium-site-arranged and(geographic.setting(?meeting, ?site),
                                               accomodations(?meeting, ?hotel)))
                 (symposium-submissions-judged
                  papers.chosen.for.presentation(?meeting, done))
                 (symposium-schedule-completed schedule.status(?meeting, complete))
 *control: (BEFORE symposium-proposal-approved symposium-submissions-judged)
           (BEFORE program-committee-chosen symposium-submissions-judged)
           (BEFORE symposium-proposal-approved symposium-site-arranged)
           (BEFORE symposium-submissions-judged symposium-schedule-completed)
}

{ADVERTISEMENT plan.a.symposium.offline
 *agents: ?(chair conference.chair)
 *comment: "Plan a symposium -- do it all offline."
 *goal: completion.status(?meeting, planned)
 *input-goal-vars: ?meeting
}

{ADVERTISEMENT choose.program.committee
 *agents: ?(chair conference.chair)
 *goal: program.committee(?meeting, chosen)
 *input-goal-vars: ?meeting
}

```

```

{ACTIVITY conference.takes.place
 *agents:?(chair conference.chair)
 *comment:"Conduct the conference."
 *goal: completion.status(?meeting,held)
 *input-goal-vars: ?meeting
 *decomposition: (registration-conducted
                  registration.completed(?meeting, yes))
                  (conference-held conference.status(?meeting,in.progress))
 *control: (BEFORE registration-conducted conference-held)
}

{ADVERTISEMENT conduct.registration
 *agents:?(chair conference.chair)
 *goal: registration.completed(?meeting, yes)
 *input-goal-vars: ?meeting
}

{ADVERTISEMENT hold.conference
 *agents:?(chair conference.chair)
 *comment:"Conduct the conference."
 *goal: conference.status(?meeting,in.progress)
 *input-goal-vars: ?meeting
}

{ACTIVITY postprocess.a.symposium
 *agents:?(chairperson conference.chair)
 *comment:"Do all the wrap-up paperwork after a symposium has concluded."
 *goal: post.conference.paperwork(?meeting, done)
 *input-goal-vars: ?meeting
 *decomposition: (final-report-made final.report(?meeting,
                                                    ?(report meeting.report)))
                  (symposium-expenses-reimbursed
                   expenses.settled(?meeting, yes))
                  (copies-of-proceedings-to-sponsor
                   proceedings.sent.to.sponsor(?meeting, yes))
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;RELATED to GETTING APPROVAL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

{ACTIVITY get.approval.for.symposium
 *goal: and(proposal(?meeting, ?proposal), status(?proposal,approved))
 *decomposition: (calendar-entry-form-created
                  calendar.form(?meeting,

```

```

                                ?(calendar-entry-form,
                                  calendar.entry.form))
      (calendar-entry-form-submitted
        and(calendar.form(?meeting, ?calendar-entry-form),
          submitted(?calendar-entry-form, yes)))
      (tmrf-completed and(proposal(?meeting, ?(proposal, TMRF)),
        completed(?proposal, TMRF), yes)))
      (tmrf-sent status(?proposal, sent-for-approval))
      (receive-approved-tmrf status(?proposal, received))
*input-goal-vars: ?(meeting technical.meeting)
*control: (BEFORE calendar-entry-form-created calendar-entry-form-submitted)
          (BEFORE tmrf-completed tmrf-sent)
          (BEFORE tmrf-sent receive-approved-tmrf)
}

;;Creating calendar-entry-form and tmrf are tool actions

{ADVERTISEMENT submit.calendar.entry.form
  *comment: "Submit completed Calendar entry form as part of
    getting conference approval."
  *goal: and(calendar.form(?meeting, ?(calendar-entry-form,
    calendar.entry.form))
    submitted(?calendar-entry-form, yes))
  *input-goal-vars: ?(meeting technical.meeting)
    ?(calendar-entry-form calendar.entry.form)
}

{ACTIVITY complete.tmrf
  *comment: "Fill out a technical meeting request form to hold a conference."
  *goal: and(proposal(?meeting, ?(proposal, TMRF)),
    completed(?proposal, yes))
  *input-goal-vars: ?meeting
  *decomposition: (new-tmrf-created meeting(?proposal, TMRF), ?meeting)
    (date-overlaps-checked date.conflicts(?proposal, TMRF),
      none)
    (dates-selected and(meeting(?proposal, ?meeting),
      proposed.meeting.start.date
        (?proposal, ?start-day)
      proposed.meeting.finish.date
        (?proposal, ?finish-day)))
  *control: (BEFORE new-tmrf-created date-overlaps-checked)
    (BEFORE new-tmrf-created dates-selected)
}

{ADVERTISEMENT check.for.date.overlaps
  *comment: "Make sure there are no date overlaps when
    setting up a conference."

```



```

*goal: date.conflicts?(proposal,TMRF), none)
*input-goal-vars: ?proposal
}

{ADVERTISEMENT designate.symposium.dates
 *comment: "Designate the start and finish dates of a symposium."
 *agents:?(chair conference.chair)
 *goal: and(meeting(?proposal,?meeting),
            proposed.meeting.start.date?(proposal,TMRF),?start-day),
            proposed.meeting.finish.date?(proposal, TMRF), ?finish-day))
 *input-goal-vars: ?meeting, ?proposal
 *constraints: and(numeric.value(?start-day, ?start-date-val),
                  numeric.value(?finish-day,
                                ?finish-date-val),
                  <(start-date-val, ?finish-date-val),)
 *effects: add start.date(?meeting, ?start-day)
            add finish.date(?meeting, ?finish-day)
}

{ADVERTISEMENT send.proposal
 *comment: "Send a proposal out for approval."
 *goal: status?(proposal TMRF),sent-for-approval)
 *input-goal-vars: ?proposal
}

{ADVERTISEMENT receive.approved.proposal
 *comment: "Receive an approved proposal."
 *goal: status?(proposal TMRF),received)
 *input-goal-vars: ?proposal
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; ARRANGING THE SITE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

{ADVERTISEMENT designate.symposium.site
 *agents:?(chair conference.chair)
 *comment: "Designate the site for a symposium."
 *goal: and(geographic.setting(?meeting,?site),
            accomodations(?meeting,?hotel))
 *input-goal-vars: ?meeting
 *constraints: ;;capacity of hotel must be greater than symposium size,
               ;; and city of hotel must be
               ;; same as geographic.setting of meeting
               and(geographic.setting(?meeting, ?site),
                   metropolitan.area(?hotel, ?site))
               and(capacity(?hotel,?size),

```

```

                >( ?size,100))
*effects: add availability(?hotel, booked)
}

{ADVERTISEMENT designate.university.symposium.site
*agents: ?(chair conference.chair)
*comment: "Designate the site for a university symposium."
*goal: and(geographic.setting(?meeting,?site),
           accommodations(?meeting,?hotel))
*input-goal-vars: ?meeting
*constraints: ;;capacity of hotel must be greater than symposium size,
              ;;and city of hotel must be
              ;; same as geographic.setting of meeting
              and(geographic.setting(?meeting, ?site),
                  metropolitan.area(?hotel, ?site))
                  and(member(?university, university),
                      locale(?university, ?site))
                  and(capacity(?hotel,?size),
                      >( ?size,100))
*effects: add availability(?hotel, booked)
}

{ADVERTISEMENT report.on.conference
*comment: "Do the final report for a conference."
*goal: final.report(?meeting,?(final-report meeting.report))
*input-goal-vars: ?meeting
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                               ;;;
;;;  JOURNAL ACTIVITIES          ;;;
;;; for judging submissions      ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

{ACTIVITY invite.papers.for.presentation
*comment: "An alternate way to get papers -- invite from authors."
*agents: ?(program-chair program.chair)
*goal: papers.chosen.for.presentation?(meeting technical.meeting), done)
*input-goal-vars: ?meeting
*decomposition: (solicit-submissions submissions.received(?program-chair,
                                                           ?subms))
                 (submissions-invited decision(?submission, invited))
*control: (BEFORE solicit-submissions submissions-invited)
           (REPEAT submissions-invited for ?submission in ?subms)
*constraints: current.meeting(?program-chair, ?meeting)

```

}

```

{ACTIVITY judge.papers.for.presentation
 *comment: "Top level activity for conference presentation selection."
 *agents:?(program-chair program.chair)
 *goal: papers.chosen.for.presentation?(meeting technical.meeting), done)
 *input-goal-vars: ?meeting
 *decomposition (submissions-solicited
                 issued.call.for.papers(?meeting,?(call call.for.papers)))
                 (has-submissions
                  submissions.received(?program-chair, ?subms))
                 (submissions-reviewed review.status(?submission, reviewed))
                 (decisions-reached and(program.committee.decision.made
                                         (?submission t),
                                         decision(?submission, ?decision)))
                 (authors-notified and(paper(?submission, ?paper)
                                       authors(?paper ?author)
                                       notified(?author, yes)))
 *control: (BEFORE submissions-solicited has-submissions)
           (BEFORE has-submissions submissions-reviewed)
           (BEFORE submissions-reviewed decisions-reached)
           (BEFORE decisions-reached authors-notified)
           (REPEAT submissions-reviewed for ?submission in ?subms)
           (REPEAT decisions-reached for ?submission in ?subms)
           (REPEAT authors-notified for ?submission in ?subms)
 *constraints: current.meeting(?program-chair, ?meeting)
}

```

```

{ACTIVITY issue.call.for.papers
 *agents:?(program-chair program.chair)
 *goal: issued.call.for.papers(?meeting,?(call-for-papers call.for.papers))
 *input-goal-vars: ?meeting
 *decomposition: (call-for-papers-created
                 status?(call-for-papers call.for.papers),
                 created))
                 (call-for-papers-processed-and-issued
                  status?(call-for-papers call.for.papers) issued))
 *control: (BEFORE call-for-papers-created
           call-for-papers-processed-and-issued)
 *constraints: current.meeting(?program-chair ?meeting)
 *effects: add meeting(?call-for-papers, ?meeting)
}

```

```

{ACTIVITY process.call.for.papers
 *agents ?(program-chair program.chair)
 *goal: status?(call-for-papers call.for.papers), issued)
}

```

```

*input-goal-vars: ?call-for-papers
*decomposition: (call-for-papers-typeset status(?call-for-papers, typeset))
                 (call-for-papers-printed status(?call-for-papers, printed))
                 (call-for-papers-distributed status(?call-for-papers,
                                                       distributed))
*control: (BEFORE call-for-papers-typeset call-for-papers-printed)
           (BEFORE call-for-papers-printed call-for-papers-distributed)
}

```

```

{ACTIVITY invite.a.paper
 *agents: ?(program-chair program.chair)
 *goal: decision?(submission submissions), invited)
 *input-goal-vars: ?submission
 *decomposition: (invite-author invited.to.submit?(author authors),
                 ?submission)
}

```

```

{ADVERTISEMENT contract.out.call.for.papers
 *comment: "An optional service to handle all stages
           of processing the call for papers."
 *agents: ACM
 *goal: status(?call-for-papers, issued)
 *input-goal-vars: ?call-for-papers
 *effects: add status(?call-for-papers, typeset)
           add status(?call-for-papers, printed)
           add status(?call-for-papers, distributed)
}

```

```

{ADVERTISEMENT program.committee.decides.on.paper
 *comment: "Program committee decides whether to accept or reject the paper"
 *agents: ?program-chair
 *input-goal-vars: ?submission
 *goal: and(program.committee.decision.made(?submission, t),
           decision?(submission submissions), ?decision))
}

```

```

{ADVERTISEMENT paper.rejected.without.review
 *comment: "Program committee decides whether to accept or reject the paper"
 *agents: ?program-chair
 *preconditions: word.count(?submission, over.limit)
 *input-goal-vars: ?submission
 *goal: and(program.committee.decision.made(?submission, t),
           decision?(submission submissions), reject-paper))
}

```

```

{ADVERTISEMENT notify.author
 *comment: "Program chair notifies author of decision on submission."
}

```

```

*agents: ?program-chair
*input-goal-vars: ?author,?submission,?paper
*goal: and(paper(?submission, ?paper)
           authors(?paper ?author)
           notified(?author, yes))
}

{ADVERTISEMENT typeset.document
 *agents: ?(typesetter, type.setting.company)
 *goal: status(?document, typeset)
 *input-goal-vars: ?document
}

{ADVERTISEMENT print.document
 *agents: ?(printer printing.company)
 *goal: status(?document, printed)
 *input-goal-vars: ?document
}

{ADVERTISEMENT distribute.document
 *agents: ?program-chair
 *goal: status(?document, distributed)
 *input-goal-vars: ?document
}

{ADVERTISEMENT invite.a.paper.ad
 *agents: ?(program-chair program.chair)
 *goal: invited.to.submit(?author, ?submission)
 *input-goal-vars ?submission
 *constraints: paper(?submission, ?paper)
               authors(?paper, ?author)
}

{ADVERTISEMENT receive.submissions
 *agents: ?(program-chair program.chair)
 *goal: submissions.received(?program-chair, ?(subs submissions))
 *constraints: and(current.meeting(?program-chair ?conference)
                  submitted.to(?subs ?conference))
}

{ACTIVITY get.paper.reviewed
 *comment: "Select reviewers, send them the paper, and get their reviews."
 *agents: ?(program-chair program.chair)
 *goal: review.status?(?submission submissions), reviewed)
 *input-goal-vars: ?submission
 *decomposition: (reviewers-selected
                  reviewers(?submission, ?reviewers))
                  (reviews-completed)
}

```

```

        and(reviews(?submission, ?reviews),
            reviewers(?submission, ?reviewers)))
    *control: (BEFORE reviewers-selected, reviews-completed)
}

{ADVERTISEMENT get.reviewers
  *comment: "Designate reviewers for a given paper."
  ;; right now, only 1 is selected
  *agents: ?(program-chair program.chair)
  *goal: reviewers(?submission, ?(reviewers reviewers))
  *input-goal-vars: ?submission
  *constraints: paper(?submission, ?paper)
                keywords(?paper, ?subject-area),
                areas.of.expertise(?reviewers ?subject-area)
}

{ACTIVITY review.paper
  *comment: "Review the paper."
  *agents: ?(reviewer reviewers)
  *goal: and(reviews(?submission, ?reviews),
            reviewers(?submission, ?reviewer))
  *input-goal-vars: ?submission, ?reviewer
  *decomposition : (write-review and(submission(? (review reviews),
                                     ?submission),
                                     reviewer(?review, ?reviewer)))
                  (make-recommendation recommendation(?review,
                                                         ?decision))
  *control: (BEFORE write-review make-recommendation)
  *constraints: reviewers(?submission, ?reviewer)
}

{ADVERTISEMENT reviewer.make.recommendation
  *comment: "An individual reviewer recommends whether to
            accept or reject a submission."
  *goal: recommendation(? (review, reviews), ?(decision possible.decisions))
  *input-goal-vars: ?review
}

{ACTIVITY devise.symposium.schedule
  *comment: "Schedule paper sessions for symposium."
  *goal: schedule.status(?meeting, complete)
  *input-goal-vars: ?meeting
  *decomposition: (advance-program-issued
                  and(advance.program(?meeting,
                                       ?(advance-program advance.program)),
                     status(?advance-program, distributed)))
}

```

```

        (final-program-issued
         and(final.program(?meeting,
                          ?(final-program, final.program))
            status(?final-program, distributed)))
*control: (BEFORE advance-program-issued final-program-issued)
}

{ACTIVITY issue.advance.program
*agents: ?(program-chair program.chair)
*goal: and (advance.program(?meeting, ?(advance-program advance.program)),
            status(?advance-program, distributed))
*input-goal-vars: ?meeting
*decomposition: (advance-program-created advance.program(?meeting,
                                                         ?advance-program))
                 (advance-program-typeset status(?advance-program, typeset))
                 (advance-program-printed status(?advance-program, printed))
                 (advance-program-distributed status(?advance-program,
                                                         distributed))
*control: (BEFORE advance-program-created advance-program-typeset)
           (BEFORE advance-program-typeset advance-program-printed)
           (BEFORE advance-program-printed advance-program-distributed)
}

{ACTIVITY issue.final.program
*agents: ?(program-chair program.chair)
*goal: and (final.program(?meeting, ?(final-program final.program)),
            status(?final-program, distributed))
*input-goal-vars: ?meeting
*decomposition: (final-program-created final.program(?meeting,
                                                         ?final-program))
                 (final-program-typeset status(?final-program, typeset))
                 (final-program-printed status(?final-program, printed))
                 (final-program-distributed status(?final-program,
                                                         distributed))
*control: (BEFORE final-program-created final-program-typeset)
           (BEFORE final-program-typeset final-program-printed)
           (BEFORE final-program-printed final-program-distributed)
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Some ADS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

{ADVERTISEMENT settle.expenses
*comment: "Reimburse sponsors for loans or allocate surplus income."
*goal: expenses.settled(?(meeting technical.meeting), yes)
}

```

```

}

{ADVERTISEMENT send.proceedings.to.sponsor
 *comment: "Send 5 copies of proceedings to sponsor."
 *goal: proceedings.sent.to.sponsor(?meeting, yes)
 ;*effects: there should be 5 less proceedings now
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Some specifics
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

{INSTANCE accept-paper possible.decisions.on.conference.papers
}

{INSTANCE reject-paper possible.decisions.on.conference.papers
}

{INSTANCE accept-paper-with-modifications
    possible.decisions.on.conference.papers
}

{INSTANCE invited possible.decisions.on.workshop.papers
}

{INSTANCE john.smith vice.president.for.conferences
 *name: "John Smith"
}

{INSTANCE charles.jones vice.president.for.conferences
 *name: "Charles Jones"
}

{INSTANCE jack.jones conference.chair
 *name: "Jack Jones"
}

{INSTANCE Gerald.Salton conference.chair
 *name: "Gerald Salton"
}

{INSTANCE Marcel.Schoppers conference.chair
 *name: "Marcel Schoppers"
}

{INSTANCE gerhard.fischer program.chair
 *name: "Gerhard Fischer"
 *current.meeting: knowledge.based.human.computer.communication
}

```



```
}

{INSTANCE knowledge.based.human.computer.communication.symposium
 *elected.conference.chair: jack.jones
}

{INSTANCE planning.in.uncertain.environments.symposium
 *elected.conference.chair: Marcel.Schoppers
}

{INSTANCE text.based.intelligent.systems.symposium
 *elected.conference.chair: Gerald.Salton
}

{INSTANCE ACM.professional.organization
}

{INSTANCE IEEE.professional.organization
}

{INSTANCE AAAI.professional.organization
}

{INSTANCE Joes.printers.printing.company
}

{INSTANCE acm.printers.printing.company
}

{INSTANCE joes.typesetters.type.setting.company
}

{INSTANCE james.hendler.program.chair
 *name: "James Hendler"
 *current.meeting: planning.in.uncertain.environments
}

{INSTANCE paul.jacobs.program.chair
 *name: "Paul Jacobs"
 *current.meeting: text.based.intelligent.systems
}

{INSTANCE clayton.lewis.reviewers
 *name: "Clayton Lewis"
 *areas.of.expertise: user.modelling, user.interfaces,
                    isual.languages
}
```

```
{INSTANCE elaine.rich reviewers
  *name: "Elaine Rich"
  *areas.of.expertise: user.modelling
}

{INSTANCE thomas.dean reviewers
  *name: "Thomas Dean"
  *areas.of.expertise: time.representations, planning,
                      knowledge.representation
}

{INSTANCE alberto.segre reviewers
  *name: "Alberto Segre"
  *areas.of.expertise: robot.planning, explanation.based.learning
}

{INSTANCE Ryszard.S.Michalski reviewers
  *name: "Michalski"
  *areas.of.expertise: machine.learning
}

{INSTANCE Jaime.G.Carbonell reviewers
  *name: "Carbonell"
  *areas.of.expertise: machine.learning
}

{INSTANCE Tom.M.Mitchell reviewers
  *name: "Mitchell"
  *areas.of.expertise: machine.learning
}

{INSTANCE bruce.croft reviewers
  *name: "Bruce Croft"
  *areas.of.expertise: office.automation, document.retrieval,
                      cooperative.planning
}

{INSTANCE david.waltz reviewers
  *name: "David Waltz"
  *areas.of.expertise: natural.language, connectionism
}

{INSTANCE tom.gruber reviewers
  *name: "Thomas Gruber"
  *areas.of.expertise: knowledge.acquisition, knowledge.representation
}
```

```
{INSTANCE paul.utgoff reviewers
  *name: "Paul Utgoff"
  *areas.of.expertise: similarity.based.learning
}
```

```
{INSTANCE larry.lefkowitz reviewers
  *name: "Lawrence S. Lefkowitz"
  *areas.of.expertise: expertise.transfer
}
```

```
{INSTANCE oddmar.sandvik authors
}
```

```
{INSTANCE dyrk.mahling authors
}
```

```
{INSTANCE carol.broverman authors
}
```

```
{INSTANCE carla.brodley authors
}
```

```
{INSTANCE david.lewis authors
}
```

```
{INSTANCE Amherst city
  *state: Massachusetts
}
```

```
{INSTANCE Boston city
  *state: Massachusetts
}
```

```
{INSTANCE Chicago city
  *state: Illinois
}
```

```
{INSTANCE Honolulu city
  *state: Hawaii
}
```

```
{INSTANCE Pittsfield city
  *state: Massachusetts
}
```

```
{INSTANCE York city
  *state: Maine
}
```

```
{INSTANCE university.of.massachusetts university
  *locale: Amherst
}

{INSTANCE massachusetts.institute.of.technology university
  *locale: Boston
}

{INSTANCE university.of.chicago university
  *locale: Chicago
}

{INSTANCE pittsfield.hilton hotel
  *name: "Hilton Inn"
  *metropolitan.area: Pittsfield
  *capacity: 500
}

{INSTANCE travellers.inn hotel
  *name: "Travellers Inn"
  *metropolitan.area: Boston
  *capacity: 200
}

{INSTANCE sleepy.time.inn hotel
  *name: "Sleepy Time Inn"
  *metropolitan.area: Boston
  *capacity: 150
}

{INSTANCE cambridge-bed-and-breakfast hotel
  *name: "Cambridge Bed and Breakfast"
  *metropolitan.area: Boston
  *capacity: 40
}

{INSTANCE ramada.boston hotel
  *name: "Ramada"
  *metropolitan.area: Boston
  *capacity: 500
}

{INSTANCE ramada.chicago hotel
  *name: "Ramada"
  *metropolitan.area: chicago
  *capacity: 500
}
```

```
{INSTANCE hyatt.regency hotel
  *name: "Hyatt Regency"
  *metropolitan.area: Boston
  *capacity: 1000
}

{INSTANCE campus.center hotel
  *name: "Campus Center Hotel"
  *metropolitan.area: Amherst
  *capacity: 1000
}

{INSTANCE skipper.inn hotel
  *name: "The Skipper Inn"
  *metropolitan.area: York
  *capacity: 100
}

{INSTANCE january.1 date
  *numeric.value: 1
}

{INSTANCE january.4 date
  *numeric.value: 4
}

{INSTANCE february.1 date
  *numeric.value: 32
}

{INSTANCE february.4 date
  *numeric.value: 36
}

{INSTANCE june.1 date
  *numeric.value: 152
}

{INSTANCE june.4 date
  *numeric.value: 156
}

{INSTANCE paper.001 papers
  *title: "Multivariate decision trees for supervised learning"
  *authors: carla.brodley
  *keywords: machine.learning
```

```
}

{INSTANCE paper.002 papers
 *title: "DACRON: A graphical interface for knowledge acquisition"
 *authors: dyrk.mahling
 *keywords: knowledge.acquisition, knowledge.representation,
           visual.languages
}

{INSTANCE paper.003 papers
 *title: "Plan Execution Using Human Agents"
 *authors: carol.broverman
 *keywords: cooperative.planning, planning, explanation.based.learning,
           knowledge.representation
}

{INSTANCE paper.004 papers
 *title: "ODDCRON: An Animated Intelligent Interface"
 *authors: oddmar.sandvik
 *keywords: knowledge.acquisition, visual.languages
}

{INSTANCE paper.005 papers
 *title: "Document Queries Using Natural Language"
 *authors: david.lewis
 *keywords: document.retrieval, natural.language
}

{INSTANCE submission.001 submissions
 *paper: paper.001
 *submitted.to: planning.in.uncertain.environments
 *word.count: within.limit
}

{INSTANCE submission.002 submissions
 *paper: paper.002
 *submitted.to: planning.in.uncertain.environments
 *word.count: within.limit
}

{INSTANCE submission.003 submissions
 *paper: paper.003
 *submitted.to: planning.in.uncertain.environments
 *word.count: within.limit
}

{INSTANCE submission.004 submissions
 *paper: paper.004
```

```

*submitted.to: planning.in.uncertain.environments
*word.count: within.limit
}

{INSTANCE submission.005 submissions
 *paper: paper.005
 *submitted.to: text.based.intelligent.systems
 *word.count: within.limit
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;   TOOL ACTIONS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

{TOOL create.form.objects
}

{ACTION create.form.for.meeting
 *comment: "Create a new technical meeting request form to start
           planning a meeting."
 *goal: meeting(?proposal TMRF), ?meeting)
 *calling.arglist: (?proposal ?meeting)
 *input-goal-vars: ?meeting
 *code: (lambda (new-tmrf-var the-meeting)
         (let ((tmrf-instance (create-dynamic-instance 'tmrf)))
             (action-put-value tmrf-instance 'meeting the-meeting)
             (action-put-value the-meeting 'proposal tmrf-instance)
             (set-action-binding new-tmrf-var tmrf-instance)))
 *duration: "1 minutes"
}

{ACTION create.calendar.entry.form
 *comment: "Create a calendar entry form for planning a meeting."
 *goal: calendar.form(?meeting technical.meeting), ?calendar-form)
 *calling.arglist: (?calendar-form ?meeting)
 *input-goal-vars: ?meeting
 *code: (lambda (new-calendar-form-var the-meeting)
         (let ((calendar-instance (create-dynamic-instance 'calendar.entry.form)))
             (action-put-value calendar-instance 'meeting the-meeting)
             (action-put-value the-meeting 'calendar.form calendar-instance)
             (set-action-binding new-calendar-form-var calendar-instance)))
 *duration: "1 minutes"
}

{ACTION create.call.for.papers
 *comment: "Create a call for papers for planning a meeting."

```

```

*goal: status?(call-for-papers, call.for.papers) created)
*calling.arglist: (?call-for-papers)
*code: (lambda (new-call-for-papers-var)
  (let ((call-for-papers-instance (create-dynamic-instance
                                   'call.for.papers)))
    (set-action-binding new-call-for-papers-var call-for-papers-instance)))
*duration: "1 minutes"
}

{ACTION create.advance.program
 *comment: "Create a call for papers for planning a meeting."
 *goal: advance.program?(meeting technical.meeting), ?advance-program)
 *calling.arglist: (?advance-program ?meeting)
 *input-goal-vars: ?meeting
 *code: (lambda (new-advance-program-var the-meeting)
  (let ((advance-program-instance (create-dynamic-instance
                                   'advance.program)))
    (action-put-value advance-program-instance 'meeting the-meeting)
    (action-put-value the-meeting 'advance.program advance-program-instance)
    (set-action-binding new-advance-program-var advance-program-instance)))
 *duration: "1 minutes"
}

{ACTION create.final.program
 *comment: "Create a call for papers for planning a meeting."
 *goal: final.program?(meeting technical.meeting), ?final-program)
 *calling.arglist: (?final-program ?meeting)
 *input-goal-vars: ?meeting
 *code: (lambda (new-final-program-var the-meeting)
  (let ((final-program-instance (create-dynamic-instance 'final.program)))
    (action-put-value final-program-instance 'meeting the-meeting)
    (action-put-value the-meeting 'final.program final-program-instance)
    (set-action-binding new-final-program-var final-program-instance)))
 *duration: "1 minutes"
}

{ACTION create.hotel.reservation.form
 *comment: "Create a call for papers for planning a meeting."
 *goal: hotel.reservation.form?(advance-program advance.program),
      ?hotel-reservation-form)
 *calling.arglist: (?hotel-reservation-form ?advance-program)
 *input-goal-vars: ?advance-program
 *code: (lambda (new-hotel-reservation-form-var the-advance-program)
  (let ((hotel-reservation-form-instance (create-dynamic-instance
                                           'hotel.reservation.form)))
    (action-put-value hotel-reservation-form-instance 'advance.program
                      the-advance-program)

```



```

      (action-put-value the-advance-program 'hotel.reservation.form
                        hotel-reservation-form-instance)
      (set-action-binding new-hotel-reservation-form-var
                          hotel-reservation-form-instance)))
    *duration: "1 minutes"
  }

{ACTION create.conference.registration.form
 *comment: "Create a call for papers for planning a meeting."
 *goal: conference.registration.form(?advance-program advance.program),
      conference.registration-form
 *calling.arglist: (?conference-registration-form ?advance-program)
 *input-goal-vars: ?advance-program
 *code: (lambda (new-conference-registration-form-var the-advance-program)
         (let ((conference-registration-form-instance
                (create-dynamic-instance 'conference.registration.form)))
             (action-put-value conference-registration-form-instance
                                'advance.program the-advance-program)
             (action-put-value the-advance-program 'conference.registration.form
                                conference-registration-form-instance)
             (set-action-binding new-conference-registration-form-var
                                conference-registration-form-instance)))
    *duration: "1 minutes"
  }

{TOOL reviewing.tools
  }

{ACTION create.and.compose.a.review
 *comment: "Create and compose a review for a submission."
 *goal : and(submission(?review reviews),?(submission submissions)),
          reviewer(?review,?(reviewer reviewers)))
 *input-goal-vars: ?submission, ?reviewer
 *calling.arglist: (?review ?reviewer ?submission)
 *code: (lambda (new-review-form-var the-reviewer the-submission)
         (let ((review-form-instance (create-dynamic-instance 'reviews)))
             (action-put-value review-form-instance 'reviewer the-reviewer)
             (action-put-value review-form-instance 'submission the-submission)
             (set-action-binding new-review-form-var review-form-instance)))
    *duration: "2 weeks"
  }

```

A P P E N D I X F

SURVEY: EXPERIENCE IN ORGANIZING AND ATTENDING CONFERENCES

Part 1

1. What is your status at COINS (check where applies):

Student	
Faculty	
Post-doc	
Staff	
Other(specify: _____)	

2. Have you attended academic conferences/workshops/symposia? If so, check and note approximately how many have you attended in the following size categories:

<i>Size</i>	<i>yes</i>	<i>no</i>	<i>approx. how many times</i>
< 50 participants?			
50-100 participants?			
100-300 participants?			
> 300 participants?			

3. Have you ever been on the program committee for an academic conference?

<i>yes</i>	<i>no</i>

4. Have you ever been a reviewer for a conference?

<i>yes</i>	<i>no</i>

5. Have you performed any of the following functions in the organization of an academic conference? If so, check the appropriate function and note the

approximate size of the corresponding conference (using the size categories in Question 2).

<i>Function</i>	<i>check here if performed</i>	<i>conference size</i>
Conference chair		
Program chair		
Finance chair		
Tutorials chair		
Exhibits chair		
Publicity chair		
Registration chair		
Local arrangements chair		
Other function (specify: _____)		

Part 2

Suppose you are to organize a conference. Attached to this questionnaire is a partial description of how a conference is typically staged¹. The task is described in terms of its breakdown into subtasks (refer to both the outline and graphical descriptions of the task), as well as the timing dependencies between subtasks (refer to the graphical description). Take a few minutes to review the attached information about conference planning before answering the questions which follow. Use extra paper if necessary.

Task Breakdown

Answer the following questions. Where your answer is based on actual experience, please note with an asterisk (*).

1. Do you think that any orderings specified in the plan are inaccurate? If so, also answer the following:
 - (a) Which orderings, if any, are too rigid? Explain, indicating changes you would make to the diagram (if possible).
 - (b) Which orderings, if any, are missing? Explain.
 - (c) Which orderings, if any, are simply wrong? Explain.

¹This description is based on ACM guidelines.

2. Do you think it's possible that some later part of the plan could precede a part of the plan which is supposed to occur earlier? If so, under what circumstances would this be permitted?
3. Can you think of a case in which parts of the plan can be skipped? If so, what would allow this to happen? Describe.
4. Are there cases in which you might want to repeat parts of this plan?
5. Do you think there are important steps missing from this plan?
6. Are there any other ways in which you think the ordering constraints in this plan are unrealistic?
7. If you were to have devised this plan, would you have broken the subtasks down differently? How?

Constraints

Assume the following constraints have been defined for the task of planning a conference: For each constraint, answer the following questions:

1. Do you believe that the constraint is accurate? Are there cases in which you think it is not appropriate? If so:
 - (a) State whether the constraint should be relaxed or tightened.
 - (b) State the situations in which it might be inappropriate (always? only sometimes? when?).
 - (c) If possible, suggest a new or modified constraint (that also covers the situations you have just identified).
1. A reviewer's decision whether to accept a paper or not is one of: "strong-accept," "accept," "reject," "strong-reject."
2. The reviewer designated for a submission must be 1) in the same field as one of the keywords listed on the paper, AND 2) a member of the program committee.
3. The sleeping capacity of the hosting hotel chosen when designating the site of the conference should be 1) larger than the anticipated number of participants, AND 2) smaller than twice the number of anticipated participants.
4. The location in which the conference is held should be a major city, because that is the only place where meeting halls are large enough.

5. If the conference has less than 50 attendees, it is considered to be a workshop, and proceedings are not published.
6. When the host hotel is chosen, a block of rooms is reserved and cannot be reserved by anyone but the conference accommodations person (the local arrangements chair).
7. The meeting report must be submitted within 120 days after the conference is concluded.
8. The same individual should not be asked to review more than 5 papers.
9. Students should not be selected as reviewers of submissions.
10. Submissions should be received either by U.S. Mail or via Federal Express.
11. An approved TMRF (technical meeting request form) must be received before any other conference arrangements can take place.
12. All submissions must be received before any reviewers are chosen.
13. The site of the conference must be selected before the call for papers can be issued.

Agents

Referring to the outline form of the task, notice that the agent (e.g., conference chair, program chair, etc.) responsible for overseeing or accomplishing each subtask or block of subtasks is indicated. Are there cases in which you think a subtask may be performed by an agent other than the indicated responsible agent? Describe.

Miscellaneous

Can you think of any other unusual occurrences or changes in plan that occurred during a conference you organized or attended (that wasn't addressed by this questionnaire)? You may consider any subtasks that may be part of staging a conference, even if they are not described in detail in the attached description (for example, during conference registration, budget balancing, etc.).

Thank you very much for your help.

Work breakdown description: Outline form

The purpose of this task is to successfully organize and hold an academic conference. The outline organization reflects the subtask structure. For example, staging

a conference can be broken down into three major subtasks: planning the conference, holding the conference, and completing post-conference paperwork. As part of planning the conference, approval must be obtained, a site must be selected, etc.

1. The conference is planned (conference chair)
 - (a) Approval for conference obtained from sponsor
 - i. Technical meeting request form (TMRF) completed (conference chair)
 - A. Start filling out a new TMRF (conference chair)
 - B. Date conflicts with other conferences checked (conference chair)
 - C. Conference start and end dates selected (conference chair)
 - ii. TMRF sent to sponsor (conference chair)
 - iii. Approved TMRF received (conference chair)
 - (b) Establish site for conference (conference chair)
 - (c) Papers to be presented are selected (program chair)
 - i. Submissions solicited (program chair)
 - A. Call for papers prepared (program chair)
 - B. Call for papers typeset (program chair)
 - C. Call for papers printed (program chair)
 - D. Call for papers distributed (program chair)
 - ii. Submissions received (program chair)
 - iii. Submissions refereed (for each received submission) (program chair)
 - A. A reviewer is designated (program chair)
 - B. Select reviewer for submission (program chair)
 - C. Get reviewer's agreement to review (program chair)
 - D. Receive back paper with reviewer's decision (program chair)
 - E. Program committee meets to make decisions (program chair)
 - F. Authors are notified of decisions (program chair)
 - G. Final versions of papers are received (program chair)
 - (d) Proceedings of conference are ready
 - i. Camera-ready copy edited for errors (program chair)
 - ii. Typesetting and printing done by ACM service (ACM service)
 - iii. Proceedings delivered to site of conference (ACM service)
 - (e) Schedule of conference is completed (publicity chair)
 - i. Advance program issued (publicity chair)
 - ii. Final program issued (publicity chair)

2. Conference is held

- (a) Registration is conducted (registration chair)**
- (b) Technical sessions are held**
- (c) The conference banquet is held**

3. Post-conference paperwork is submitted (conference chair)

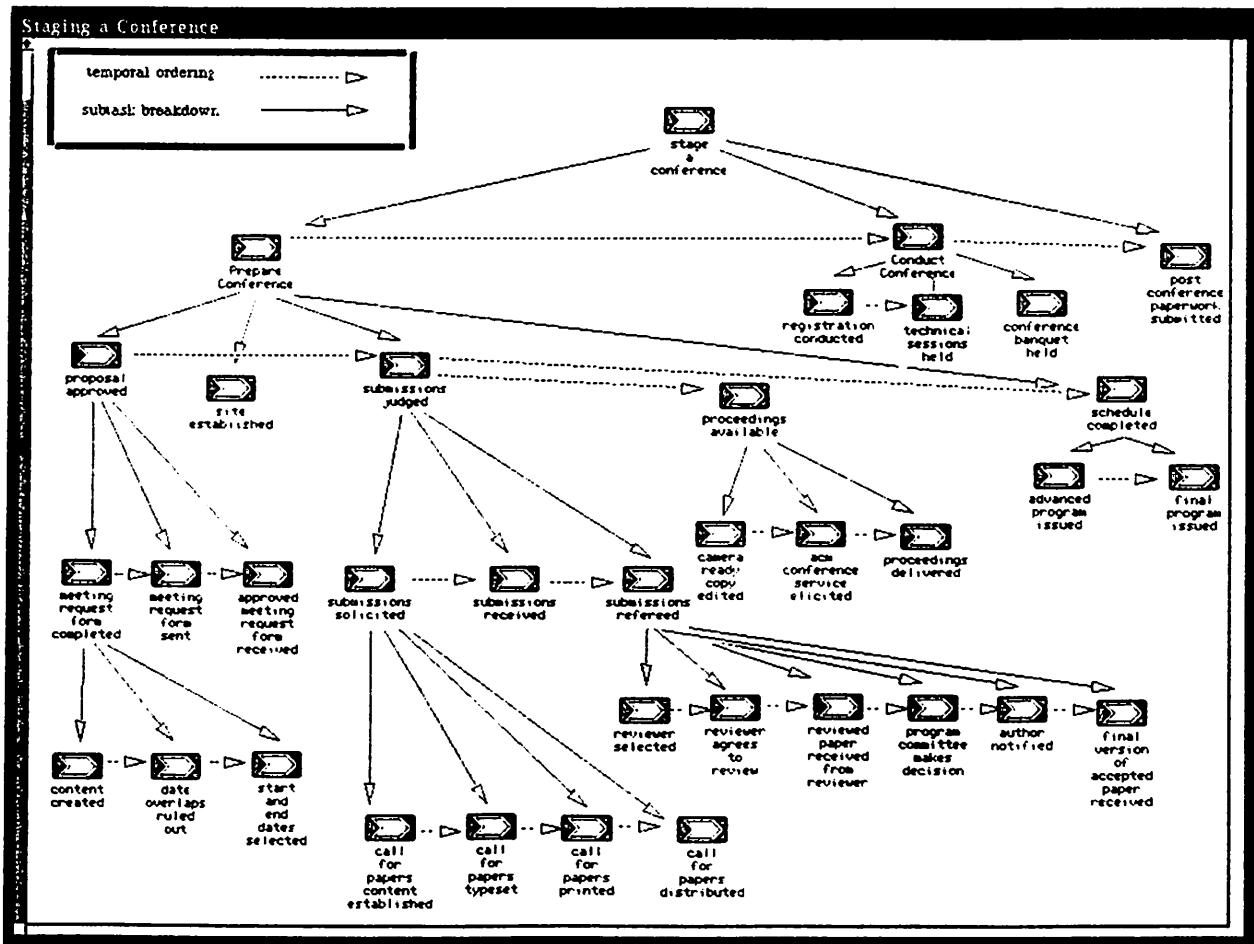


Figure 91: The work breakdown structure for staging a conference

BIBLIOGRAPHY

- [1] Agre, P. and D. Chapman. What are plans for? A.I. Memo 1050, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, September 1988.
- [2] Alterman, R. Issues in adaptive planning. Tech Report UCB/CSD 87-304, Computer Science Division, University of California, Berkeley, July 1985.
- [3] Alterman, R. An adaptive planner. In *Proceedings of The Fifth National Conference on Artificial Intelligence*, pages 65-69, 1986.
- [4] Ambros-Ingerson, J.A. and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 83-88, 1988.
- [5] Anderson, John S. and Arthur M. Farley. Plan abstraction based on operation generalization. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 100-104, 1988.
- [6] Bainbridge. Ironies of automation. In Leplat Rasmussen, Duncan, editor, *New Technology and Human Error*. John Wiley and Sons, New York, 1987.
- [7] Beetz, M. and L. Lefkowitz. Reasoning about justified events: A unified treatment of temporal projection, planning rationale and domain constraints. CSL Technical Report 89-6, University of Massachusetts, 1989.
- [8] Borgida, A. and K.E. Williamson. Accomodating exceptions in databases, and refining the schema by learning from them. In *Proceedings of the Very Large Data Base Conference 11*, pages 72-81, 1985.
- [9] Borgida, A. Language features for flexible handling of exceptions in information systems. *ACM Transactions on Database Systems*, 10(4):565-603, December 1985.

- [10] Brooks, R.A. A robust layered control system for a mobile robot. A.I. Memo 864, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1985.
- [11] Broverman, C. A. and W.B. Croft. A knowledge-based approach to data management for intelligent user interfaces. In *Proceedings of the Very Large Data Base Conference 11*, pages 96-104, 1985.
- [12] Carver, Norman, Victor Lesser, and Daniel McCue. Focusing in plan recognition. In *Proceedings of AAAI-84*, pages 42-48, 1984.
- [13] Chapman, D. Planning for conjunctive goals. *Artificial Intelligence*, 32:222-377, 1987.
- [14] Chapman, D. and P. Agre. Abstract reasoning as emergent from concrete activity. In *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, Los Altos, CA, 1987. Morgan-Kaufman.
- [15] Collins, G. Plan invention and plan transformation. In Michalski Mitchell, Carbonell, editor, *Machine Learning: A Guide to Current Research*, pages 43-45. Morgan Kaufmann, 1986.
- [16] Collins, G. Plan adaptation: A transformational approach. In K. Hammond, editor, *Proceedings of the DARPA Case Based Reasoning Workshop*, pages 90-93, 1989.
- [17] Croft, W. B. and Lawrence S. Lefkowitz. Task support in an office system. *ACM Transactions on Office Information Systems*, July 1984.
- [18] Darpa Santa Cruz Workshop On Planning. *AI Magazine*, pages 115-130, Summer 1988.
- [19] DeJong, G.F. An approach to learning from observation. In Michalski Mitchell, Carbonell, editor, *Machine Learning: Volume II.*, pages 571-590. Morgan Kaufmann, 1986.
- [20] DeJong, G.F. Generalizations based on explanations. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 67-70, Vancouver, B.C., Canada, August 1981.
- [21] DeJong, G.F. and R. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1, 1986.
- [22] Doyle, J. A truth maintenance system. *Artificial Intelligence*, 12:231-272, 1979.

- [23] Doyle, R. J., D.J. Atkinson, and R.S. Doshi. Generating perception requests and expectaions to verify the execution of plans. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 81-87, Philadelphia, PA, 1986.
- [24] Fikes, R.E. and N.J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189-208, 1971.
- [25] Fikes, R.E. A commitment-based framework for describing informal cooperative work. *Cognitive Science*, 6:331-347, 1982.
- [26] Fox, Mark S. Constraint-directed search: A case study of job-shop scheduling. CMU-RI-TR 83-22, Carnegie-Mellon University, December 1983. Ph.D. Thesis.
- [27] Hammond, K.J., editor. *Proceedings of the DARPA Case Based Reasoning Workshop*, Pensacola Beach, Florida, 1989.
- [28] Hammond, K.J. *Case-Based Planning: an Integrated Theory of Planning, Learning and Memory*. PhD thesis, Yale University, New Haven, CT, 1986.
- [29] Hammond K.J. Chef: A model of case-based planning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 267-271, Philadelphia, PA, August 1986.
- [30] Hammond, K.J. Learning to anticipate and avoid planning problems through the explanation of failure. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 556-560, Philadelphia, PA, 1986.
- [31] Hammond, K.J. Explaining and repairing plans that fail. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 109-115, Milan, Italy, August 1987.
- [32] Hayes, P.J. A representation for robot plans. In *Proceedings IJCAI-75*, pages 181-188, 1975.
- [33] Hollnagel, E. Action not as planned: The phenotype and genotype of erroneous actions. Technical report, Computer Resources International, Copenhagen, Denmark, 1987. DRAFT.

- [34] Huff, Karen E. *Plan-Based Intelligent Assistance: An Approach to Supporting the Software Development Process*. PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1989.
- [35] Huff, Karen E. and Victor E. Lesser. Meta-plans that dynamically transform plans. COINS Technical Report 87-10, COINS Department, University of Massachusetts, November 1987.
- [36] Kolodner, J., editor. *Proceedings of the DARPA Case Based Reasoning Workshop*, Clearwater Beach, Florida, 1988.
- [37] Lefkowitz, Lawrence S. *Knowledge Acquisition through Anticipation of Modifications*. PhD thesis, University of Massachusetts, Amherst, MA, 1987.
- [38] Lefkowitz, L.S. and W.B. Croft. An interim report on the polymer planning system. Coins technical report, University of Massachusetts, Amherst, Ma., 1987.
- [39] Lefkowitz, L.S. and W.B. Croft. Planning and execution of tasks in cooperative work environments. In *Proceedings of the 5th IEEE Conference on Artificial Intelligence Applications*, 1989.
- [40] Lefkowitz, L.S. and W.B. Croft. Interactive planning for knowledge based task management, April 1990. To be submitted to *Journal of Data and Knowledge Engineering*.
- [41] Lewis, C. and D.A. Norman. Designing for error. In Donald A. Norman, editor, *User-Centered System Design*, pages 411-432. Lawrence Erlbaum Associates, 1986.
- [42] Mahling, D.E. *A Visual Language for Knowledge Acquisition, Display and Animation by Domain Experts and Novices*. PhD thesis, University of Massachusetts, February 1990.
- [43] Mahling, D.E. and W. B. Croft. Relating human knowledge to plans. *International Journal of Man Machine Studies*, 31:61 - 97, 1989.
- [44] McDermott, D.V. Planning and acting. *Cognitive Science*, 2, 1978.
- [45] Nilsson, N.J. *Principles of Artificial Intelligence*. Tioga, Palo Alto, California, 1980.
- [46] Norman, Donald A. Categorization of action slips. *Psychological Review*, 88(1):1-15, January 1981.

- [47] Pednault, E.P. Formulating multiagent, dynamic problems in the classical planning framework. In *Proceedings of the 1986 Workshop on Reasoning About Actions and Plans*, pages 47-82, Timberline, Oregon, 1986.
- [48] Pietras, Christine M. and Oddmar A. Sandvik. Review of project management tasks and tools. CSL Technical Report CSL-90-1, Collaborative Systems Laboratory, COINS, University of Massachusetts, 1990.
- [49] Rajamoney, S., G. DeJong, and B. Faltings. Towards a model of conceptual knowledge acquisition through directed experimentation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 688-690, Los Angeles, CA, 1985.
- [50] Rasmussen, J. What can be learned from human error reports? In K. Duncan, M. Gruneberg, and D. Wallis, editors, *Changes in Working Life*. John Wiley, London, 1980.
- [51] Rasmussen, J. The definition of human error and a taxonomy for technical system design. In Leplat, Rasmussen, Duncan, editors, *New Technology and Human Error*. John Wiley and Sons, New York, 1987.
- [52] Leplat, Rasmussen, Duncan, editors. *New Technology and Human Error*. John Wiley and Sons, New York, 1987.
- [53] Reason, J. Actions not as planned: The price of automatization. In G. Underwood and R. Stevens, editors, *Aspects of Consciousness*, volume 1. Academic Press, London, 1979.
- [54] Reason, J. and K. Mycielska. *Absent-Minded? The Psychology of Mental Lapses and Everyday Errors*. Prentice-Hall, Inc., 1982.
- [55] Reason, J. A framework for classifying errors. In Leplat Rasmussen, Duncan, editor, *New Technology and Human Error*. John Wiley and Sons, New York, 1987.
- [56] Sacerdoti, E.D. *A Structure for Plans and Behavior*. Elsevier North-Holland, Inc., New York, NY, 1977.
- [57] Sathi, A., T.E. Morton, and S.F. Roth. Callisto: An intelligent project management system. *AI Magazine*, 7(5):34-52, Winter 1986.
- [58] Schoppers, M.J. Universal plans for reactive robots in unpredictable environments. In *Proceedings of IJCAI-87*, pages 1039-1046, Milan, Italy, 1987.

- [59] Simmons, R. A theory of debugging plans and interpretations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 94-99, 1988.
- [60] Simmons, R. and R. Davis. Generate, test, and debug: Combining associational rules and causal models. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1071-1078, Milan, Italy, August 1987.
- [61] Stefik, M. Planning and metaplanning. In *Readings in Artificial Intelligence*, pages 272-286. Tioga Publishing, Palo Alto, CA, 1981.
- [62] Suchman, Lucy A. *Plans and Situated Actions: The problem of human machine communication*. Cambridge University Press, 1987.
- [63] Sussman, G.J. *A Computer Model of Skill Acquisition*. Elsevier North-Holland, 1975.
- [64] Tate, A. Generating project networks. In *Proceedings IJCAI-77*, pages 888-893, Boston, 1977.
- [65] Taylor, Donald H. The role of human action in man-machine system errors. In Leplat Rasmussen, Duncan, editor, *New Technology and Human Error*. John Wiley and Sons, New York, 1987.
- [66] Teitelman, W. Do what i mean: The programmer's assistant. *Computers and Automation*, 21(4):8-11, April 1972.
- [67] Tenenber, J. Planning with abstraction. In *Proceedings of AAAI-86*, pages 76-80, 1986.
- [68] VanLehn, Kurt. Learning a domain theory by completing explanations. In Michalski Mitchell, Carbonell, editor, *Machine Learning: A Guide to Current Research*, pages 359-362. Morgan Kaufmann, 1986.
- [69] Vere, S. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:246-267, 1983.
- [70] Wilkins, D., W. Clancey, and B. Buchanan. Odysseus: A learning apprentice. In *Proceedings of the 1985 International Machine Learning Workshop*, pages 221-223, Skytop, PA, June 1985.

- [71] Wilkins, D.C. Knowledge base refinement using apprenticeship learning techniques. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.
- [72] Wilkins, D.E. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269-301, 1984.
- [73] Wilkins, D.E. Recovering from execution errors in sipe. SRI International Technical Report 346, SRI International, 1985.
- [74] Wilkins, D.E. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan-Kaufman Publishers, San Mateo, CA., 1988.
- [75] Williamson, Keith. Learning from exceptions in databases. In Michalski Mitchell, Carbonell, editor, *Machine Learning: A Guide to Current Research*, pages 3775-377. Morgan Kaufmann, 1986.