AL-TP-1992-0010

ARTIFICIAL INTELLIGENCE IN TRAINING (AIT)

Charles P. Bloom
Peter T. Bullemer
Mohammed Nasiruddin
Jodi Ray
Robin R. Penner

Honeywell Sensor and System Development Center
3660 Technology Drive
Minneapolis, MN 55418

HUMAN RESOURCES DIRECTORATE
TECHNICAL TRAINING RESEARCH DIVISION
Brooks Air Force Base, TX 78235-5000

April 1992

Interim Technical Paper for Period February 1991 – October 1991

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5000

NOTICES

This technical paper is published as received and has not been edited by the technical editing staff of the Armstrong Laboratory.

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Office of Public Affairs has reviewed this paper, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.

KURT STEUCK
Contract Monitor

HENDRICK W. RUCK, Technical Director
Technical Training Research Division

RODGER D. BALLENTINE, Colonel, USAF
Chief, Technical Training Research Division

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1992	Interim – February 1991 – October 1991

4. TITLE AND SUBTITLE

Artificial Intelligence in Training (AIT)

5. FUNDING NUMBERS

C – F33615-90-C-0013
PE – 62205F
PR – 1121
TA – 10
WU – 49

6. AUTHOR(S)

Charles P. Bloom
Peter T. Bullemer
Mohammed Nasiruddin
Jodi Ray
Robin R. Penner

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Honeywell Sensor and System Development Center
3660 Technology Drive
Minneapolis, MN 55418

8. PERFORMING ORGANIZATION REPORT NUMBER

C920134

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

Armstrong Laboratory
Human Resources Directorate
Technical Training Research Division
Brooks Air Force Base, TX 78235-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AL-TP-1992-0010

11. SUPPLEMENTARY NOTES

Armstrong Laboratory Technical Monitor: Dr. Kurt Steuck, (512) 536-2034

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The objective of the current research program was to develop a prototype knowledge acquisition tool that could enable the investigation of knowledge acquisition issues for instructional systems. Our approach to accomplishing this objective entailed the identification of the types of knowledge to be acquired, the evaluation of methods of acquiring such knowledge, the specification of appropriate interaction metaphors for the direct acquisition of knowledge from multiple human sources, and the iterative development of an Artificial Intelligence in Training (AIT) prototype that operationalized the findings of the first three tasks. The AIT prototype demonstrated critical characteristics of a knowledge acquisition environment designed to acquire multiple types of knowledge using a simulated domain environment in which multiple, animated views of knowledge are completely introspective and manipulatable. In addition, strategic knowledge construction was demonstrated based on observations of user actions. We described sample knowledge acquisition scenarios to characterize the functionality supported by the AIT prototype. The following areas for additional research were recommended: acquisition of instructional heuristics, dynamic construction of justification knowledge, use of higher level problem representations such as the goal-action hierarchy, and most importantly, the veridicality of the expertise captured using these methods.

14. SUBJECT TERMS

Instructional heuristics
Intelligent tutoring systems
Justification construction
Knowledge acquisition

15. NUMBER OF PAGES

70

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Table of Contents

List of Figures

List of Tables

PREFACE

The mission of the Intelligent Training Branch of the Technical Training Research Division of the Human Resources Directorate of the Armstrong Laboratory (AL/HRTI) is to design, develop, and evaluate the application of artificial intelligence (AI) technologies to computer-assisted training systems. The current effort was undertaken as part of HRTI's research on intelligent tutoring systems (ITS) and ITS development tools. The work was accomplished under workunit 1121-10-49, Artificial Intelligence in Training. The proposal for this research was solicited using a Broad Agency Announcement.

Summary

The objective of the current research program was to develop a prototype knowledge acquisition tool to assist in the investigation of knowledge acquisition issues for instructional systems. Our approach entailed identifying the types of knowledge to be acquired, evaluating methods of acquiring such knowledge, specifying appropriate interaction metaphors for the direct acquisition of knowledge from multiple human sources, and iteratively developing an Artificial Intelligence in Training (AIT) prototype that operationalized the findings of the first three tasks. The current research effort produced a number of knowledge acquisition tool requirements for training troubleshooting activities in complex systems. This domain requires the integration of at least three different types of knowledge: (1) knowledge about systems structure, function and their normal and abnormal behavior (how it works), (2) knowledge about interacting with these systems to perform a task (how to use it), and (3) knowledge about how to teach students to perform a task (how to teach it). In addition, the following principles were specified for a knowledge acquisition environment designed to acquire these types of knowledge:

- Knowledge acquisition should take place in a simulated environment in which users interact directly with a representation of their domain that emulates their real-world interactions and promotes the solicitation of natural and instinctive responses through a direct manipulation interface.

- The knowledge acquisition tool should employ an action-based solicitation method in which all user actions are recorded in sequence and in "pseudo-real-time."

- The knowledge acquisition tool should support knowledge integration by providing users with multiple, animated views of knowledge that are completely introspective and manipulatable.

- The knowledge acquisition tool should support knowledge construction from observations of user actions.

The following areas for additional research were recommended: Acquisition of instructional heuristics, dynamic construction of justification knowledge, use of higher level problem representations such as the goal-action hierarchy, and most importantly, the accuracy of the expertise captured using these methods. By demonstrating new approaches and integrating established approaches to knowledge acquisition, we hoped to stimulate additional research on approaches to the direct acquisition of multiple knowledge types from multiple knowledge sources for intelligent instructional environments.

Section 1
Introduction

Methods for facilitating intelligent tutoring systems (ITS) development are key concerns in both the Air Force (Regian, 1989) and industry (Bloom, Bullemer, and Cochran, 1989). Of the many technical challenges facing the developers of ITSs, the most challenging and intransigent revolve around the issue of knowledge acquisition. Traditional knowledge acquisition problems include the knowledge acquisition bottleneck, the knowledge filtration problem, cognitive biases, and the difficulty in "unpacking" compiled expertise. Compounding these traditional knowledge acquisition problems, knowledge acquisition for instructional systems introduces additional problems, including the need to acquire multiple knowledge types from multiple sources (i.e., various types of domain and instructional knowledge), the need for the knowledge input by one expert to be completely understood by another expert, necessitating the acquisition of "deep" knowledge that can be both inspected and manipulated (i.e., actions *and* justifications, the acquisition of which previously required knowledge engineers to independently employ a variety of direct and indirect knowledge acquisition methods), and the fact that the acquisition of instructional knowledge and domain knowledge is interrelated.

Employing the traditional knowledge acquisition scenario of having knowledge engineers acquire knowledge from domain experts and instructional developers separately for an ITS is untenable for a variety of reasons. First, there is the long and costly development cycle produced by this scenario. Second, because knowledge engineers use a variety of widely differing direct and indirect knowledge acquisition methods, the knowledge acquired can often be infused with inaccuracies, making it a "best guess," rather than a true representation of expertise. Third, because there are many domains represented, the knowledge input by an expert of any one domain must be accessible to and understood by an expert of any other domain.

The objective of the current research program was to develop a prototype knowledge acquisition tool that can assist in the investigation of knowledge acquisition issues for instructional systems by demonstrating new approaches or integrating established approaches to knowledge acquisition that will help illuminate research questions regarding the direct acquisition of multiple knowledge types from multiple knowledge sources. The approach employed to accomplish this objective comprised four major tasks. The first task involved identifying the types of knowledge to be acquired for an ITS. The second task involved evaluating methods of acquiring such knowledge, including the exploration and evaluation of new alternatives if necessary. The third task involved defining appropriate metaphors of interaction to facilitate the direct acquisition of knowledge from human sources without needing the help of knowledge engineering or AI programming specialists. The fourth and final task involved iteratively developing a prototype that operationalizes the above findings.

To summarize briefly, the solution being proposed in the current project has several aspects to it. First, domain experts and instructional developers must interact directly with the knowledge acquisition tool. This should alleviate some of the traditional knowledge acquisition problems associated with having a knowledge engineer in the loop, such as the knowledge acquisition

bottleneck and knowledge filtration problems. Second, the knowledge acquisition method(s) employed must be specifically designed to promote natural and instinctive responses by both domain and instructional experts. To demonstrate this aspect, we are proposing the following:

- Provide experts and instructors with an environment that closely emulates their real-world interactions with their domain.

- Ensure that the knowledge input by any one expert can be inspected and manipulated by any other. This is an instantiation of the notion of a "glass box architecture" (Anderson, 1988; Wenger, 1989), in which we enable multiple views and representations of the acquired knowledge that can be manipulated.

- Employ a knowledge acquisition method that is able to obtain information both about what experts do and why they are doing it (Gruber, 1991) in a manner that does not distract them from their primary task.

Third, the tool must be capable of both supporting and integrating multiple types of input from multiple experts to ensure that the knowledge acquired is as free from individual cognitive biases as possible.

This report is organized as follows: Section 2 contains a description of critical issues in knowledge acquisition, including those introduced by expanding knowledge acquisition to the domain of intelligent tutoring systems. Section 3 describes the knowledge acquisition approach proposed to address these issues, including discussions of the prototype's knowledge requirements and concepts underlying the knowledge acquisition capabilities implemented in the prototype. Section 4 contains scenarios depicting how sample knowledge acquisition sessions might be conducted using the AI in Training (AIT) prototype. Section 5 contains conclusions reached from the current effort. Section 6 contains recommendations and a discussion of future research directions and issues to be addressed in the area of knowledge acquisition for ITSs.

Section 2
Critical Issues in Knowledge Acquisition

The problem of knowledge acquisition is one of the biggest issues affecting the development and delivery of artificial intelligence (AI) systems such as expert systems and intelligent tutoring systems. Early AI systems were plagued by long and costly development cycles, driven primarily by the difficulty of knowledge acquisition. The most successful attempts to address these problems have involved developing systems in which the knowledge engineers and AI programmers have been removed from the knowledge engineering loop entirely.

So far, however, such automated knowledge acquisition has been possible only for specific domains. While the resulting domain-specific shells are very effective in providing solutions to tasks for which they were designed, they are not easily generalized to new areas. Furthermore, it has been well established that expertise is made up of several different kinds of knowledge (Berry, 1987; McGraw & Seale, 1988). When expanding a system's scope from expert advisory to instruction, failure to capture and represent different knowledge types, which could come from multiple sources (i.e., domain expert, cognitive scientist, teaching expert) could result in a tutor that is both limited in scope and lacks the flexibility of a human tutor (Woolf and Cunningham, 1987).

Some of the critical issues in knowledge acquisition revolve around the problems of the knowledge acquisition bottleneck and knowledge filtration, the nature of compiled expertise, and cognitive bias. Additional issues present themselves when designing a system for instruction as opposed to a system for performance. These issues are discussed further in the subsections below.

2.1 Knowledge Acquisition Bottleneck

At present, knowledge acquisition for AI systems, in general, is very inefficient. In the early days of AI systems development, a typical development scenario would have a knowledge engineer interviewing a subject matter expert to acquire the knowledge to be incorporated into the system. The knowledge engineer would in turn interact with an AI programmer to convert the acquired knowledge into an identified AI representation written in a general-purpose AI language, such as OPS-5, LISP, or Prolog. This process of characterizing the knowledge of experts is the significant bottleneck in intelligent systems development The obvious drawback to this methodology was that the high-level data structures and reasoning schemes that represented the domain knowledge had to be built from scratch for each new application. The technique amplifies the inefficiency of software development, requiring knowledge engineers to write one kind of program and programmers to convert it to another.

2.2 Knowledge Filtration

In addition to the bottleneck problem, there is also the filtration problem, which refers to the multiple transformations performed on knowledge as it passes from expert to knowledge engineer

to programmer to software. Inaccuracies in the system's encoded knowledge can result from the filtering or processing of that knowledge by two nonexperts prior to its entry in the system (Cochran, Bloom, and Bullemer, 1990).

2.3 Compiled Expertise

The solution to these problems is obvious yet difficult to accomplish: eliminate the need for knowledge engineers and programmers by developing authoring environments easy enough for domain experts to use unassisted. The essential difficulty of this strategy is that expertise is not so easily acquired. One of the fundamental difficulties in most knowledge acquisition methods is that the human thinking process we wish to understand and model is not available to direct observation. This difficulty is further compounded by the fact that expertise is typically not reportable, due to the compilation of knowledge that results from extensive practice in a domain of problem-solving activity (Anderson, Greeno, Kline and Neves, 1981). Because experts "see" the solutions to complex problems more often than they deduce them, their expertise is said to be "compiled," that is, the individual elements of expertise are not available. Thus, knowledge engineers tend to use a variety of different knowledge acquisition techniques and methods developed to try to gain access to the experts' reasoning processes. Olson and Rueter (1987) reviewed a number of these methods of knowledge acquisition, classifying them into direct and indirect methods. Table 1 lists a number of different knowledge acquisition methods:

Table 1. Methods of Knowledge Acquisition

Direct Methods	Indirect Methods
Interviews	Multidimensional Scaling
Questionnaires	Hierarchical Clustering
Observation of Task Performance	Weighted Networks
Protocol Analysis	Ordered Trees
Interruption Analysis	Repertory Grid Analysis
Drawing Closed Curves	Device Models

All the direct methods for acquiring knowledge listed in Table 1 can elicit a wide range of knowledge types. However, their drawback is that until now, they have been techniques employed by knowledge engineers, not techniques implemented in a computer-based knowledge acquisition tool usable directly by domain experts. On the other hand, even though all the indirect methods lend themselves to being implemented in knowledge acquisition tools, and in many cases, individually, they have been, they illuminate only particular types of domain expertise making them unsuitable for the task of acquiring the multiple types of knowledge required by an ITS. In addition, each of these methods was developed as a compromise for dealing with the problems elaborated on previously. The result of applying any one of these methods is the acquisition of knowledge that can be biased both by the method itself and by the representation employed by that method.

For example, the repertory grid analysis approach to acquiring expert knowledge begins by specifying a taxonomy of domain concepts and their interrelationships, such as listing problem features, solution states, and the relationship between problem features and solution states (Boose, Bradshaw, Kitto, and Shema, 1989). Procedural rules are then derived that reflect the underlying functional model of the domain. Although this approach allows experts to interact directly with the knowledge acquisition tool, entering their knowledge using their own vocabulary, heuristic knowledge is not captured *directly* using this approach. Declarative knowledge of the domain concepts is captured and manipulated to produce if-then rules that are used to simulate or model task performance. The resulting procedural knowledge is derived independent of any specific task situation, often failing to capture an expert's heuristic knowledge.

What is needed is a computer-based knowledge acquisition method that combines the best of both the direct and indirect methods. First, it is important to be able to accomplish on a computer what knowledge engineers accomplish using direct acquisition methods. It is widely acknowledged that the best way to discover how experts make a judgment, diagnosis or design decision is to watch them work on a real problem (Olson and Rueter, 1987). Therefore, what would be desirable would be to implement a knowledge acquisition method that automatically records all the experts' actions (i.e., what they did, when they did it, and the order of their actions) while they are engaged in a real-world-like problem solving activity on the computer. In addition, this computer-based knowledge acquisition tool needs to be able to accomplish what knowledge engineers accomplish using indirect methods: the unpacking of expertise in the form of explanations and justifications for the various actions an expert takes and decisions the expert makes.

According to Gruber (1991), justifications are declarative specifications of what the situation is, what the possible choices or actions are, and a set of reasons, defined as specifications of relevant features of the situation or choice, for a specific choice being appropriate. To determine why a choice is appropriate in a given situation it is necessary to solicit a set of relevant features of both the situation and the choice from the expert. Solicitation of these features should take place within the context of a specific example in a running system in which the features can be computed. Within the domain of diagnosis, whether it be medical or complex-system faults, situations can be defined as diagnostic steps or hypotheses and choices can be defined as the diagnostic tests the diagnostician performs to rule out or confirm those hypotheses.

2.4 Cognitive Bias

An additional problem that has always affected expert systems, one that unfortunately becomes amplified by allowing domain experts to develop expert systems directly, is that the expert system is only as good as the expert (Cleaves, 1987; Meyers and Booker, 1989). Often, cognitive biases, such as inaccurate or incomplete domain representations, are accidentally built into a knowledge base. Cognitive bias can result from either limitations in an expert's knowledge or expertise, or from the expert having to generate procedures from memory during knowledge acquisition. Cognitive limitations in recalling or simulating the tasks could also produce gaps or inaccuracies in the knowledge acquired. While this may not greatly affect the utility of advisory expert systems (if the system is as good as a carefully selected expert in solving problems in a particular domain, any cognitive bias in the approach to those solutions is an *advantage*), the interaction of cognitive

biases can lead to significant problems when the knowledge is used for more than one purpose, which is the case for training systems.

One solution to the cognitive bias problem requires system developers to provide more simultaneous assistance to experts. For example, knowledge engineers are skilled in detecting some of the symptoms of bias and in questioning experts to ascertain the gaps in their knowledge. In an automated knowledge acquisition tool, much of this assistance could instead be provided by having the expert(s) work in an environment that closely emulates their real-world domain interactions. By engaging experts in interactions with an emulated environment that responds to their actions as their actual domain would, and by constraining those interactions to environment-legal activities only, one provides the context necessary to avoid or detect biases as a result of the experts' cognitive limitations.

Another solution to the cognitive bias problem involves the use of multiple experts for a single type of domain knowledge. The use of multiple experts in this sense reduces the likelihood that idiosyncratic and/or inaccurate knowledge will be encoded in the knowledge base. In an automated knowledge acquisition tool, this could be accomplished by developing an approach for acquiring expertise distributed across individuals, that is, it would allow experts to input their knowledge separately, combined with facilities to review, evaluate, and integrate the acquired distributed expertise.

2.5 Knowledge Requirements

A knowledge base for use in a training system will need to be able to represent all the expertise of a specific domain as well as the various types of knowledge used in the expression of that expertise. With the quantity of knowledge that could be acquired for any domain and the likelihood that the different types of knowledge will come from different sources, a different type of multiple expert problem is created. As such, additional issues that must be considered in designing a knowledge acquisition tool for instructional systems are identifying the types of knowledge to be acquired and the relationships between those types of knowledge.

The objectives of a system designed for instruction are different from the objectives of a system designed for task performance. Consequently, the knowledge acquisition requirements will be different as well. For example, Clancey (1984) reported that an attempt to use MYCIN as a basis for teaching expert diagnostic skill proved difficult because specific types of information were either absent or implicitly represented in the systems production rules. Clancey reported that using an expert system to teach requires a shift in orientation (objective) from simply trying to generate good task solutions to simulating in some degree of detail the reasoning processes itself. Types of information required depend on the instructional objectives and methods. An ITS requires more explicit, psychologically valid models of task performance.

A knowledge acquisition tool for instructional systems will need to be able to acquire a number of different types of interrelated knowledge. To teach problem solving in complex systems, it will be necessary to acquire knowledge about the system itself, about the types of problems one can

encounter in that system, about the ways experts solve those problems, and about the ways instructors teach that knowledge.

To begin with, to emulate the expert's real-world interactions in a domain, the knowledge acquisition tool will need to possess and use an abundance of knowledge about the structure, function, and behavior of the various devices, subsystems and systems of the domain, that is, device knowledge. When the expert takes an action on some object in the knowledge acquisition tool's domain representation, the represented system's reaction to that action should emulate the reaction expected in the actual system. Next, the knowledge acquisition tool will need to acquire knowledge about problems. The basis for teaching problem solving knowledge, or acquiring problem solving expertise, is direct interaction with domain problems. One approach to accomplishing this is to enable experts to build libraries of real domain problems, reflecting both common and uncommon occurrences. This will involve the initialization of input and control values and parameters as well as the introduction of faults (and their consequences) into the system being represented. This acquired problem knowledge can then be used to initialize the tool's representation of the device knowledge to create scenarios around which to conduct the "real-time" acquisition of problem solving knowledge. This will involve recording the actions the expert takes, as well as the represented system's behaviors in response to those actions, and constructing justifications for those actions to give the problem solutions explanatory power. Finally, the knowledge acquisition tool will need to support the instructor's use of all the previously acquired knowledge in instructional interactions with "students." This will involve enabling the instructor to review and comprehend the problems being solved and the expert approaches to solving those problems so that they can evaluate the student's performance and depth of knowledge. In addition, the tool needs to support methods for recording the instructor's interventions and construct justifications for those interventions from the recorded actions.

In the section that follows we will describe the AIT system at a conceptual level, including a discussion of the system's knowledge requirements and descriptions of the various features of the knowledge acquisition environment developed in response to the issues elaborated on previously.

Section 3
Current Approach: AIT

3.1 AIT Knowledge Requirements

The AIT demonstrates acquisition of knowledge used to support training of troubleshooting activities in a process control domain. Support for the training and performance of troubleshooting activities in the process control domain is provided by three basic types of knowledge. These include:

- Knowledge about systems and their normal and abnormal processes and behavior (how it works),
- Knowledge about interacting with these systems to perform a job or task (how to use it),
- Knowledge about how to teach students to perform a job or task (how to teach it) (adapted from Kieras, 1988).

3.1.1 "How-it-Works" Knowledge

A basic knowledge requirement for a simulation-based ITS is device knowledge. Device knowledge characterizes the structure, function and behavior of a process control system. System designers are typical sources of expertise regarding device knowledge, particularly with respect to normal structure, function and behavior. Field technicians and engineers are typical sources of expertise for knowledge about a device's abnormal structure, function and behavior. System designers often have less experience with abnormal behavior.

Table 2 contains a list of common representations of device knowledge. These representations contain knowledge about the physical layout of major systems, subsystems, components and their interconnections; the flow of data between system components; and normal behavior of the system in terms of conditional state transitions or cause-effect relationships.

There are a number of research programs currently investigating methods of acquiring and building device model representations (e.g., IMTS, ICATT) as well as commercially available tools for building object-based system simulations. To avoid duplicating these efforts, it was decided that the current effort would not address this issue, assuming instead that one of the methods existing or under investigation could be incorporated into the AIT system to support the acquisition of device knowledge. Instead, it was decided to represent device knowledge to the extent necessary to support the acquisition of the other knowledge types (see Section 3) and to do so in an economical and efficient manner. As such, it was decided to represent device knowledge in AIT using a qualitative behavioral model of a process control subsystem at the level of replaceable components. A qualitative behavioral model enables the system to simulate the

Table 2. Real-World Representations of Device Knowledge

Device Knowledge Representations
Component operating characteristic (system specifications)
Process control descriptions (control schematics)
General physical principles (physics)
System operation descriptions (manuals)
Functional block diagrams
Data flow diagrams
Component-level schematic diagram (usually at subsystem level)
PID drawings

behavior of the subsystem. The behavioral model relates qualitative changes in component properties to qualitative changes in the component's output. The specification of the causal relation between each component's input and output enables the qualitative simulation of the subsystem's normal behavior. The interface presents a component-level schematic diagram with access to the input and output of the subsystem and properties of the individual components, such as percent opening of a valve.

Because the task domain of the AIT is troubleshooting, the behavioral model must be able to simulate abnormal behavior as well as normal behavior. Knowing about how a device can fail involves knowing what components can fail and what the impact would be on the behavior of the system. Thus, the device knowledge of AIT represents failure states of each component and their impact on the component's output.

3.1.2 "How-to-Use-it" Knowledge

"How-to-use-it" knowledge is the knowledge a user has about interacting with a system to perform a job or task. The scope of the "how-to-use-it" knowledge in the AIT demonstration prototype was limited to troubleshooting system faults in a simplified, prototypical process control application. Troubleshooting can be considered a diagnostic problem solving task, requiring the identification of changes in normal system behavior (deviation from expected operation) and localization of the causes of abnormal behavior (Rasmussen, 1981). This means that in addition to an understanding of the application's device knowledge that enables experts to identify instances of normal and abnormal system behavior, troubleshooters need to possess two types of "how-to-use-it" knowledge: problem knowledge and problem solving knowledge.

Problem knowledge can be defined as knowledge of the specific types of problems, symptoms and faults for a system as well as knowledge of the mappings from symptoms to faults and problems, that is, knowing what components can fail and how they can fail. Sources of this type of knowledge are typically field technicians (operators, troubleshooters) and in rare cases, system designers.

Problem knowledge in AIT includes an initial symptom description, a fault state (property) and a value for that fault state. Classes of faults have been enumerated in AIT's device knowledge. Problem specification involves describing the initial symptoms, selecting components and either initializing input values or choosing a fault from a list of potential faults and assigning a value to the fault. Circumstantial information and fault probability knowledge can be included using explanations attached to specific problem setup actions.

Problem solving knowledge can be characterized as knowledge about problem solving strategies used to generate problem solving goals and knowledge about procedures for manipulating and testing the system. The diagnostic problem solving task can be described as follows: It begins with the identification of some initial problem state (initial symptoms) that indicates that the system is behaving abnormally. Next come the generation of a set of possible faults (hypotheses) that could have produced the initial symptoms. Finally, there is the performance of information gathering activities that allow the problem solver to confirm or rule out hypotheses in the hypotheses set.

Diagnostic problem solving expertise is often thought to consist of heuristics (rules of thumb) for generating possible problem hypotheses and for evaluating the hypothesis given available information (Clancey, 1985) as well as the strategic knowledge that guides problem solvers in deciding what action to perform next (Gruber, 1991). Problem solving expertise is acquired with years of experience in performing a task in a given domain. Heuristics are often represented as associations between problem symptoms and fault states.

Gruber (1991) has proposed that strategic knowledge can be represented in terms of justifications for specific problem solving actions. Justifications are specified in terms of the relevant features of the current problem situation that explain why an action taken is the appropriate action. Relevant features of a current situation include problem solving goals, current hypotheses, and knowledge about the outcome of tests that provide evidence for or against current hypotheses.

AIT represents problem solving knowledge in terms of sequences of actions used to localize problem causes and the generation and elimination of problem hypotheses from a hypothesis space. Problem solving actions include viewing available component property information and conducting tests to obtain additional component state information.

In AIT, strategic knowledge is represented in AIT in terms of task objectives and justifications for performing problem solving actions. Justifications are acquired in two ways. First, AIT dynamically generates justifications for each user action in terms of its impact on the problem hypothesis space. Each action can add and/or rule out hypotheses from the hypothesis space. Second, a user may attach an explanatory statement to any problem solving action directly. In addition, following completion of their problem solving activity, experts are instructed to specify a goal-action hierarchy that shows the relation between task objectives and the activities performed to achieve those objectives. The goal-action hierarchy contains goal and subgoal states that are linked to each of the knowledge acquisition actions.

AIT supports acquisition of problem solving knowledge by observing and recording problem solving actions and by soliciting and updating a problem hypothesis space. AIT records user actions in accessing component information and prompts users to add and/or remove hypotheses

following each information acquisition action. For each problem solution, users specify a goal-action hierarchy. Users enter goal and subgoal descriptions with links to problem solving actions showing the relation between task objectives and the activities performed to achieve those objectives.

3.1.3 "How-to-Teach-it" Knowledge

Murray and Woolf (1991) have defined Instructional knowledge as "the knowledge related to teaching and learning a given domain," including in their definition only the types of knowledge that we consider to be instructional plan information: knowledge about the structuring of a domain (e.g., the selection and sequencing of lessons and topics) and the identification of parameters or "bugs" to monitor for student evaluation purposes.

There has been considerable research undertaken to develop tools and methods to facilitate the acquisition of instruction plan information (Bonar, Cunningham, and Schultz, 1986; Macmillan, Emme, and Berkowitz, 1988; Merrill, 1989; Murray and Woolf, 1991; Russell, Moran, and Jordan, 1988). Murray and Woolf have developed a prototype knowledge acquisition system called KAFITS (Knowledge Acquisition Framework for Intelligent Tutoring Systems), which was designed to acquire and represent knowledge from instructional experts. KAFITS incorporates instructional design paradigms and facilitates the rapid creation and manipulation of multiple tutoring strategies. Merrill (1989) has developed an instructional system design expert system for instructional designers that guides instructional design decisions so that resulting products can more adequately implement what is known about learning and instructional design. Russell et al. (1988) have developed the instructional design environment (IDE), a sophisticated, integrated computer-based environment for instructional design. IDE facilitates the instructional design process and the creation of instructional materials by providing a hypermedia-based, flexible representation workbench. In IDE, all the information used for designing and developing a course can be represented and manipulated. IDE simplifies the course development task and enables the construction of more consistent instruction by providing tools and structures that automate many of the routine tasks.

Our research indicates that there is an additional category of instructional knowledge—knowledge about how instructors dynamically and spontaneously interact with a student—that has either been ignored or else dismissed as being irrelevant or too difficult to obtain (Murray and Woolf, 1991). We call this type of instructional knowledge instructional heuristics, and it is the premise of the current research effort that acquiring this type of instructional knowledge for an ITS is critical, particularly within the context of teaching problem solving in complex systems, a significant problem for industry and the military.

Human instructors respond to student performance in a number of different ways, for example, providing advice or demonstrating how to perform a task. Instructional heuristics specify the instructional situations for different types of instructional responses. Situations can be defined in terms of features of student troubleshooting actions such as type of information accessed, type of diagnostic test performed, or relation of student strategies to expert strategies. The types of instructional responses can vary from making certain types of advice available to presenting an expert solution to the problem. Table 3 lists a number of types of intervention derived from the research of Burton (1988) and Wenger (1989).

Table 3. Types of Assistance

Type of Intervention	Description	Prototypical Instructor Action	Examples
Assistance	The instructor takes over parts of the problem-solving task, freeing the student to concentrate on the remaining parts. This type of intervention is usually applied to nonessential parts of the task that may be blocking the student from progressing and grasping the larger structural properties of the domain.	Instructor provides student with "answer" to that part of the problem solving task.	Do-for, remind
Introspection	The instructor allows the student to review their actions and decisions by presenting them in an appropriate manner and allowing the student to browse through them. This type of intervention encourages the student to reflect on their problem-solving activities.	Instructor presents student/expert goal/action tree to student.	Review, critique
Modeling	The instructor performs the task; modeling for the student the way an expert does the activity. An important component of modelling is having the system articulate the decisions it is faced with and the strategies it is using to make these decisions. This type of intervention is useful in making explicit the strategies the expert uses, thereby giving the student an example to follow.	Instructor "plays-back" experts solution for student.	Demonstrations, examples, counter-examples, explanation, justifications
Coaching	The instructor breaks in and makes suggestions. When suboptimal behavior or performance is recognized, the "coach" breaks in to give advice. This type of intervention is most useful in situations where one wants to give the advice designed to overcome some specific weakness noted in the student. One could also try to discern "what" the nature of the weakness is the tutor is responding to.	Instructor provides a specific "hint" or piece of advice to student.	Advice, hints, suggestions, choices
Material Control	The instructor specifies some particular material or problem to present to the student. This type of intervention is useful for students who seem to be having difficulty putting basic concepts together to form a deeper model of the domain.	Instructor requests some particular material or problem be presented to the student.	Topic selection/ sequencing, task generation/ selection, examples

AIT supports acquisition of instructional heuristics by recording an instructor's observations of a student's problem solving activity as well as the instructor's intervention specifications. While viewing a student's sequence of troubleshooting actions, an instructor can assess the appropriateness of each student action in terms of its relationship to the expert's goal hierarchies. The instructor can also assess the system inquiries and tests initiated by the student as well as the student's interpretation of those test results as indicated by changes in their hypothesis space. In addition, the instructor can specify an instructional intervention following any student action.

3.2 AIT Environment

There are four key principles that guide knowledge acquisition in AIT. These principles were driven by the knowledge acquisition issues described in Section 2 and are based on current research advances in knowledge acquisition, analyses of existing knowledge acquisition and instructional systems, and feedback from a number of domain experts and instructors.

Knowledge acquisition must take place in a *simulated environment.* In this simulated environment, users interact directly with a representation of their domain that emulates their real-world interactions and promotes the solicitation of natural and instinctive responses through a direct manipulation interface; that is, users take actions on the objects representing the components of the domain, and the domain representation's response to their action is the same as the user would expect in a real domain. An issue in constructing a simulated environment is the fidelity of the computer-based environment to the real-world working environment (Duncan, 1981).

The approach taken in the AIT prototype is to simulate information available in the real-world but not necessarily the format in which it is presented. For purposes of tracking problem solving performance, component property information is presented only upon request of the user. Since information gathering is a critical aspect of diagnostic problem solving, it is important to be able to explicitly represent what information is used and when. This will enable instructional experts to better understand student or expert problem solving performance better.

The knowledge acquisition tool should employ an *action-based solicitation* method in which all user actions in the simulated environment are recorded in sequence and in "pseudo real-time." In addition to recording user actions, the method should also solicit from the user in "pseudo real-time" a problem hypothesis space, that is, the set of possible problem causes that the problem solver is considering at any given time that is recorded and updated after each user action.

The knowledge acquisition tool should support *knowledge integration* by providing instructors with multiple, animated views of the knowledge that is completely introspective and can be manipulated. The use of a "glass box" architecture (Anderson, 1988; Wenger, 1989) enables users to introspect the content and structure of knowledge. Multiple views allows different users to view knowledge in a manner appropriate to their context of use. The tool should be capable of both supporting and integrating multiple input from multiple experts to ensure that the knowledge acquired is as free from individual biases as possible and appropriate for its specific context of use.

One example of accomplishing this is through the construction and use of goal-action problem solving hierarchies. The troubleshooting expert can construct a goal-subgoal hierarchy in relation to a sequence of actions just performed to depict his or her troubleshooting strategy. An instructional expert can view the same goal-action hierarchy to understand a particular expert's problem solving approach as the troubleshooting performance is animated. The instructional expert can modify the goal-action hierarchy for instructional purposes or choose to provide the student with a view of the expert's strategy. The instructional expert can view the student's problem solving performance in relation to the expert's performance by accessing a mapping of student actions onto the expert's goal-action hierarchy.

The knowledge acquisition tool should support *knowledge construction* by building representations of knowledge from observations of user actions. An indirect method of acquiring knowledge from experts is based on observing experts perform their task. By observing what actions are performed and the consequences of those actions, it is possible for an observer to infer strategic knowledge (Gruber, 1991).

Our instantiation of Gruber's (1991) model of justification construction is as follows: Justifications for the actions taken by the expert problem solvers are constructed from differences in their dynamic problem hypothesis space from one action, or set of actions, to another. After a user action has been taken, AIT looks at the user's problem hypothesis space to see if it has been updated as a consequence of that action. If so, AIT then builds a justification for that action as the change in the hypothesis space. Justifications for the interventions initiated by the instructor during a session with a particular "pseudo student" are constructed from a "snapshot" of the configuration of student actions to expert goals and subgoals at the time the intervention was initiated, and from observations by the instructor, obtained and recorded in real time, of the student's actions taken and their problem hypothesis space.

Section 4
AIT Knowledge Acquisition Scenarios

This section contains descriptions of sample knowledge acquisition sessions that might be conducted using the AIT prototype. The scenarios are narrative-based, with figures of the windows, boxes, and objects used in the AIT prototype. The purpose of this section is to provide a generic story-board that users of the AIT prototype can follow to explore the scenarios of the various features and methods employed in the areas of session startup, problem setup, expert problem solving, "pseudo student" problem solving, and student instruction.

It is important to keep in mind while reading this section that the purpose of the AIT prototype is to facilitate the evaluation of various new and established knowledge acquisition methods and interaction metaphors within the context of instructional systems development. AIT is neither exhaustive in scope and coverage, nor are the methods and metaphors implemented the only ones possible. Rather, the AIT prototype is intended to serve as a tool to help researchers identify additional issues and questions concerning knowledge acquisition for instructional systems. In each of the following knowledge acquisition scenarios, attempts have been made to identify, wherever possible, alternate approaches that could be used, or that should be evaluated as alternatives, and issues or questions deserving of additional research. The true success of this prototype will be measured by the help it is able to provide researchers in identifying additional important issues and concerns.

4.1 System Startup

Working sessions with the AIT prototype begin with the specification of a problem to develop or use. A user can create a new problem or open an existing problem in the "Open Problem" dialog box as depicted in Figures 1 and 2. When the dialog box is initially opened, a process is selected by default ("Heat Reactor" in Figure 1, the only process example supported in AIT) and presented in the "Process" option filed. If the user wishes to change the process, clicking on the "Process" option field presents a pop-up menu listing other defined processes from which the user can select (see Figure 2). Once a process has been selected, the user can either choose an existing problem by selecting the problem name from the "Defined Problems" list box and pressing the "Open" button, respectively, or else create a new problem by typing the new problem name in the "Problem" field and pressing the "Create" button.

The first step in creating a new problem is to enter the presenting symptoms in a dialog box as depicted in Figure 3. Following input of the presenting symptoms, users are presented with the "Process Simulation" window, as illustrated in Figure 4. The "Process Simulation" window contains a schematic overview of the process application. Each of the objects in the window is selectable, and the content of the information available is dependent upon AIT's particular knowledge acquisition mode.

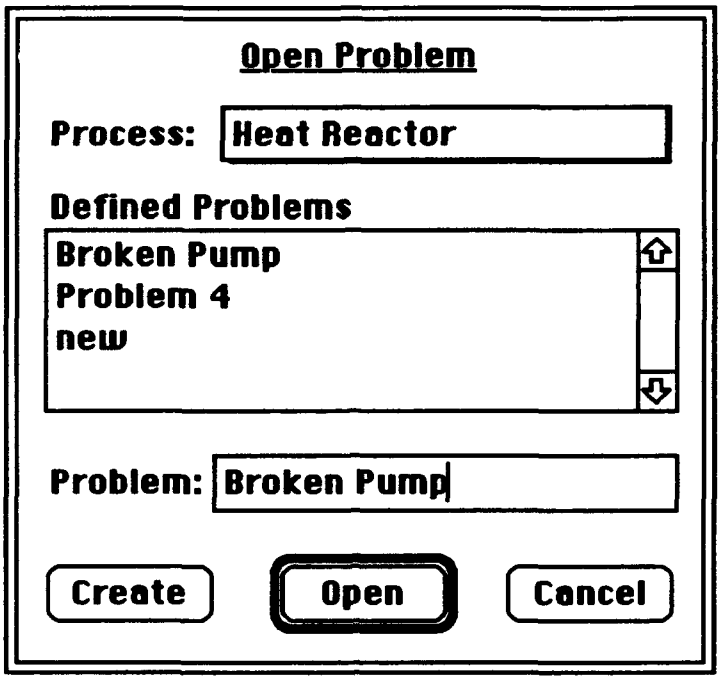

Figure 1. Session Startup

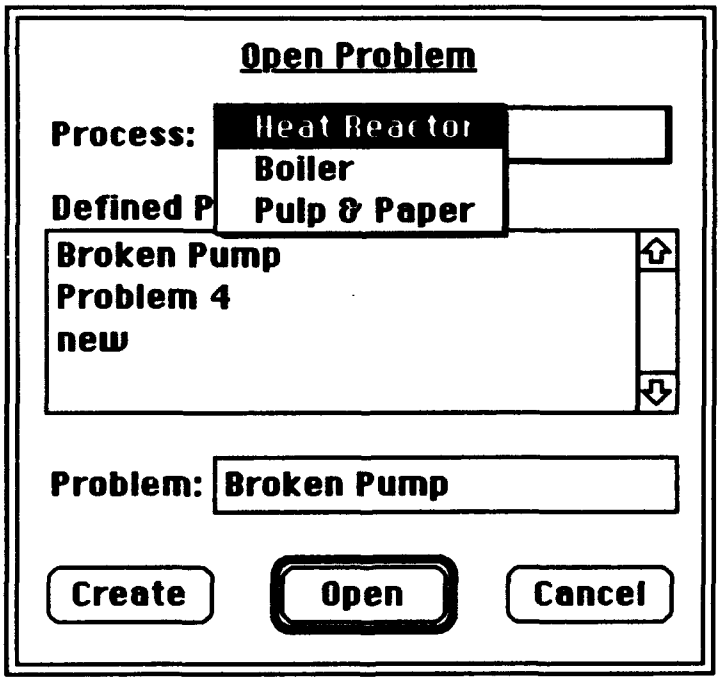

Figure 2. Process Selection (not implemented)

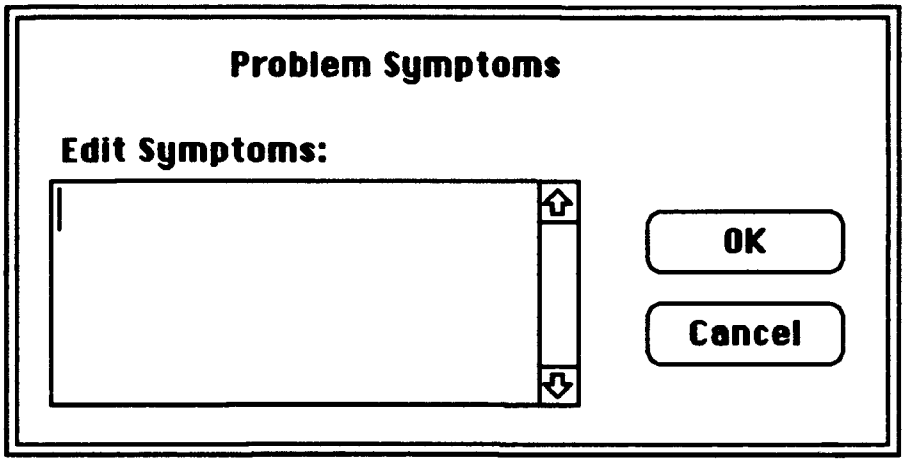

Figure 3. Problem Symptoms Input Dialog Box

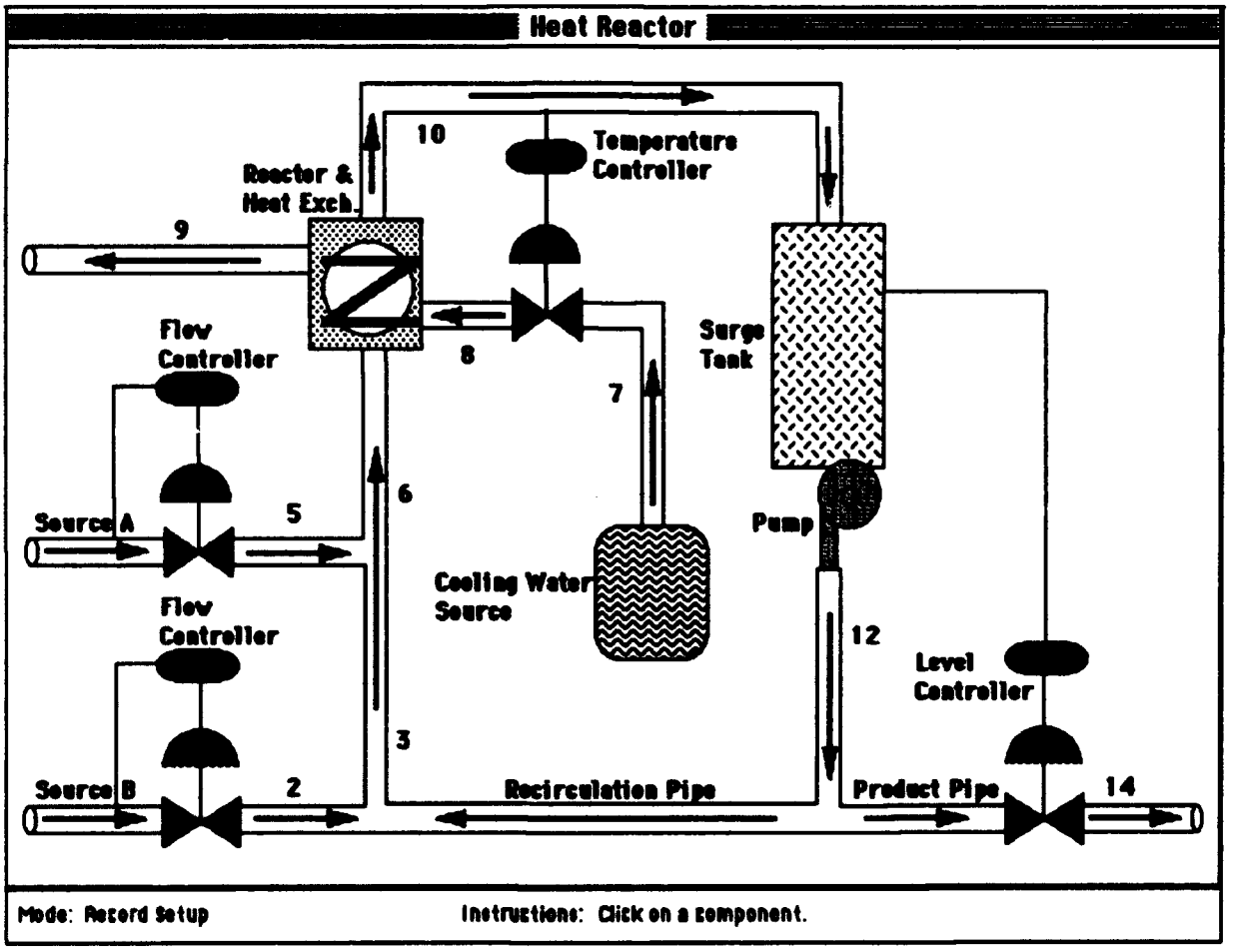

Figure 4. Process Simulation Window

Figure 5 shows a pull-down menu containing a list of possible "Problem"-related activities supported in AIT. These options are clustered into activities corresponding to the presenting symptoms, the problem setup, expert problem solving, and student problem solving. An alternate method of representing these problem-related activities might be to make them constantly available in an icon-based palette.

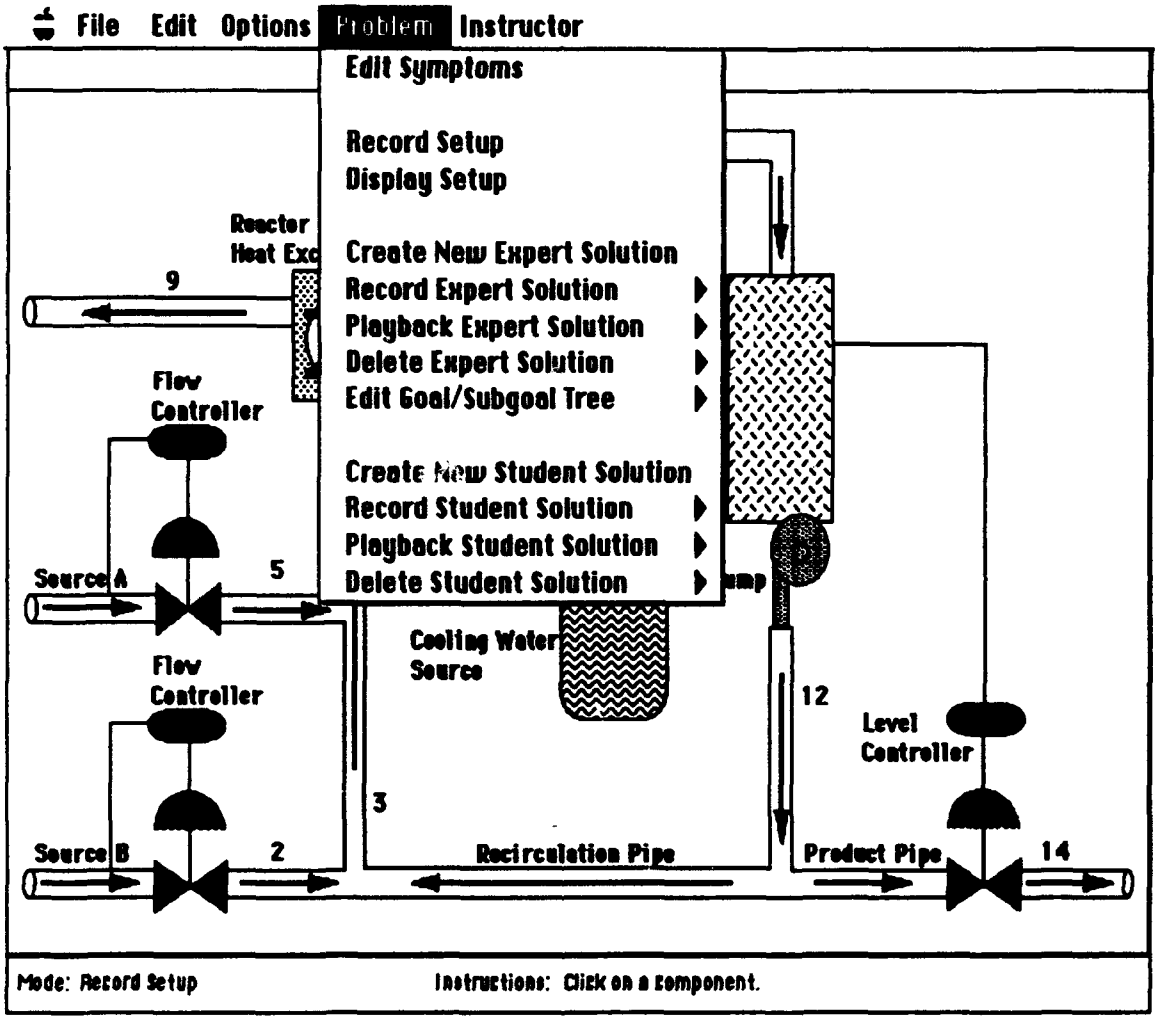

Figure 5. Problem Menu Selections

4.2 Problem Setup

As depicted in the "Problem" menu in Figure 5, problem setup consists of two main activities: record ("Record Setup") and display ("Display Setup") problem setup information. After selecting "Record Setup" from the "Problem" menu, the user clicks on an object in the simulation window (Figure 4) to bring up a problem setup dialog box as depicted in Figure 6. Note that the system mode is indicated in the lower left corner of the message box, located at the bottom of the

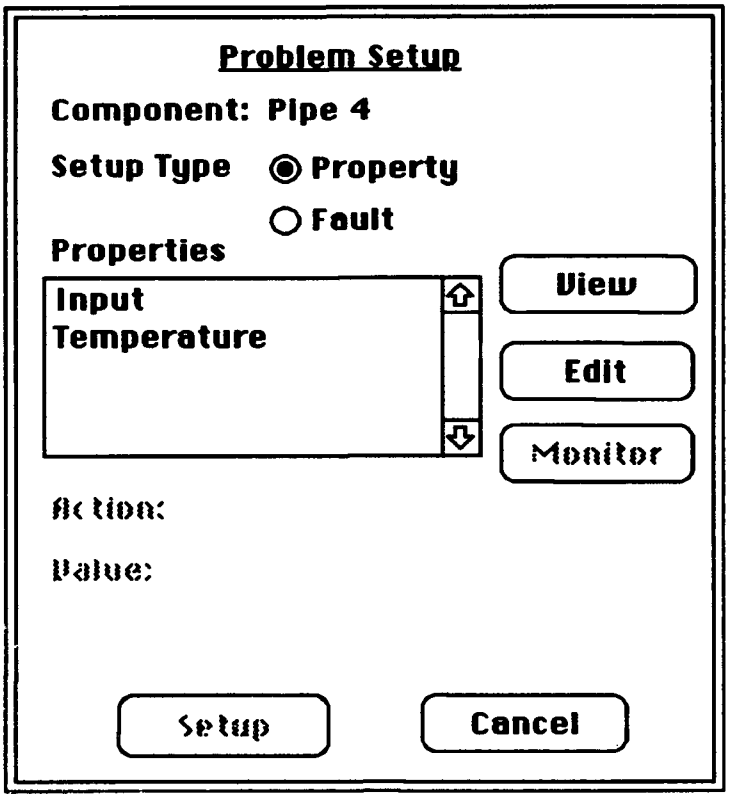

Figure 6. Problem Setup Dialog Box

simulation window. Objects in the simulation window that can be accessed are indicated by the highlighting of that object as the cursor passes over it. As can be seen in Figure 6, there are two types of setup actions a user can perform on a system component: initialize a component property value or induce a fault. The type of setup action is selected by the use of the mutually exclusive radio buttons for "Property" and "Fault," respectively.

Figures 7 and 8 show the two types of initialization activities supported in AIT's problem setup mode. Following selection of a specific property in the "Properties" list box, users can press the "Edit" button, which enables them to edit that property value either by use of a slider for continuous values (Figure 7) or by use of a pop-up menu for discrete value options (Figure 9), or else they can press "View," which presents them the component property's current value (Figure 8). The "Monitor" button and option are disabled, as indicated by by its grayed out appearance. This coding convention of graying out the interface object is used consistently throughout AIT to indicate unavailability of the interface objects.

Figure 9 illustrates how faults are induced in AIT. Users select the component property they wish to fault from the "Faults" list box and press the "Edit" button. The user then specifies a value using one of the two kinds of "Edit Value" interaction objects discussed previously: pop-up menus used to present discrete choices (Figure 9) or a value slider used to input system-constrained continuous values (Figure 7).

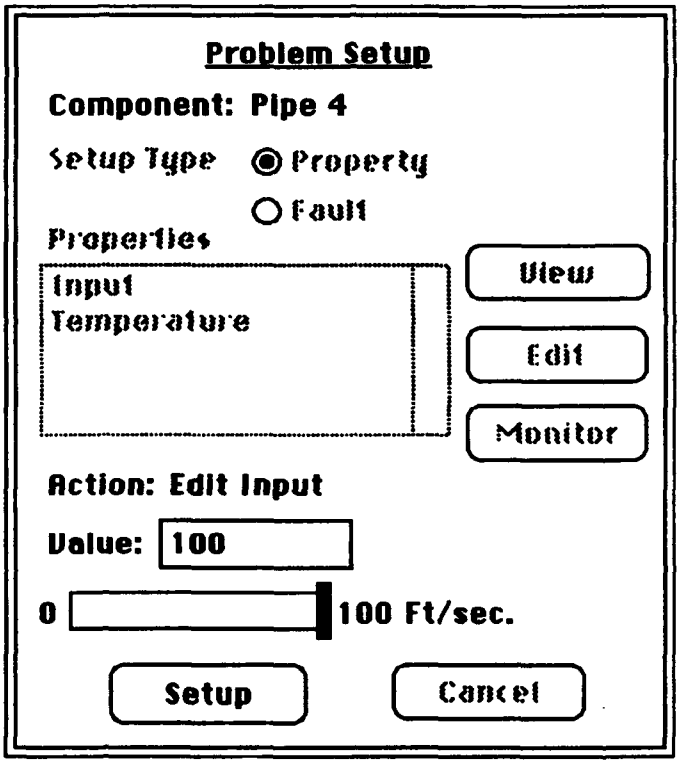

Figure 7. Problem Setup: Edit Component Property

The other type of problem setup activity supported in AIT is displaying the problem setup information for the expert to review (Figure 10). When the user selects "Display Setup" from the "Problem" menu, the "Simulation" window is presented, with all selectable objects disabled, overlaid with component property values and fault induction values currently entered.

4.3 Problem Solving

AIT supports five major activities within the context of expert problem solving (see options in pull-down menu on Figure 5). These activities include creating, recording, playing-back and deleting expert problem solutions. In addition, an expert can construct and edit a goal-action hierarchy for a recorded solution. AIT also supports four major activities within the context of student problem solving. Since these problem solving activities are identical to the activities involving experts, with the exception that the student does not construct a goal-action hierarchy, the discussions of the creating, recording, playing-back and deleting problem solutions that follow will apply to both students and experts.

In creating a new problem solution, the user simply enters a new problem solution name in the text edit field of the dialog box. This action creates a problem solution space that needs to be defined using the "Record Expert Solution" menu option.

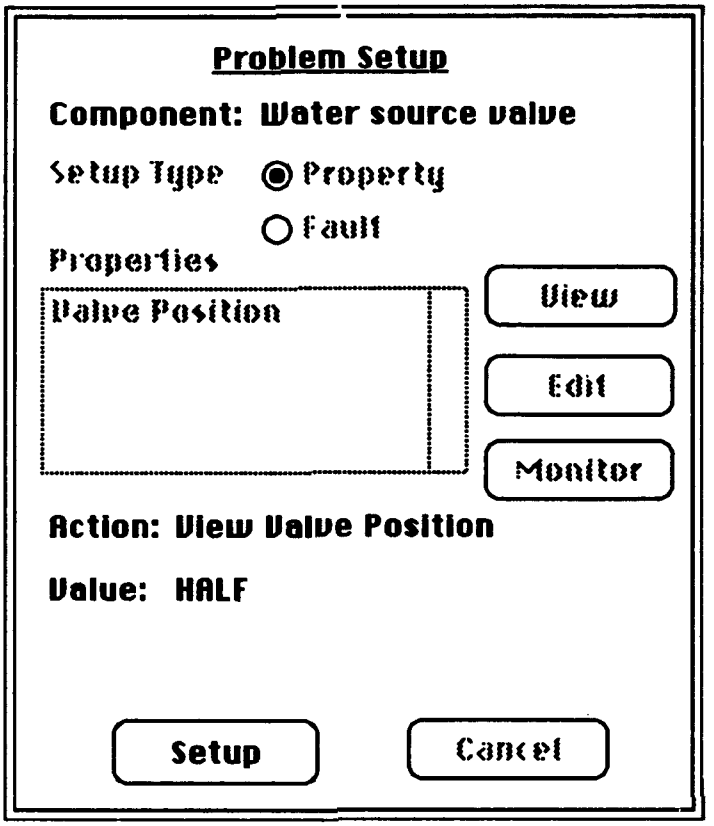

Figure 8. Problem Setup: View Component Property

When a user begins a problem solving activity, they are first presented with the window combination depicted in Figure 11. This includes the simulation window, now in "Record Solution" mode as indicated by the mode message in the lower left corner of the message box, as well as an "Action Monitor" window. The message box also contains brief instructions on what the user should do next.

The "Action Monitor" window provides a record keeping environment for the user to review during problem solving. This includes a scrollable text box containing the problem symptoms, an "Action" panel that presents and supports basic editing capabilities on user actions, and a "Hypothesis" panel that presents and supports basic editing of problem hypotheses. Action editing includes the ability to view or edit an individual action, attach an explanation to an action, move an action up or down within the list of actions, or delete an action. Each of these functions will be discussed further later in this section. Hypothesis editing includes specification of the contents of the hypothesis display in terms of hypotheses currently under suspicion, hypotheses currently ruled out, or all hypothesis, as well as the ability to select and rule out specific hypotheses. Hypothesis editing is modal. This means that users have to exit "hypothesis" mode explicitly using the "Done" button before proceeding with other problem solving activities.

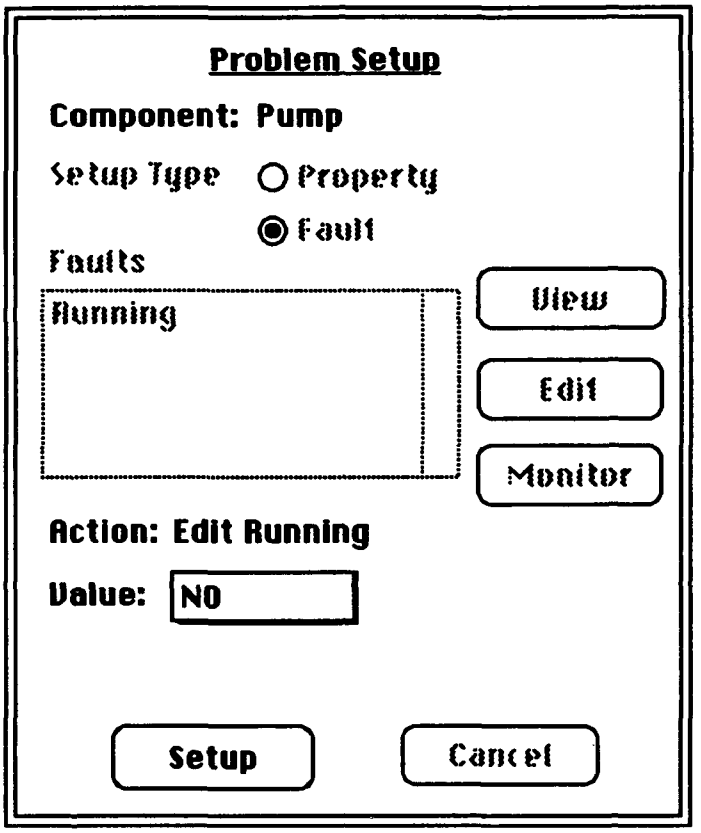

Figure 9. Problem Setup: Edit Component Fault

The first problem solving activity the user engages in is the creation of their initial problem hypothesis space. This is accomplished by clicking on an object to bring up the dialog box depicted in Figure 12. The user selects a fault hypothesis from the list box, then selects either the "Suspect" or "Rule Out" buttons, then "Exit." AIT currently uses discrete lists of problem hypotheses. However, any type of input could be used, constrained only by what the application's device knowledge will support.

Figure 13 shows a typical "Action Monitor" window after the expert has created the initial problem hypothesis space. After pressing "Done" and exiting "hypothesis" mode, users are instructed to click on an object in the simulation window. In "Record Solution" mode, clicking on an object in the simulation window will present them with a problem solution dialog box as depicted in Figure 14.

Figure 14 also shows the types of problem solving activities supported in AIT. Users can either view, edit or monitor a component property, or else they can view the results of a test on some component property. These activities are reflected in Figures 15, 16, 17 and 18, respectively. Viewing a property value is a request for the current value of that component's property. Editing a component property value is accomplished using the same methods as discussed previously: pop-up menus for discrete values and sliders for continuous values. Selecting the "Monitor" button sends a request to the system to display in continuous and dynamic fashion the property value selected in the simulation window.

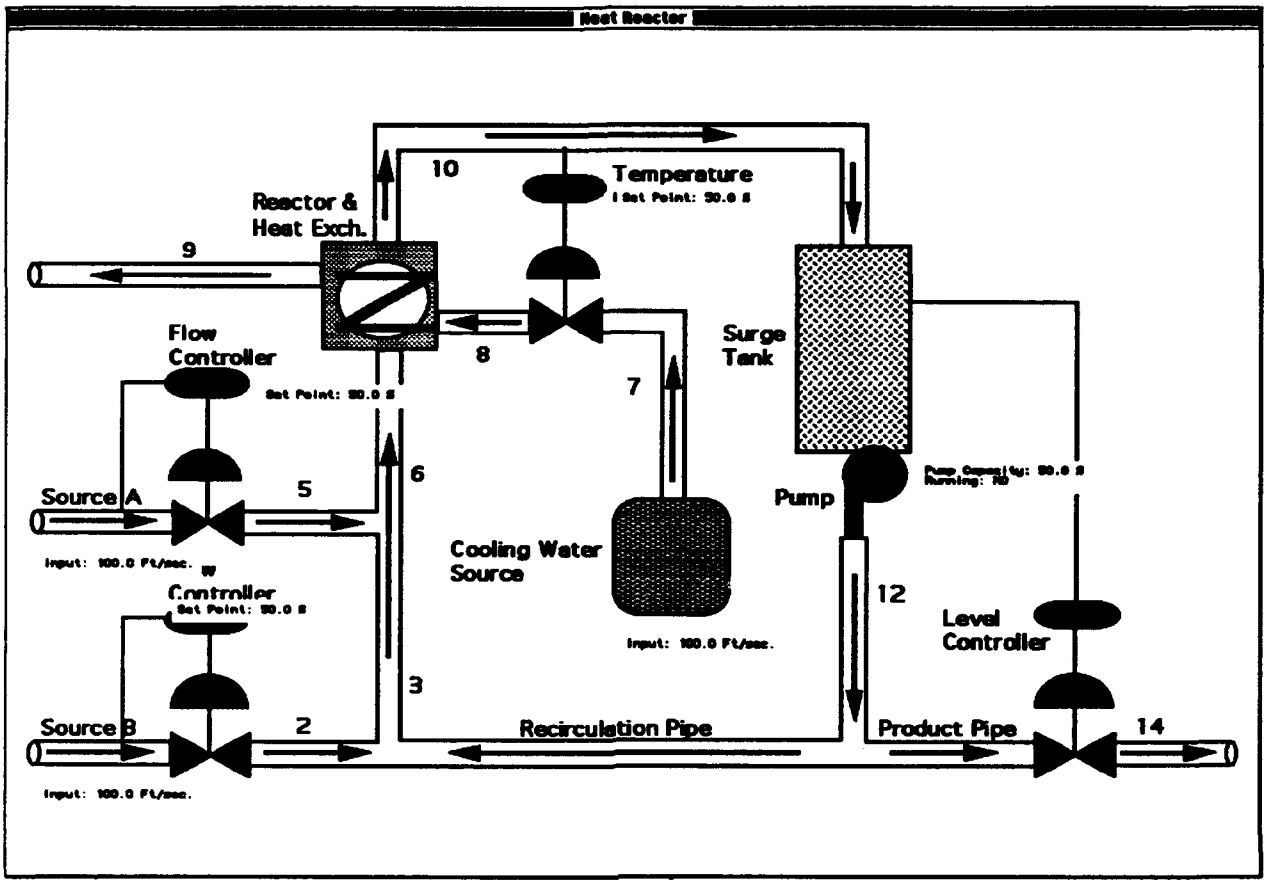

Figure 10. Problem Setup Review

Selecting the "Test" option brings up a display of possible tests for that component. The user then selects the test to be performed and presses the "View" button and AIT displays the results of that test. It is also possible to define new tests in AIT dynamically. One of the choices available in the "Options" menu in the title bar is "Define New Test." Selecting this option brings up the dialog box depicted in Figure 19. To define a new test the user first selects a component type from the "Types" list box. Depending upon the component type selected, the "legally" measurable properties of that component type are presented in the "Properties" list box, and any existing tests are presented in the "Existing Tests" list box. The user can then select a property to be tested, give the new test a name, and press the "Create" button.

This method of defining a new test is relatively unconstrained. In all likelihood, this functionality would need to be restricted by such things as available tools, practicality, time to test, etc. It was included as a demonstration of how this functionality might be implemented: the hierarchical chaining of features and properties to produce new entities.

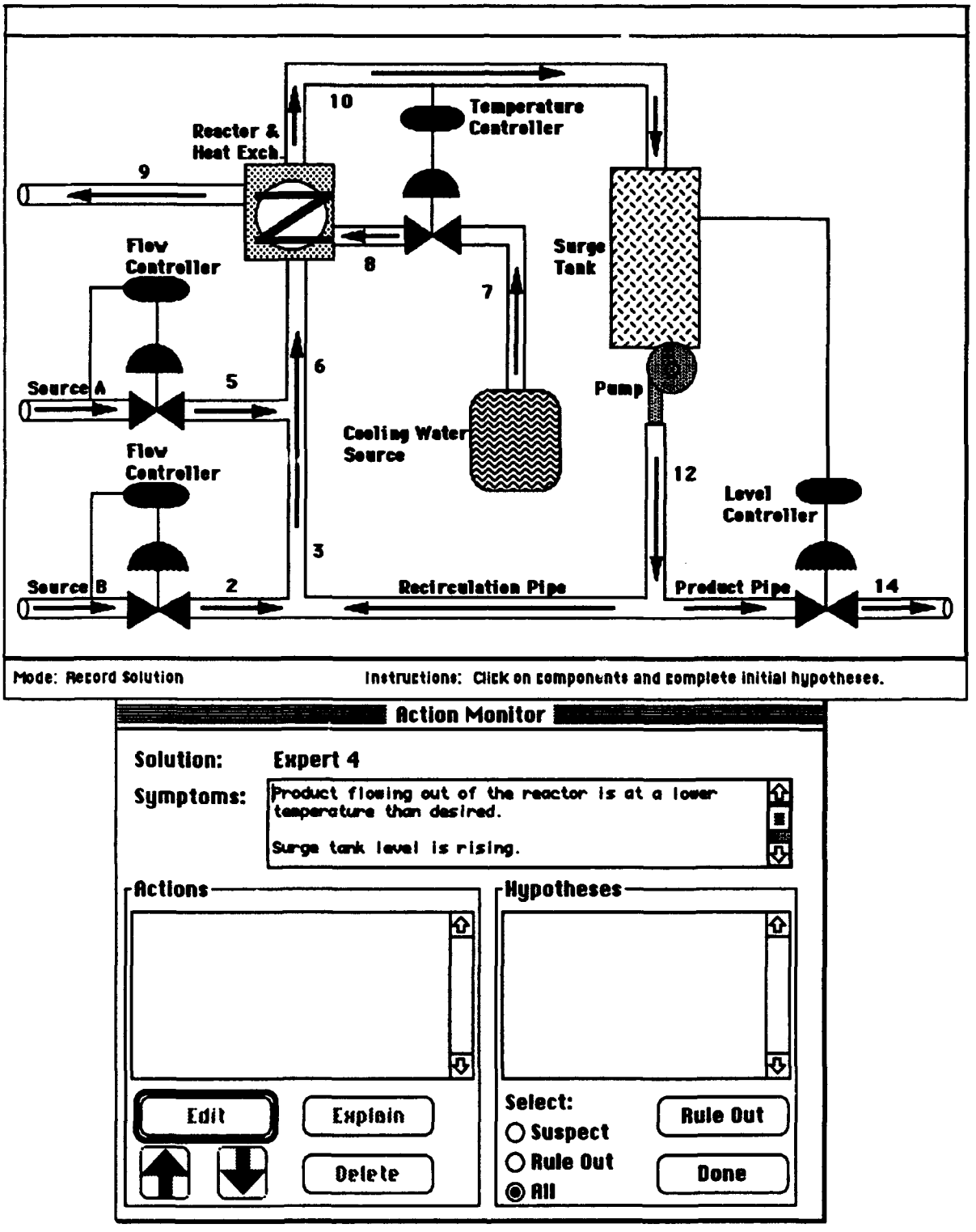

Figure 11. Problem Solution Windows

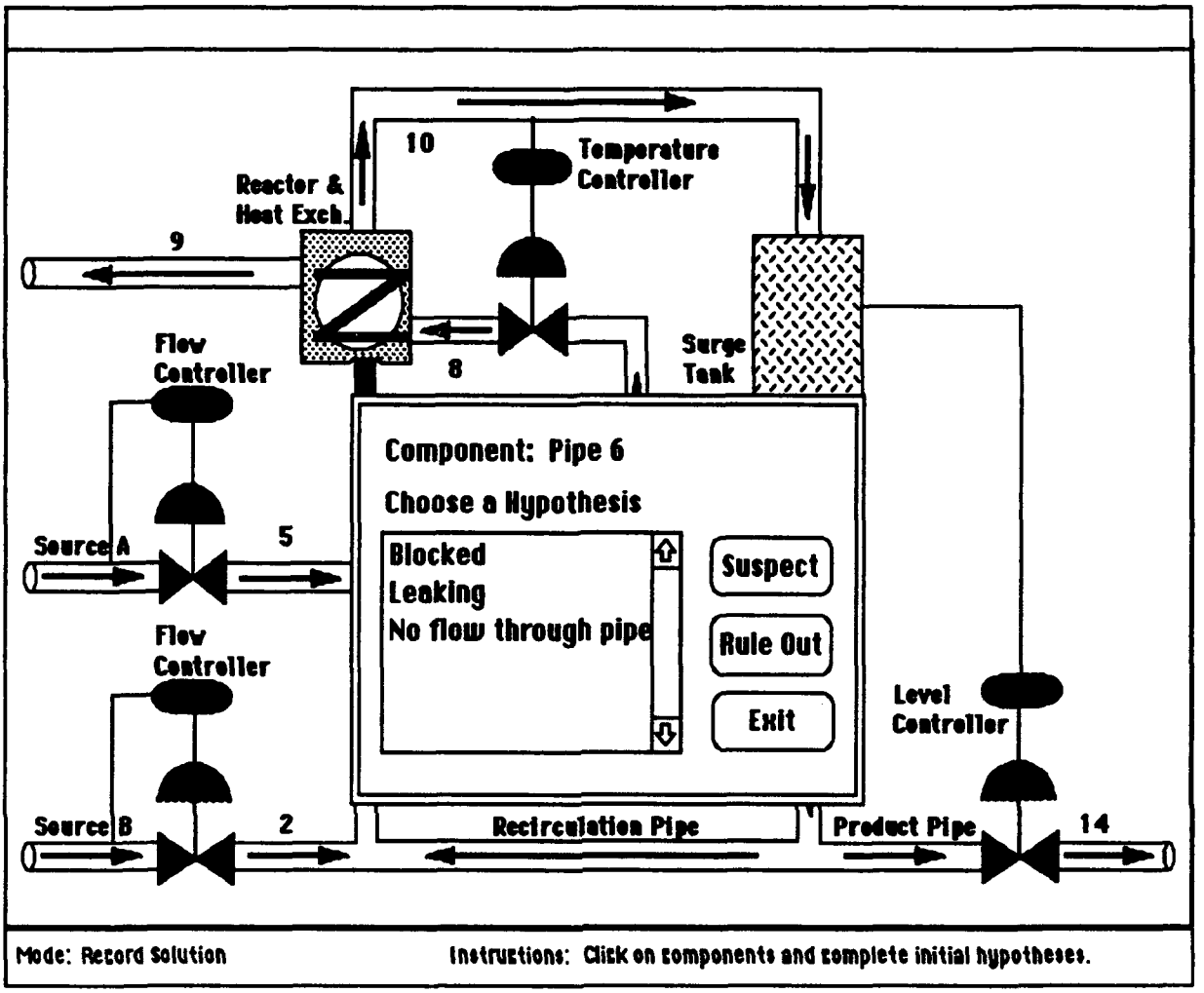

Figure 12. Hypothesis Input

The "Action Monitor" window also continually displays the actions users take in the simulation window as well as their updates to the problem hypothesis space. Figure 20 presents an example of an "Action Monitor" window following four user actions that have resulted in the ruling-out of three of the four problem hypotheses. The user can review each of these actions and either continue with the problem solving process by selecting another component or else edit an action in the list by selecting it with a single mouse click. Once selected, the user can move the action up and down in the list (by use of the up and down arrow buttons), delete the action, edit or view the action, or else attach an explanation to the action. If the type of action selected is an edit action, the user sees the "Edit" button, as depicted in Figure 20, which allows them to change the action's property value if they so desire. If the type of action selected is a view, monitor, or test activity, the "Edit" button is replaced with a "View" button, signifying that the user cannot change the value represented in the action.

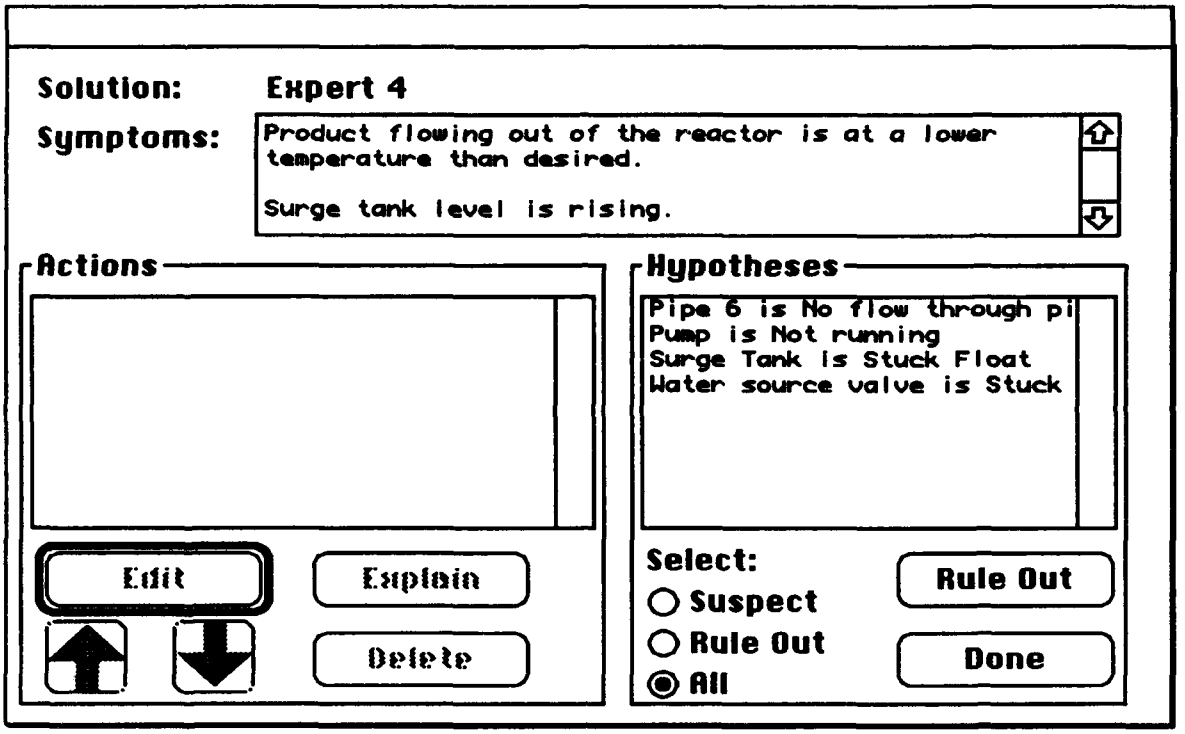

Figure 13. Problem Solution Monitor with Hypotheses

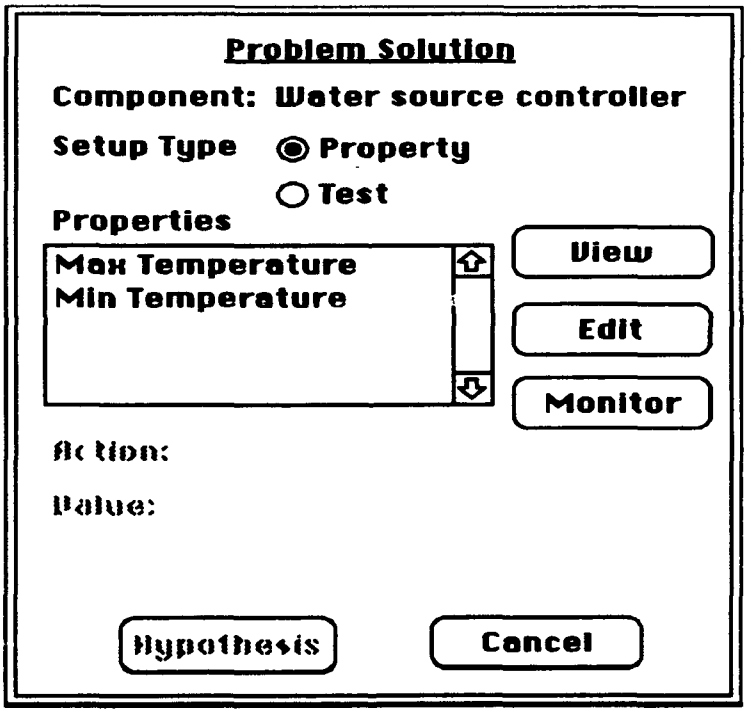

Figure 14. Problem Solution Dialog Box

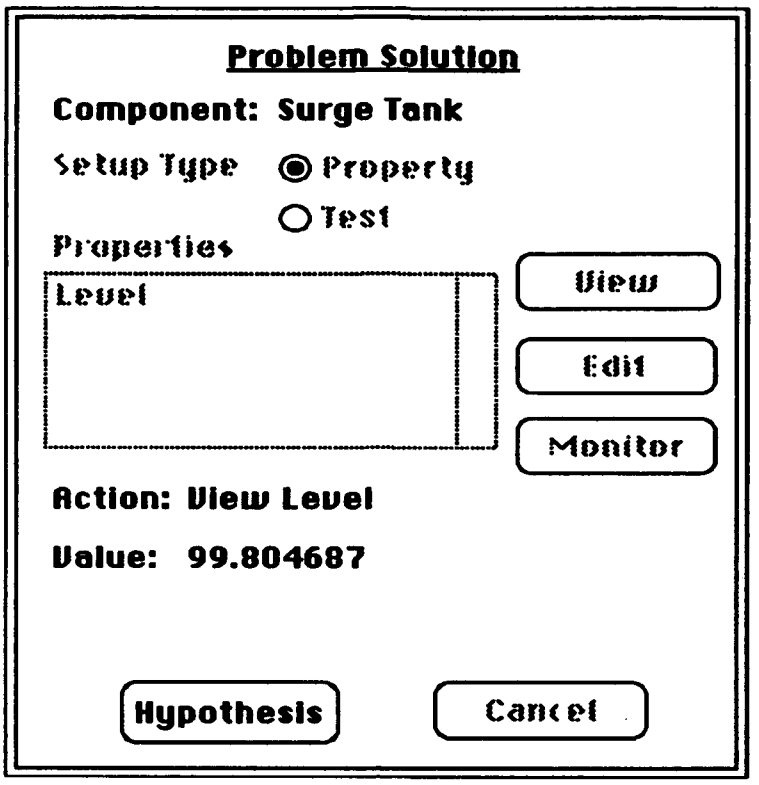

Figure 15. Problem Solution Dialog: View Component Property

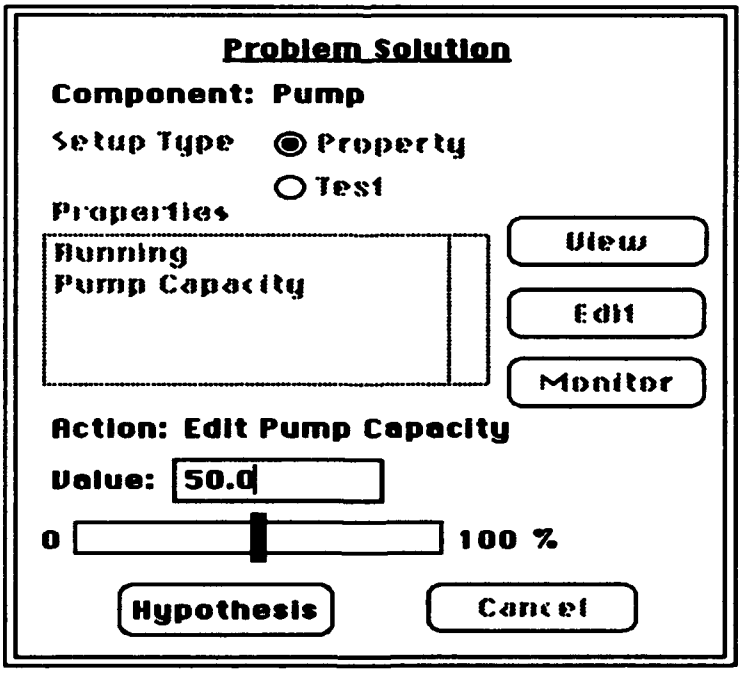

Figure 16. Problem Solution Dialog: Edit Component Property

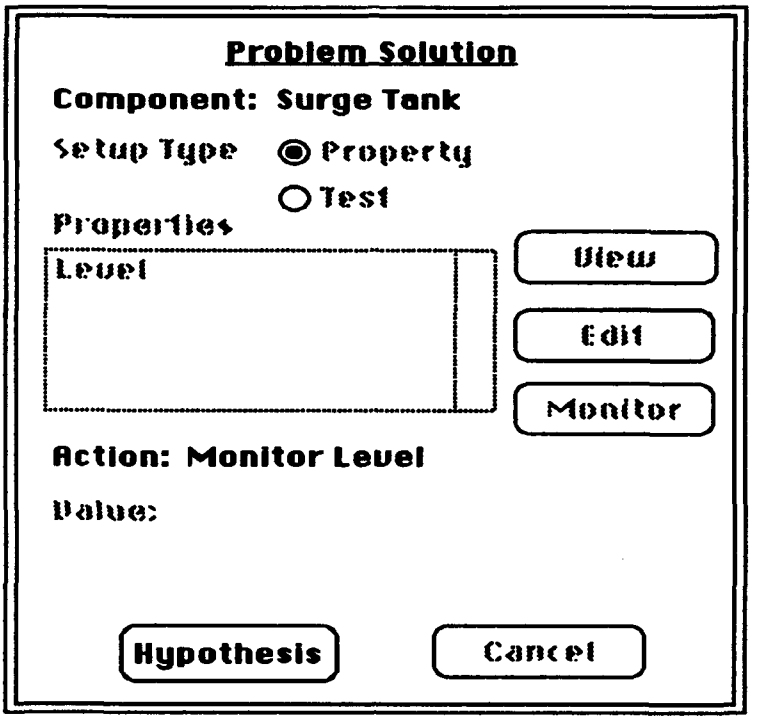

Figure 17. Problem Solution Dialog: Monitor Component Property Request

Figure 21 shows an example of the "Edit Action" dialog box opened when a user selects an editable action and presses the "Edit" button. As can be seen, values are edited using the same interaction objects discussed previously: pop-up menus for discrete choices and sliders for continuous values.

Figure 22 shows the dialog box presented when the user chooses to view an action. In this case, the value cannot be edited; only the results of the action are presented.

Figure 23 shows the dialog box presented in response to a request to attach an explanation to a specific action. The explanation is entered in free-form text in a scrolling text entry field. Explanations are saved along with their associated action for later use in building justifications for user actions.

Another problem-related activity supported in AIT is the playing back of problem solutions. Figure 24 contains the primary windows used for review of specific problem solutions: the simulation window, a playback control window, and an "Action Monitor" window customized for review purposes.

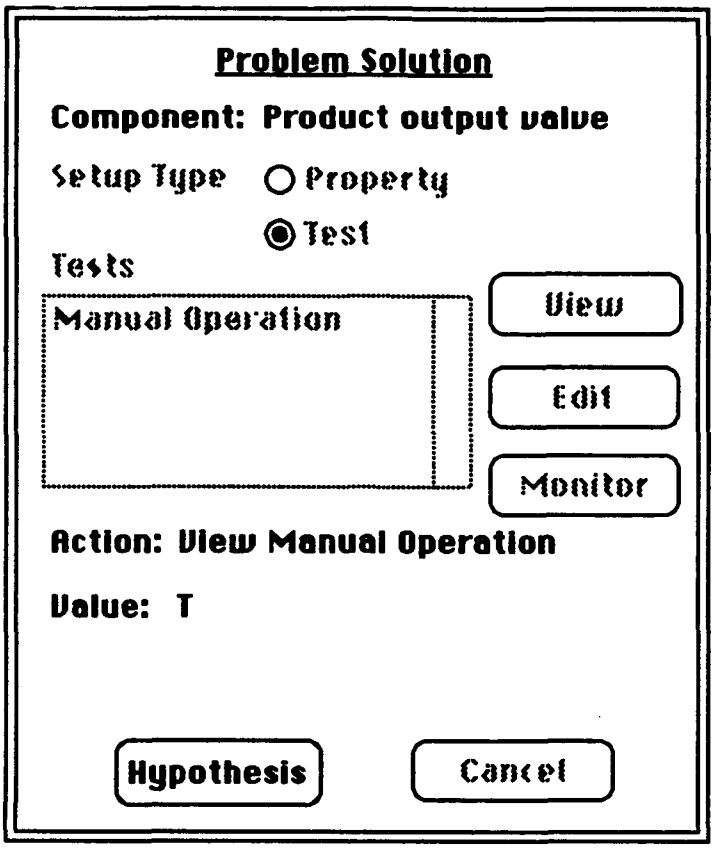

Figure 18. Problem Solution Dialog: Test Component

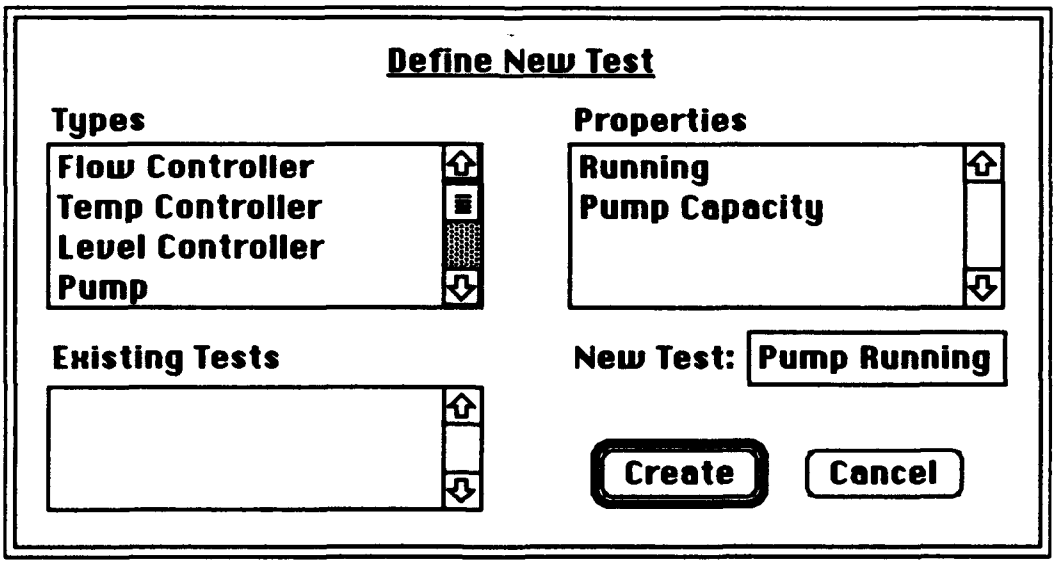

Figure 19. Define New Test Dialog Box

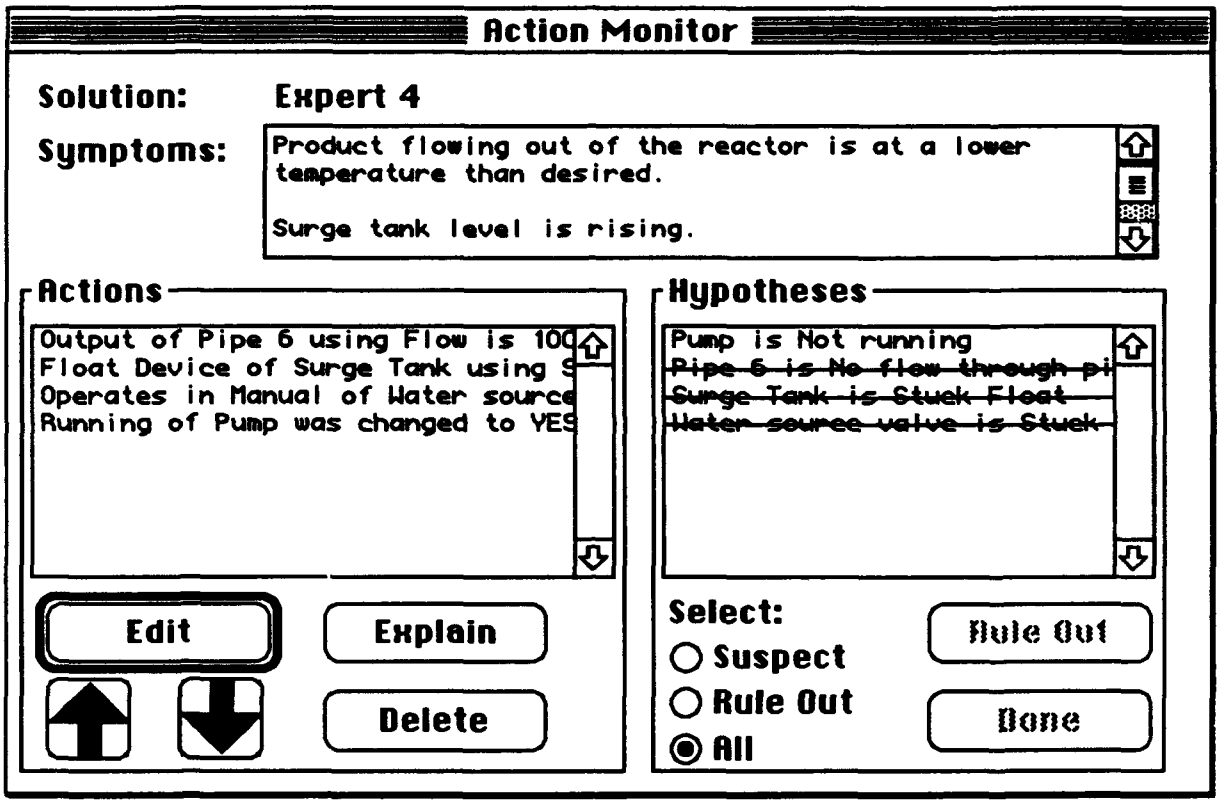

Figure 20. Problem Solution Monitor with Hypotheses Ruled-Out

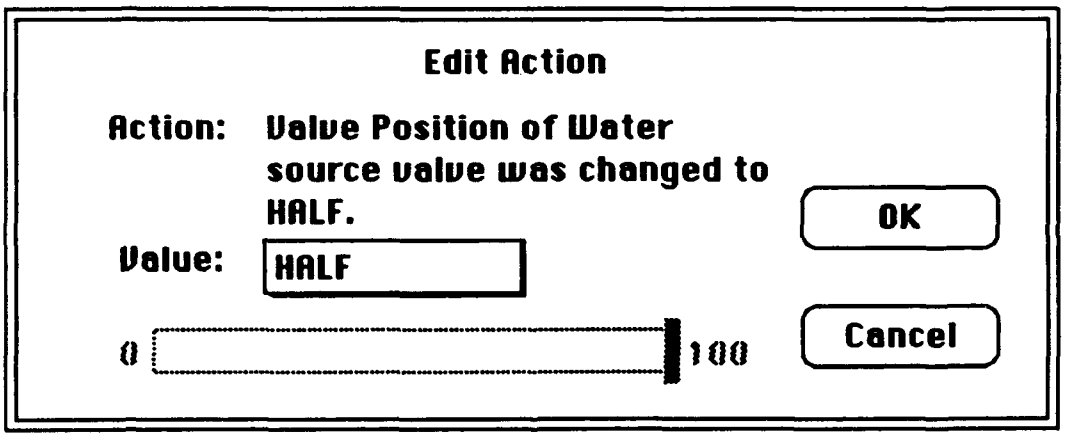

Figure 21. Editing an Action

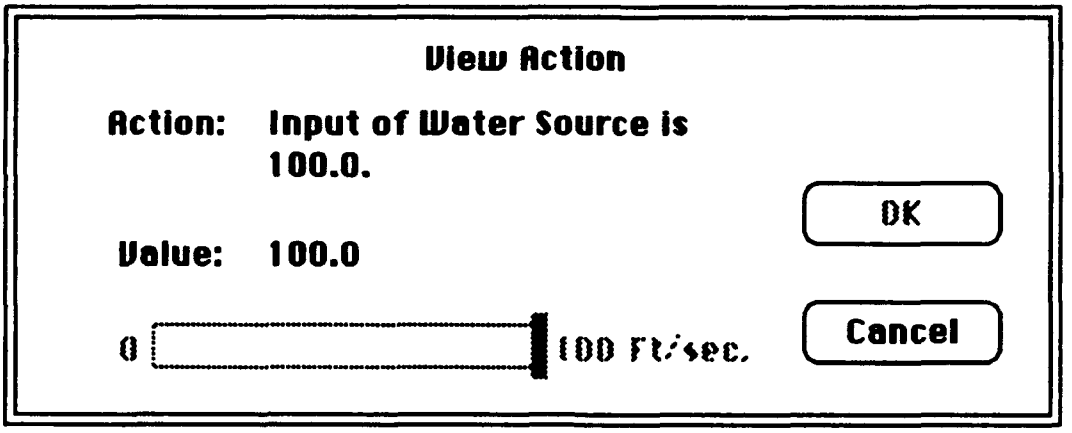

Figure 22. Viewing Action

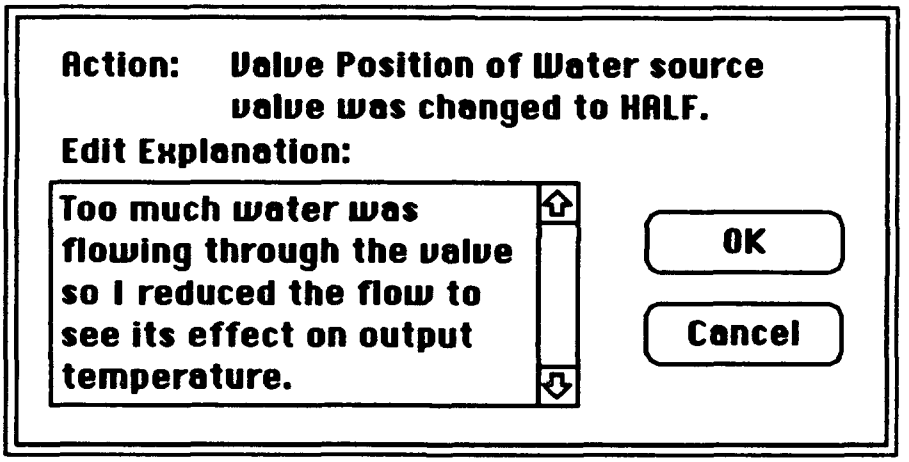

Figure 23. Attaching Explanation to Action

Playback can be executed under user control by using the "Next" button to move to each successive action, or else it can be executed under system control. Under system control, the user can set the playback speed and can still exercise some control over the playback by use of "Start," "Pause," and "Resume" buttons superimposed on top of each other in the bottom left of the "Playback Control" window. At the beginning of a playback session, the "Playback Control" window opens with the "Start" button visible and the "Pause" and "Resume" buttons disabled and invisible. Once playback has begun the "Start" button is replaced by the "Pause" button, which allows users to halt the playback temporarily. Pressing "Pause" causes the "Pause" button to be replaced by the "Resume" button, which resumes the playback from the point at which it was stopped. The "Reset" button returns the playback to the first action.

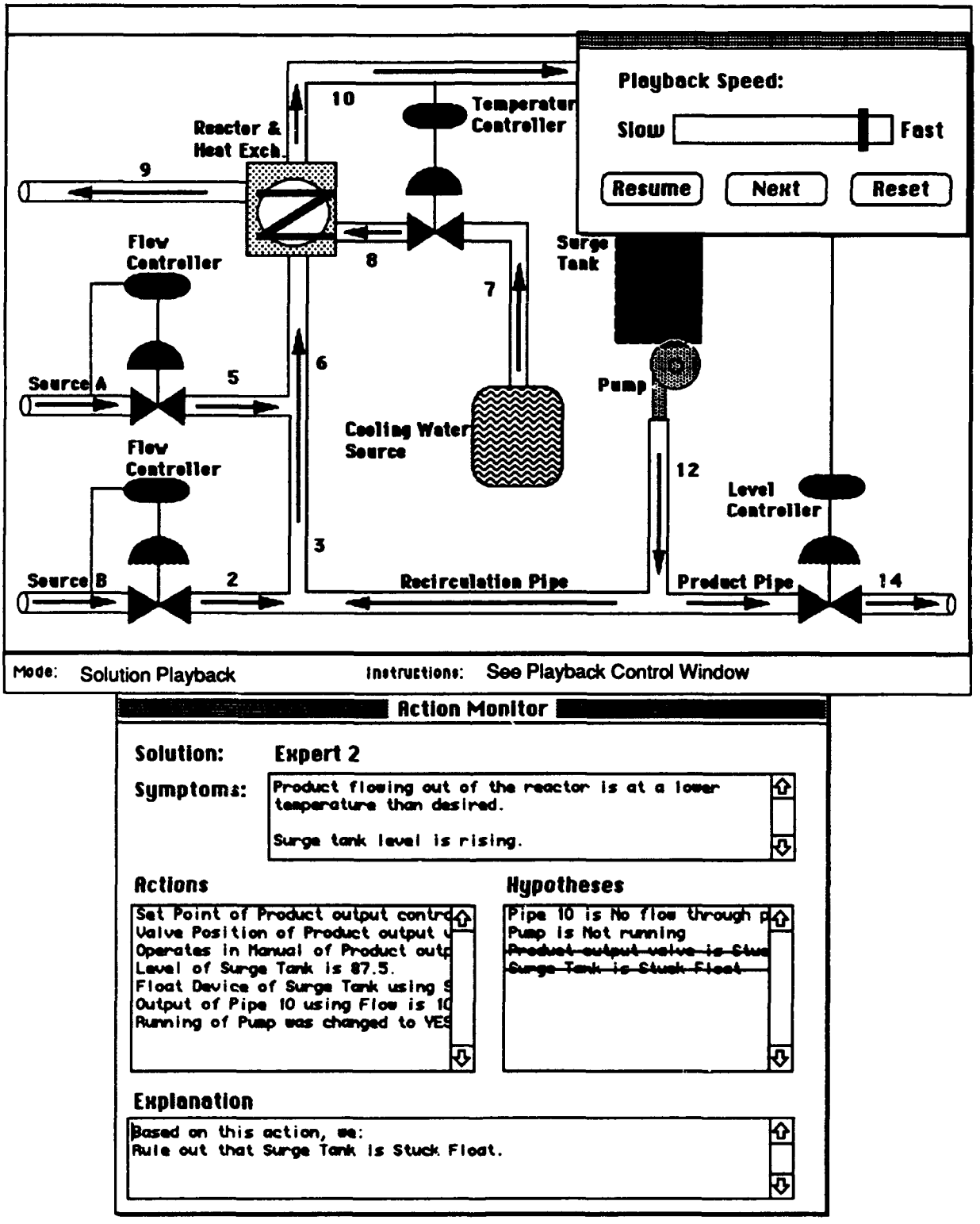

Figure 24. Problem Solution Playback

During the playback process, the first action in the action list is highlighted, as is the object representation of the component referred to in the action in the "Simulation" window. In addition, AIT presents the hypothesis set configuration following that action. The "Explanation" field contains the justification for that action. As described previously, the explanation field can contain two components: (1) any comments entered by the expert during action editing, and (2) differences in the problem hypothesis space from the previous action (or set of actions) to the present action. In Figure 24 the action taken was a test of the float device in the surge tank (note that the surge tank object in the simulation window is highlighted; unfortunately the highlighting of the action in the action field did not reproduce in Figure 24). The result of that test indicated the float device was not stuck, so the expert ruled out the hypothesis that the problem was due to a stuck float.

4.4 Goal Tree Construction and Editing

The final AIT problem solving activity that experts engage in is the construction and editing of their problem solving goal hierarchy. This process is illustrated in Figures 25 through 28. Goal tree construction and editing starts with the expert's actions being represented as an action list in the "Goal Editor" window (the lower window in Figure 25) and as graphical objects (with terse text descriptions) in the lower panel of the "Goal Tree" window (the top window in Figure 25). The user must then create the subgoals and goals for that problem solution by pressing the appropriate "Add Goal" or "Add Subgoal" button. Pressing those buttons opens a dialog box into which experts can enter goal or subgoal titles (see Figure 25). Goals and subgoals can also be deleted by selecting them from the list and pressing the "Delete Goal" or "Delete Subgoal" button. Lists of actions, subgoals and goals can be selected by holding down the shift key while selecting. Disjoint selections are made by holding down the command key during selection.

Goals and subgoals are converted into graphical objects dynamically drawn in the upper panel of the "Goal Tree" window as they are created. Figure 26 shows a situation where one goal and three subgoals have been created. Once created, the expert then links actions to subgoals and subgoals to goals by selecting the entries from two adjacent lists (i.e., goal and subgoal or subgoal and action) and pressing the corresponding "Set" button. Figures 27 and 28 contain goal trees for which the goal-subgoal, and subgoal-action links have been created, respectively. It is important to note that any action can be linked to any number of subgoals, and any subgoal can be linked to any number of goals.

The "Show" buttons are used to highlight links in the "Goal Tree Editor" window as follows: Experts select any goal or subgoal in the editor lists. Pressing the corresponding "Show" button will highlight the links between the selected item and the information in the list on the right side of the "Show" button.

The current implementation of goal tree construction was included for exploratory purposes. As such, the capability does have some functional limitations. The goal tree has no functionality to compress actions or scroll the window. Due to this limitation, only ten actions can be displayed in the bottom panel of the "Goal Tree" window regardless of how many actions there are in the problem solving record. In addition, the goal editor can only handle one expert solution at a time. Additional functionality will be required to address these two limitations and enable this capability to be evaluated as a vehicle for combining distributed sources of expertise by way of their goal trees.

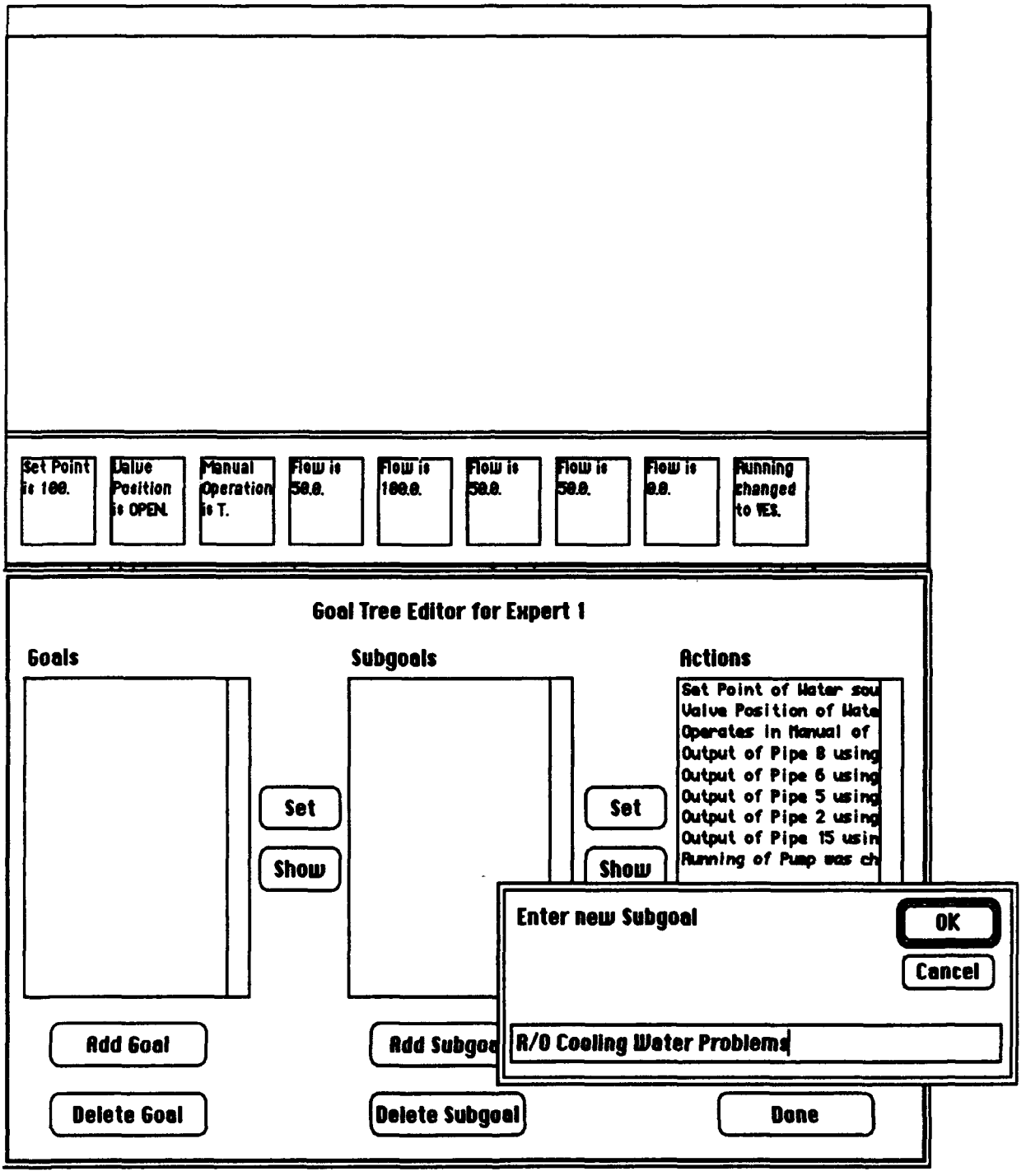

Figure 25. Goal Hierarchy Construction: Initial Goal Specification

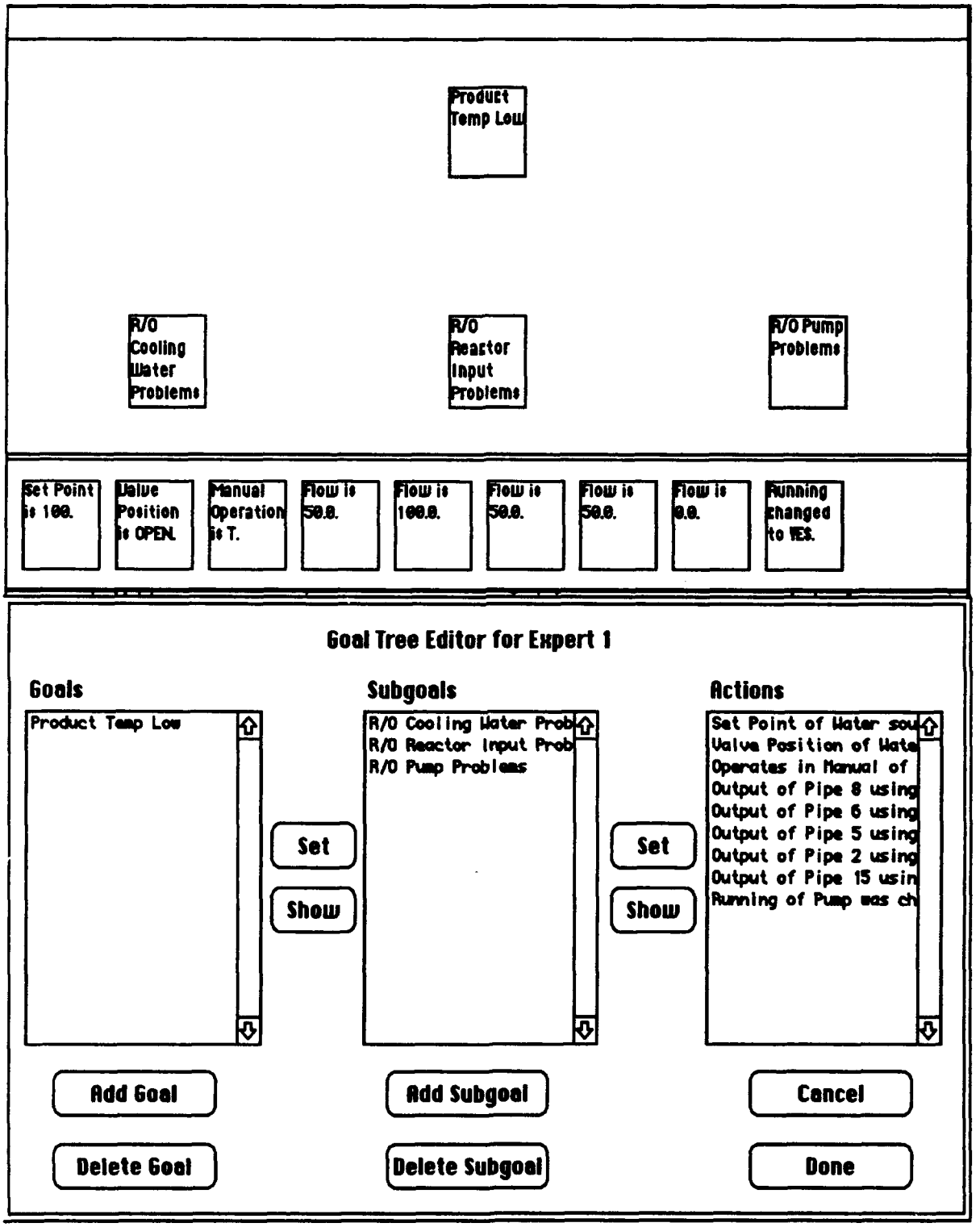

Figure 26. Goal Hierarchy Construction: Goals and Subgoals

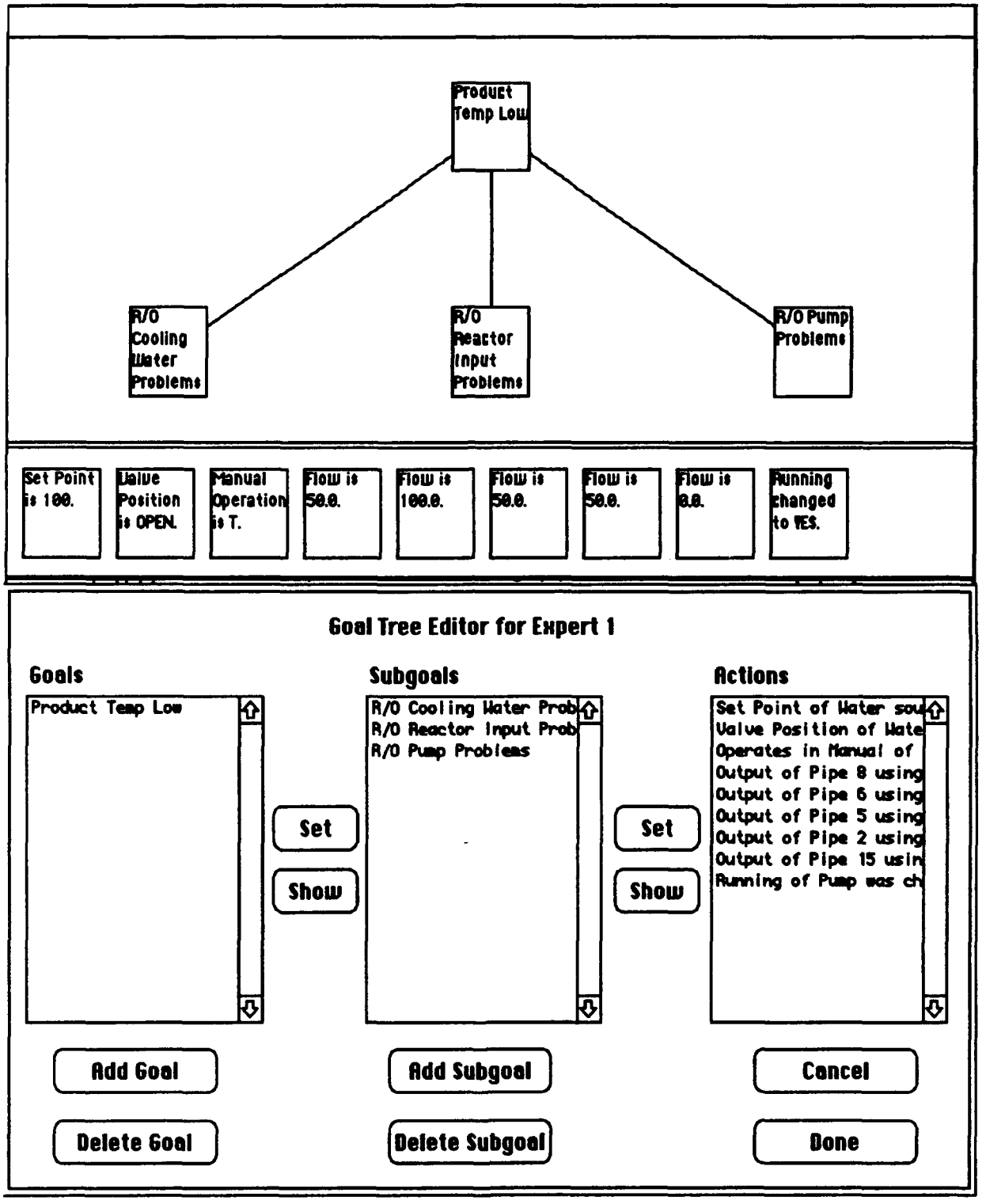

Figure 27. Goal Hierarchy Construction: Goal-Subgoal Linking

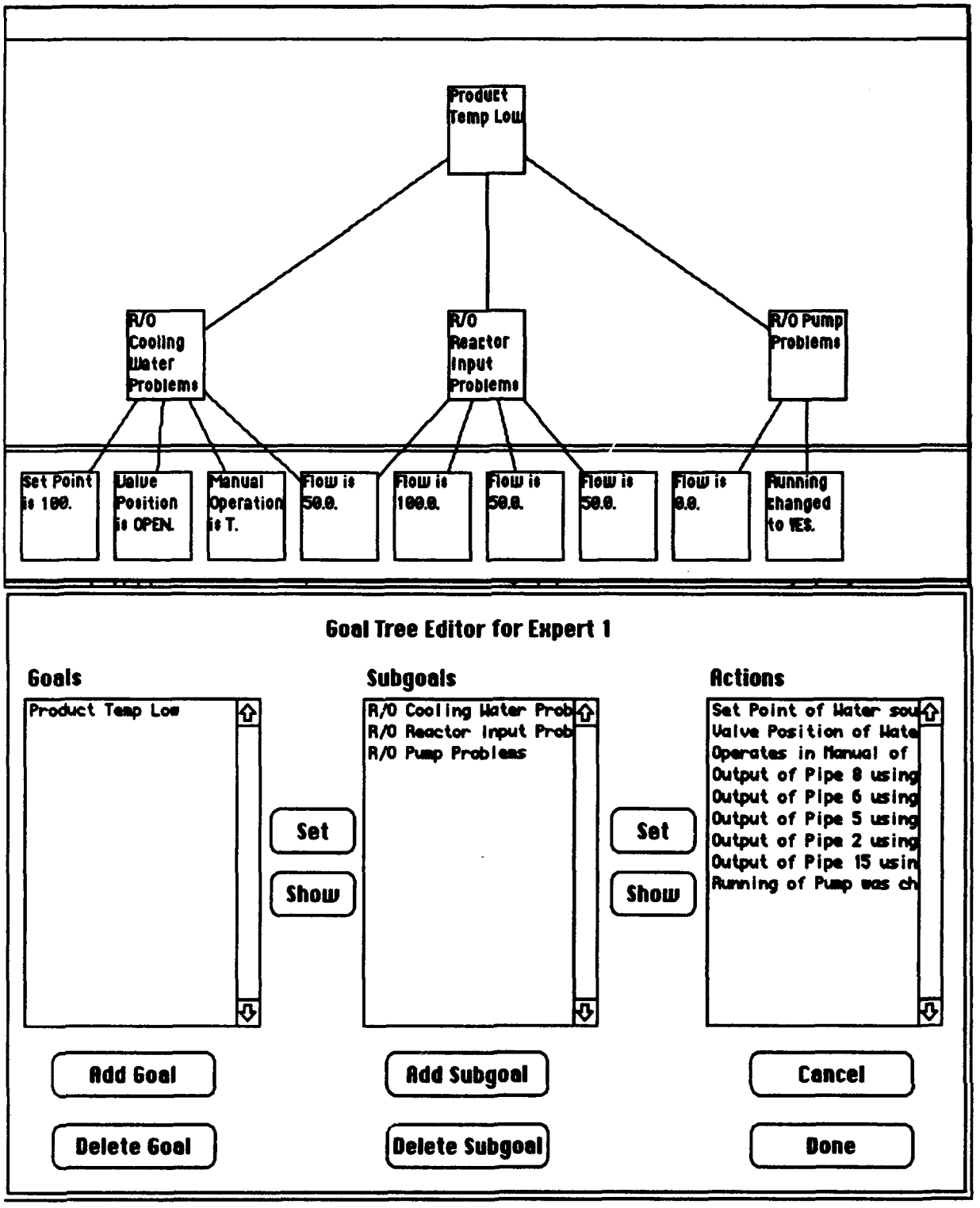

Figure 28. Goal Hierarchy Construction: Subgoal/Action Linking

4.5 Student Instruction

Figure 29 illustrates the possible "Instructor" actions supported in AIT. These activities are organized around those activities that support the instructor's understanding of the specific problem symptoms, setup and solution, and those related to the instruction of students in specific problem solving activities. Reviewing problem symptoms, problem setup, and playing back expert problem solutions access the same AIT functionality described in the scenarios on problem setup and solution. The additional functionality provided in this menu involves support for the instructor in tutorial interactions with "pseudo students" created in AIT. Pointing to the "Instruct Student" option in the menu causes a list of "pseudo students" to appear to the right of the arrow in an additional selectable list, if students have been created. One merely moves the cursor over one of the students and releases the mouse button to select one for an instructional session.

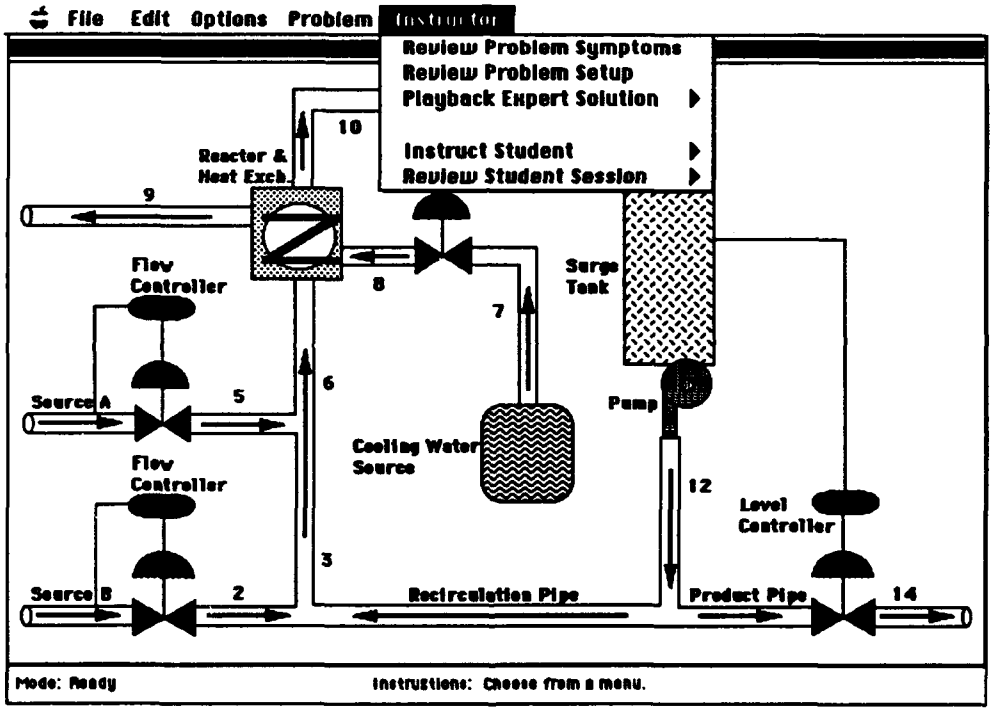

Figure 29. Instructor Menu Selections

Once a student has been selected, the instructor is presented with the window configuration displayed in Figures 30 and 31. The window in the upper portion of the figures contains a dynamic, graphical display of the relationship between student actions and expert goals. The expert goals are displayed in the top panel of the window, while the student actions are presented, and linked to expert goals if appropriate, in the bottom panel of the window. In the examples in Figures 30 and 31, the upper window contains three separate goal trees for dealing with the same problem derived from three different expert solutions. The "Student-Instructor Session" dialog box in the lower portion of Figures 30 and 31 contains the means by which instructors control the instructional session, review student hypotheses, state their observations, and recommend interventions.

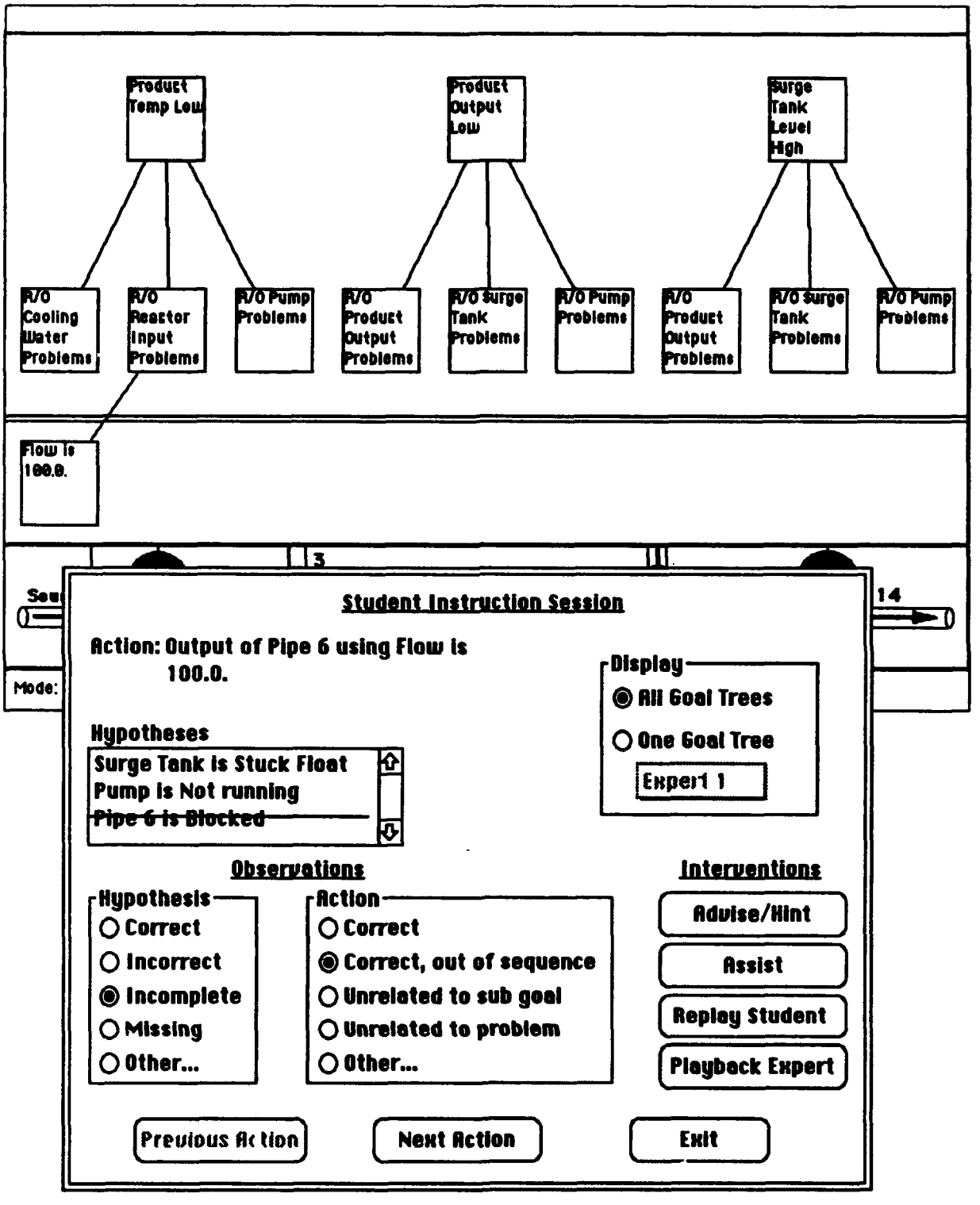

Figure 30. Instructor Intervention Windows—1

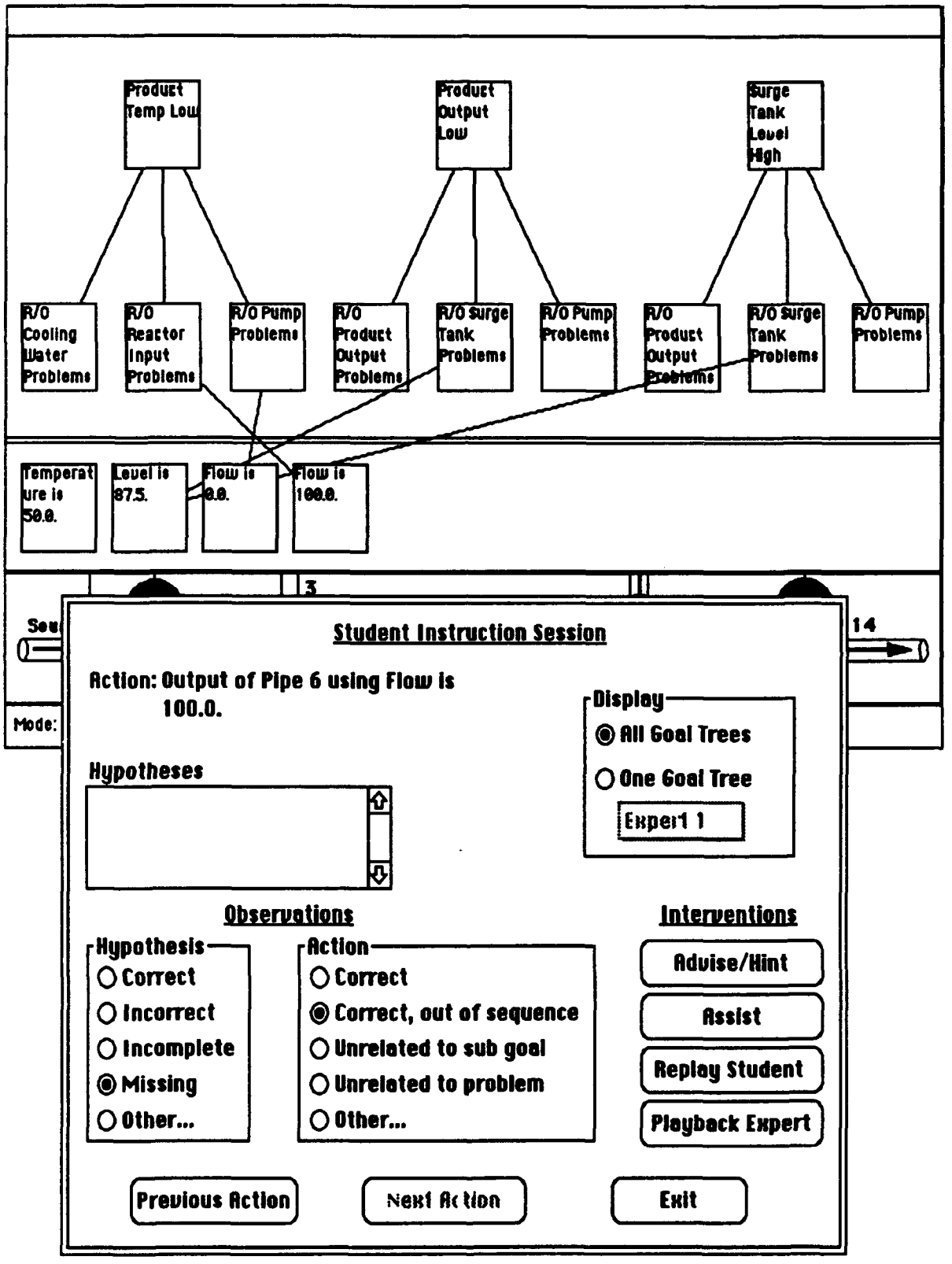

Figure 31. Instructor Intervention Windows—2

It is important to note several features of this functionality that are exploratory in nature. First, we used previous work in the area of Operator Function Modeling (Jones, Rubin, and Mitchell, in press; Mitchell and Miller 1986) to guide the representation of the goal tree. However, we were unable to find any previous work or empirical evidence to guide our metaphor for displaying single or multiple goal trees. As such, we decided to enable two different approaches of goal tree presentation, that is, all trees or a single, selected tree displayed, for the purpose of soliciting comments regarding the efficacy of the two approaches. Second, we found support from the literature on expertise (Clancey, 1985) and perceptual learning and pattern recognition (Duda and Hart, 1973) for the notion that expert teachers can identify situations requiring intervention based on their ability to "visually" recognize instances of poor performance. As such, the implementation is an attempt to explore (1) what information from student and expert problem solving activities to display to the instructor and how to display that information to facilitate the instructor's recognition of a situation requiring intervention and (2) how to provide instructors with a facility for recording those observations in as easy a manner as possible. We have also displayed a group of instructional intervention buttons derived from Table 3 on page 15. Figure 32 shows an instructor creating some advice to present to the student.

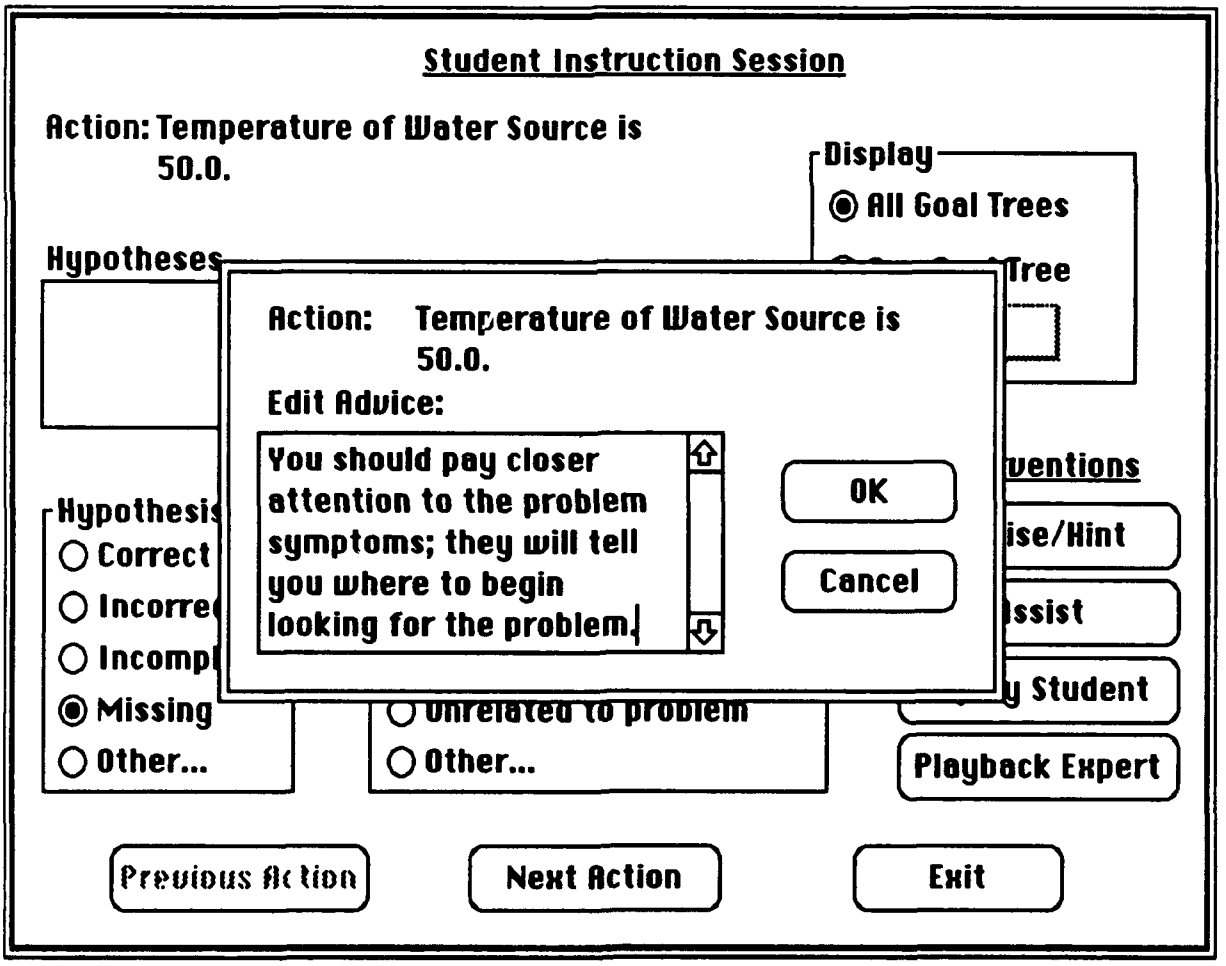

Figure 32. Instructor Intervention: Edit Advice

Given that this is an initial exploration of these issues, we believe extensive research will be required to evaluate not only the efficacy of these approaches but also to help identify the particular types of observations that real expert instructors use. In the current implementation, the instructor can state their observations and recommend a type of intervention after each student action. The rationale behind this procedure was to identify a method of collecting a large amount of data on the when, why and what of dynamic instructional interventions. "When" information is captured in the specific student actions to expert goals configuration. "Why" information is captured in the instructor's observations about the student's problem hypothesis space and their last problem solving action taken. "What" information is contained in the type of instruction recommended. Theoretically, this data could then be abstracted into general instructional heuristics for a particular problem domain.

The final "Instructor" activity supported in AIT is "Review Student Session." In this activity, the instructor is given the opportunity to review an instructional session by providing them with a dynamic expert-goal-student-action window, and an "Intervention Summary" window as depicted in Figure 33. Unfortunately, there was not sufficient time to implement the capability to edit a student session.

4.6 Session Shutdown

The final scenario involves saving the current session, if desired, and shutting down the AIT application. AIT has the ability to save all problem setup, expert solutions and goal trees, student solutions, and student-instructor interactions attached to the specific problem created or worked on to a separate file. At present there is no limit to the number of problems that can be created, hence there is no limit to the number of problem setups, solutions and interactions as well. To save the information created or edited during that session, pull down the "File" menu and select the "Save" option as depicted in Figure 34. To exit AIT once the information has been saved, pull down the "File" menu and select the "Quit" option. If you quit before saving, all information input during that session will be lost.

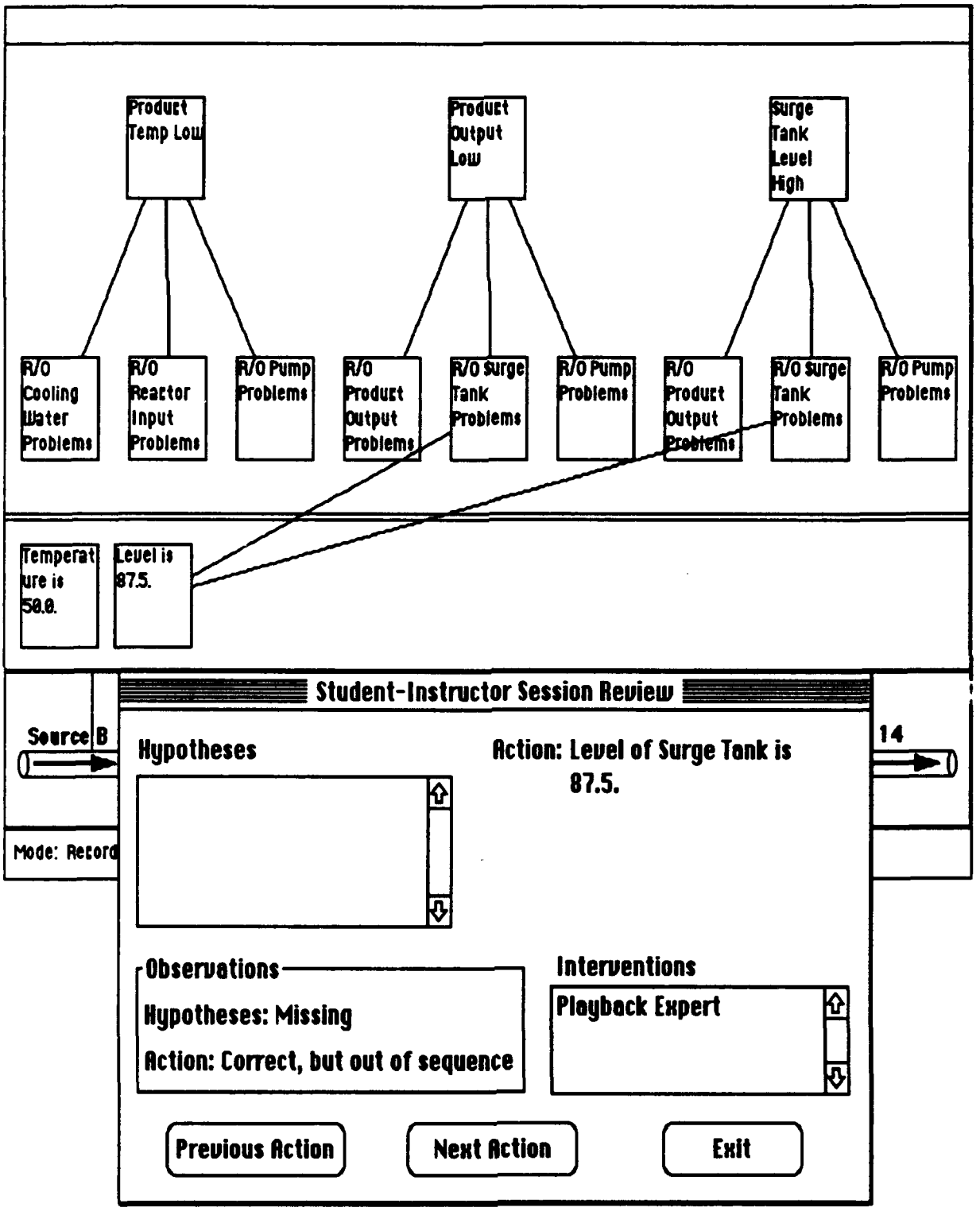

Figure 33. Student-Instructor Session Review Windows

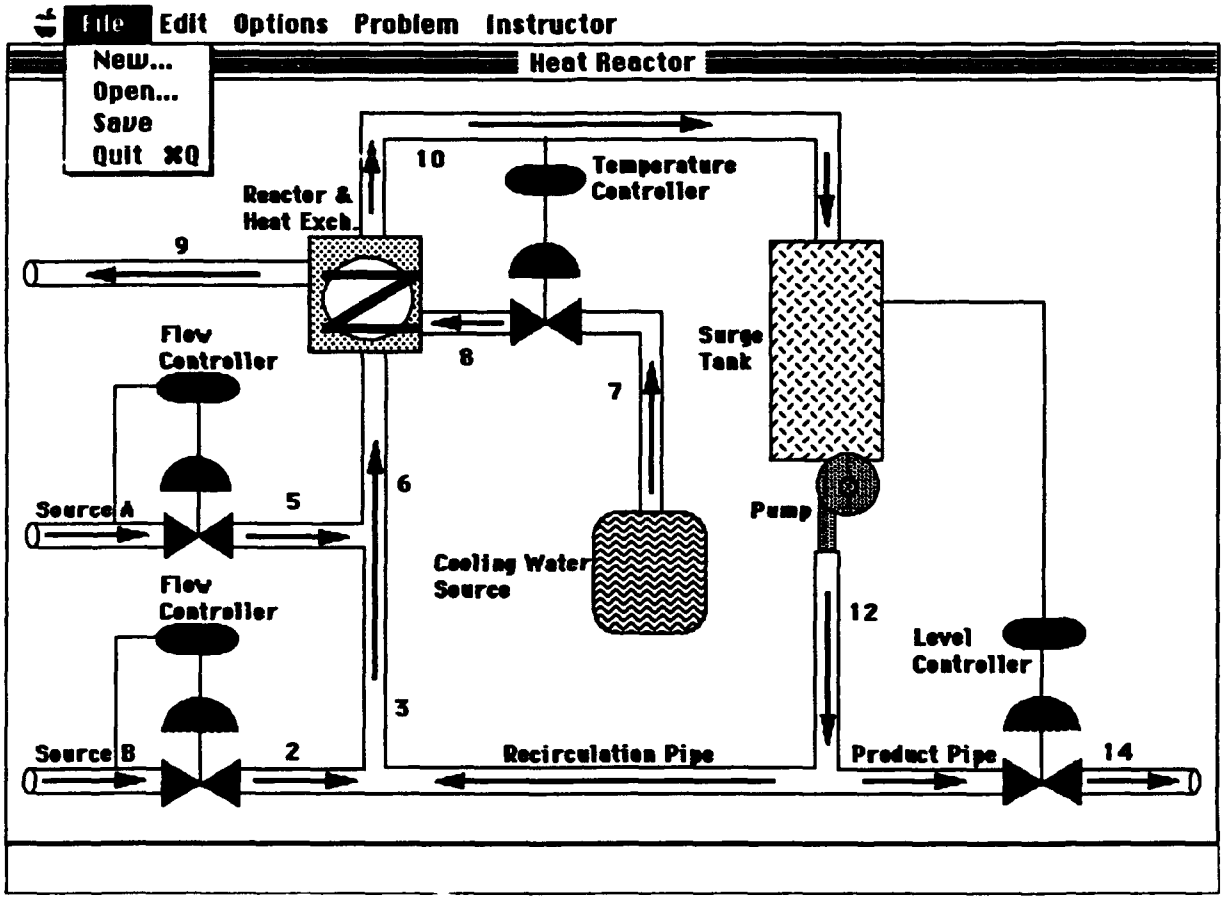

Figure 34. File Menu Options

Section 5
Conclusions

The current research effort has identified a number of conclusions regarding the requirements of a knowledge acquisition tool for instructional systems.

First, the training and performance of troubleshooting activities in complex systems appears to require the integration of at least three different types of knowledge that will likely come from different sources: (1) knowledge about systems and their normal and abnormal behavior ("how it works"), (2) knowledge about interacting with these systems to perform a job or task ("how to use it"), and (3) knowledge about how to teach students to perform a job or task ("how to teach it").

Second, it appears likely that the acquisition of these types of knowledge for use in instructional systems can be facilitated by developing a knowledge acquisition tool that has some or all of the following characteristics:

- Knowledge acquisition should take place in a simulated environment in which users interact directly with a representation of their domain that emulates their real world interactions and promotes the solicitation of natural and instinctive responses through a direct manipulation interface.

- The knowledge acquisition tool should employ an action-based solicitation method in which all user actions in the simulated environment as well as their dynamic problem hypothesis space are recorded in sequence and in "pseudo real time."

- The knowledge acquisition tool should support knowledge integration by providing users with a glass box architecture with multiple, animated views of the knowledge that are completely introspective and can be manipulated.

- The knowledge acquisition tool should support knowledge construction by building representations of strategic knowledge, such as justifications, from observations of user actions.

Two examples of justification construction employed in the present system include 1) constructing justifications for the actions taken by expert problem solvers from differences in their dynamic problem hypothesis space from one action (or set of actions) to another, and 2) constructing justifications for the interventions initiated by instructors from a "snapshot" of the configuration of student actions to expert goals and subgoals at the time the intervention was initiated, and from observations by the instructor, obtained and recorded in real time, of the student's actions taken and their problem hypothesis space.

Section 6
Recommendations

The results of the present research effort indicate that there are a number of issues that either deserve or will require additional research. First, the entire area of the acquisition of instructional heuristics is relatively new and unexplored. The present research effort represents an attempt to "scratch the surface" of this problem by helping to identify and raise additional research questions to be addressed. These additional questions include: How do expert instructors recognize that a student is in need of some type of intervention and what clues do they use to determine the type of intervention to initiate? What is the scope of the type of interventions used by expert instructors? How do you display information from student and expert problem solving activities to facilitate recognition by the instructor that an intervention is required? What information makes up instructional heuristics and how do you go about acquiring that information in a natural and unobtrusive way? And most importantly, is it possible, practical, and useful to aggregate instructional expertise from specific instructional situations to general instructional heuristics?

Another issue has to do with justification construction. In the current prototype we have implemented an approach to constructing justifications based on the theory proposed by Gruber (1991). However, the question must be raised as to just how effective this approach is in capturing the true essence of expertise. Is there additional information that needs to be included in the justification and are there unobtrusive ways of acquiring this information through natural interactions with a domain representation?

Still another issue has to do with the use and implementation of goal trees to represent higher level problem solving expertise. Are goal trees an effective representation for the teaching of problem solving expertise? What is the best way to construct and present goal trees? Can the goal tree metaphor be used to facilitate the integration of distributed sources of expertise?

In addition to the global research questions, it is recommended that further work be done on the AIT prototype in a number of areas. First, the AIT functionality should be slightly expanded to allow for its use in empirical investigations of some of the approaches and interface methods implemented. This will involve implementing the ability to enable different approaches for accomplishing the same objective selectively (e.g., display either all goal trees or a single goal tree, or use pull-down menus or icon palettes) as well as expanding some of the current capabilities from demonstration to full functionality (e.g., the ability to employ more than one process or the ability to concatenate actions in the goal tree display). In addition, additional functionality should be incorporated in AIT, such as the ability to acquire instructional plan information or device knowledge as well as the implementation of any new approaches developed in the interim.

Section 7
References

Anderson, J.R. (1988). The expert module. In M. Polson and J. Richardson (eds.), *Foundations of Intelligent Tutoring Systems*. Hillsdale, NJ: Lawrence Erlbaum Associates (LEA).

Anderson, J.R., Greeno, J.G., Kline, P.J., and Neves, D.M. (1981). Acquisition of problem solving skill. In J.R. Anderson (ed.), *Cognitive Skills and Their Acquisition*. Hillsdale, NJ: LEA.

Berry, D.C. (1987). The problem of implicit knowledge. *Expert Systems*, 4, 144-151.

Bloom, C.P., Bullemer, P.T., and Cochran, E.L. (1989). Expert systems, intelligent tutoring systems and utility operations: A future scenario. *Proceedings of the Electric Power Research Institute's Advanced Computer Technology for the Power Industry Conference*, Scottsdale, AZ.

Bonar, J., Cunningham, R., and Schultz, J. (1986). An object-oriented architecture for intelligent tutoring systems. *Proceedings of OOPSLA-86*.

Boose, J.H., Bradshaw, J.M., Kitto, C.M., and Shema, D.B. (1989). From ETS to AQUINAS: Six years of knowledge acquisition tool development. *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada.

Burton, R.R. (1988). The environment module of intelligent tutoring systems. In M.C. Polson and J.J. Richardson (eds.), *Foundations of Intelligent Tutoring Systems*. Hillsdale, NJ: LEA.

Clancey, W.J. (1984). Methodology for building an intelligent tutoring system. In W. Kintsch, J.R. Miller and P.G. Polson (eds.), *Method and Tactics in Cognitive Science*. Hillsdale, NJ: LEA.

Clancey, W.J. (1985). Heuristic classification, *Artificial Intelligence*, 27, 289-350.

Cleaves, D.A. (1987). Cognitive biases and corrective techniques: Proposals for improving elicitation procedures for knowledge-based systems. *International Journal of Man-Machine Sciences*, 27, 155-166.

Cochran, E.L., Bloom, C.P., and Bullemer, P.T. (1990). Increasing the end-user acceptance of expert systems by using multiple experts: Case studies in knowledge acquisition. In K. McGraw and C. Westphal (eds.) *Readings in Knowledge Acquisition: Current Practices and Trends*. London: Ellis Horwood.

Duda, R.O., and Hart, P.E. (1973). *Pattern Classification and Scene Analysis*. New York: Wiley.

Duncan, K.D. (1981). Training for fault diagnosis in industrial process plant. In J. Rasmussen and W.B. Rouse (eds.), *Human Detection and Diagnosis of System Failures*. New York: Plenum Press.

Gruber, T. (1991). Learning why by being told what: Interactive acquisition of justifications, *IEEE Expert*,6, 65-75

Jones, P.M., Rubin, K.S., and Mitchell, C.M. (in press). Validation of intent inferencing by a model-based operator's associate, *International Journal of Man-Machine Studies*.

Kieras D.E., (1988). What mental model should be taught: Choosing instructional content for complex engineered systems. In J. Pstotka, L.D. Massey, and S.A. Mutter (eds.), *Intelligent Tutoring Systems: Lessons Learned*. Hillsdale, NJ: LEA, pp. 85-111.

Macmillan, S., Emme, D., and Berkowitz, M. (1988). Instructional planners, lessons learned. In J. Pstotka, L.D. Massey, and S.A. Mutter (eds.), *Intelligent Tutoring Systems: Lessons Learned*. Hillsdale, NJ: LEA.

McGraw and Seale, (1988). Knowledge elicitation with multiple experts: Considerations and techniques. *Artificial Intelligence Review*, 2, 31-44.

Merrill, M.D. (1989). An instructional design expert system, *Computer-Based Instruction*, 16, 95-101.

Meyers, M.A. and Booker, J.M. (1989). A practical program for handling bias in knowledge acquisition. *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada.

Mitchell, C.M., and Miller, R.A. (1986). A discrete control model of operator function: A methodology for information display design, *IEEE Transactions of Systems, Man, and Cybernetics*, 16, 343-357

Murray, T. and Woolf, B.P. (1991). A knowledge acquisition tool for intelligent computer tutors, *SIGART Bulletin*, 2, 9-21.

Olson, J.R., and Rueter, H.H. (1987). Extracting expertise from experts: Methods for knowledge acquisition. *Expert Systems*, 4, 152-168.

Rasmussen, J. (1981). Models of mental strategies in process plant diagnosis. In J. Rasmussen and W.B. Rouse (eds.), *Human Detection and Diagnosis of System Failures*. New York: Plenum Press.

Regian, J.W. (1989). Intelligent tutoring systems: Success stories and future directions. *Proceedings of the 1989 IEEE International Conference on Systems, Man, and Cybernetics*, Cambridge, MA.

Russell, D.M., Moran, T.P., and Jordan, D.S. (1988). The instructional design environment. In J. Pstotka, L.D. Massey, and S.A. Mutter (eds.), *Intelligent Tutoring Systems: Lessons Learned*. Hillsdale, NJ: LEA.

Wenger, E. (1989). *Artificial Intelligence and Tutoring Systems*. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Woolf, B. and Cunningham, P.A. (1987). Multiple knowledge sources in intelligent teaching systems. *IEEE Expert*, 2, 41-54.

Appendix
Instructions for Software Maintenance

A.1 Software Organization

All files related to this project are kept in a folder named "AIT Folder." The contents of this folder are shown in Figure A-1. The AIT-INIT file contains some basic functions for loading and saving, and definitions of the appropriate paths for all files. The DEFSYSTEM folder contains the code for defining and maintaining systems. All the source files of this project are included in a system called the AIT system. The details of using the system feature are described below. The SOURCE folder contains the source files and the BIN folder contains the compiled versions of the project files.

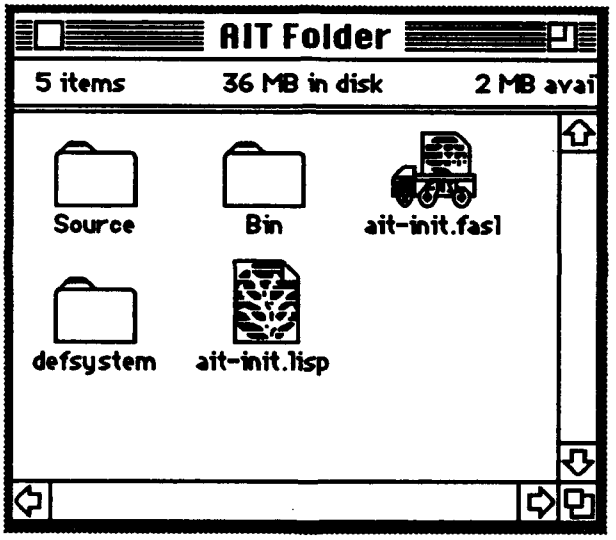

Figure A-1. The Contents of the AIT Folder

A.2 Maintaining the Source Code

The source files are maintained using a public domain DEFSYSTEM software modified to enhance its ease of use by Honeywell SSDC. The list of files that constitutes the system and the order in which they should be loaded are described in the file SYSDEFS. The SYSDEFS file is in turn maintained by the developer through the Edit System dialog box and the Systems menu. The Edit Systems dialog box (shown in Figure A-2) shows a listing of all files that make up the system. By clicking on a "Radio" button, the developer can switch between displaying the files in alphabetical order or in the order in which they are to be loaded.

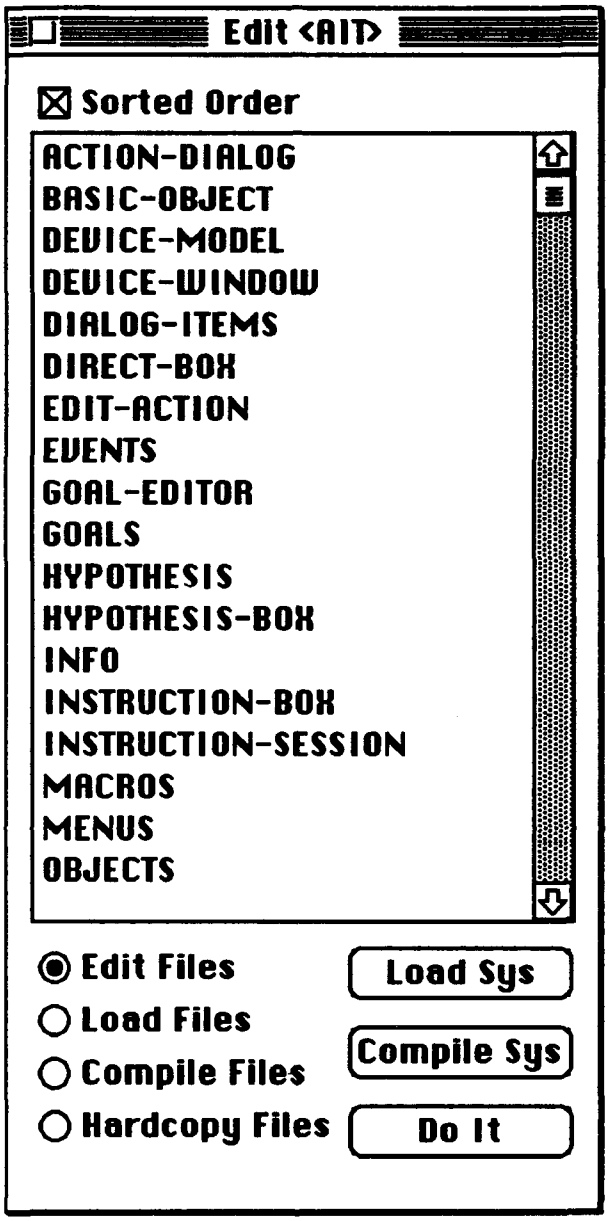

Figure A-2. The Edit System Dialog Box

Other options available in the dialog box allow the developer to load, edit, compile and hardcopy one or more files. These operations are performed by first selecting the files, setting the appropriate "Radio" button and finally clicking the "Do It" button. Clicking on the "Load Sys" button at any time loads all the files that have changed since the system was last loaded. Clicking on the "Compile Sys" button compiles all source files that have changed since the last compile.

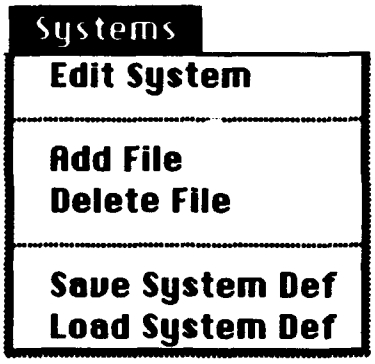

Figure A-3. The Systems Menu

The menu items under the "Systems" menu allow the developer to bring up the "Edit System" dialog box. New files can be added by choosing the "Add Files" option. A file dialog box comes up allowing the new file to be selected. The new file is inserted just above the first selected item in the "Edit System" dialog box. If no file is selected in the "Edit System" dialog box, then the new file is added at the end of the list. New files cannot be added to the system when the "Edit System" dialog box is displaying a sorted list of files. Files may be deleted from the system by selecting them and choosing "Delete File" from the "Systems" menu.

Adding and deleting files from the system does not by itself change the SYSDEFS file. After making several additions/deletions, the SYSDEFS file can be updated using the "Save System Def" item from the "Systems" menu. Also, if the systems file has been updated manually, then that information can be loaded using the "Load System Def" option under the "Systems" menu.

A.3 Loading of Code into LISP Image

The AIT software was developed under Macintosh Common LISP Version 2.0b1 with patches 1-3. All indications are that MCL V2.0 will be released in its final form in early 1992. We do not anticipate any compatibility problems with the released version.

When LISP is launched, it automatically loads the INIT file from its home folder. A sample INIT file is included in our software package. Inside the INIT file are commands to load another file AIT-INIT (described earlier). This in turn loads all the files necessary for AIT to run.

When loading is complete, the welcome prompt is displayed in the LISP listener. At this time the current package will be the AIT package. A function called STARTUP has been defined in the file STARTUP.LISP. STARTUP creates all the necessary windows, sets up the menubar and kicks off the simulation. STARTUP also looks for and loads a file HEAT-REACTOR from the home directory. This file contains all the problems, solutions and goal trees. Another file that STARTUP looks for is RECIRC.PICT. This is a PICT format file containing the picture of the heat reactor. This file can be modified using any drawing application such as MacDraw. If any changes are made to RECIRC.PICT, care must be taken to save it back as a PICT file.

A.4 Making an AIT Image

An image file is an application that contains all the functionality of the LISP environment as well as any other code that was loaded before making the image. Its advantage is that it reduces the time it takes to load user defined code. An image containing the AIT code can be easily made by invoking the function MAKE-AIT-IMAGE. You will be prompted for an image name. Subsequent changes to AIT can now be made by invoking this image.

The image is setup to load the INIT file automatically on being launched. This has the effect of loading all changes made to the source code since the last image file was created. The STARTUP function is also called automatically to create all the windows and set up the menubar.

A.5 Making the AIT Application

A LISP application is defined as one that contains all the LISP functions except for the compiler, inspector and other such developer features. An application is the form in which software is delivered. Users of applications do not require LISP or a license to use it. The AIT application can be made by calling the function MAKE-AIT-APP after loading all the AIT code. This function can also be called in a previously made image.

When the application s launched, the STARTUP function is called to setup all the windows and the menubar automatically. The AIT application should be distributed along with the file RECIRC.PICT. If example problems are to be included, the file HEAT-REACTOR must also be included.