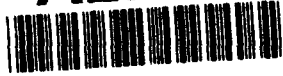


40

Center for Human-Machine Systems Research

School of Industrial and Systems Engineering, Georgia Institute of Technology
Atlanta, Georgia 30332-0205

AD-A273 986



Knowledge Organization in Intelligent Tutoring Systems for Diagnostic Problem Solving in Complex Dynamic Domains

Vijay Vasandani and T. Govindaraj

Technical Report CHMSR-93-1



September 1993

93-30542



Reproduction in whole or in part is permitted for any purpose of the United States Government.

This research was supported by the Navy Manpower, Personnel, and Training R&D Program of the Office of the Chief of Naval Research under Contract N00014-87-K-0482.

Approved for public release; distribution unlimited.

93 12 1 6002

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 1993 September		3. REPORT TYPE AND DATES COVERED Technical
4. TITLE AND SUBTITLE Knowledge Organization in Intelligent Tutoring Systems for Diagnostic Problem Solving in Complex Dynamic Domains			5. FUNDING NUMBERS C: N00014-87-K-0482 PE: 0602233N PR: RM33M20	
6. AUTHOR(S) Vijay Vasandani and T. Govindaraj				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Georgia Institute of Technology Center for Human-Machine Systems Research School of Industrial and Systems Engineering 765 Ferst Drive, Atlanta, GA 30332-0205			8. PERFORMING ORGANIZATION REPORT NUMBER CHMSR-93-1	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research Cognitive Science Program (Code 1142CS) 800 North Quincy Street Arlington, VA 22217-5000			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES Supported by the Office of the Chief of Naval Research Manpower, Personnel, and Training R&D Program.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) Turbinia-Vyasa is a computer-based instructional system that trains operators to troubleshoot and diagnose faults in marine power plants. It is implemented on Apple Macintosh II computers. The simulator, Turbinia , is based on a hierarchical representation of subsystems, components, and primitives. Vyasa is the computer-based tutor that teaches the troubleshooting task using Turbinia . The simulator, an interactive, direct manipulation interface, and the tutor (with its expert, student, and instructional modules) comprise the architecture for the instructional system. In this paper, we discuss the details of knowledge organization that supports the functions of the three major elements of the intelligent tutoring system.				
14. SUBJECT TERMS graphical interfaces; knowledge representation; fault diagnosis; training; maintenance; intelligent tutoring systems; intelligent computer assisted instruction; interactive learning environments; marine power plants; simulation			15. NUMBER OF PAGES 51	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT	

Knowledge Organization in Intelligent Tutoring Systems for Diagnostic Problem Solving in Complex Dynamic Domains

Vijay Vasandani and T. Govindaraj

Center for Human-Machine Systems Research
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0205, USA
+1 404 894 3873, tg@chmsr.gatech.edu
(Address all correspondence to TG.)

DOCUMENT PREPARED BY

Unannounced Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

Abstract

Turbinia-Vyasa is a computer-based instructional system that trains operators to troubleshoot and diagnose faults in marine power plants. It is implemented on Apple Macintosh II computers. The simulator, **Turbinia**, is based on a hierarchical representation of subsystems, components, and primitives. **Vyasa** is the computer-based tutor that teaches the troubleshooting task using **Turbinia**. The simulator, an interactive, direct manipulation interface, and the tutor (with its expert, student, and instructional modules) comprise the architecture for the instructional system. In this paper, we discuss the details of knowledge organization that supports the functions of the three major elements of the intelligent tutoring system.

Introduction

Human operators of highly automated systems such as aircrafts and steam power plants must be capable of managing the systems both under normal conditions and in the presence of system malfunctions. Typically, the operators are presented with vast quantities of information concerning the system state. They must combine this information with external inputs from the environment and determine the courses of action. Even though computers and automatic control systems are generally employed for information processing in real time, complete automation based on fully autonomous systems is neither possible nor desirable. There is currently no substitute for human judgement and knowledge to set high level system goals, monitor system states, and intervene and compensate for problems that the automated control systems are unable to handle. Therefore, it is essential that human operators are skilled in planning and problem solving (Rasmussen 1986; Woods 1986; Wickens 1984; Rouse 1982; Sheridan & Johanssen 1976).

Operators' ability to diagnose faults and take appropriate corrective actions promptly is highly desirable. Typically, an operator identifies a malfunctioning components by observing abnormal states of the system, forming hypotheses about failure based on her mental models of what is normal and abnormal, and verifying each hypothesis by conducting diagnostic tests. Operators often find it difficult to develop appropriate mental models that describe the causal behavior in such systems.

The diagnostic problem solving task is often complicated by the size, interactions and dynamics of the system. Size refers to the number of components in the system. An increase in size increases the probability of failure in the system and makes troubleshooting difficult by increasing the alternatives that can explain the observed abnormal system behavior. Interaction between parts of the system and the propagation of abnormal system behavior makes diagnosing faults difficult.

Success in fault diagnosis depends upon the operator's use of system knowledge at multiple levels of abstraction and detail (Rasmussen 1985). Efficiency of diagnosis is enhanced by timely compilation, integration and organization of operational information and system knowledge. During diagnosis, operators must combine symptom information with mental resources concerning system knowledge (Govindaraj 1988). Cognitive aspects of diagnostic problem solving such as the integration of the system state information with mental models can benefit from appropriate training. A training program should help organize system knowledge and operational information, including symptom-cause relationships.

Training for diagnostic problem solving can be provided on-the-job or on simulators (Johnson 1988; Kearsley 1987; Goldstein 1986; Bureau of Naval Personnel 1957; Naval Training Command 1973). On-the-job training is usually very expensive and the consequences of an error can be catastrophic. Malfunctions occur infrequently and it may be undesirable or impossible to duplicate them during training. Systems that can simulate a wide range of failure conditions offer a good alternative training environment. However, simulators by themselves are unable to provide appropriate help since they cannot evaluate a student's misconceptions from observed actions. A simulator coupled with an intelligent, computer-based tutor can provide effective training based on an evaluation of a student's misconceptions from observed actions. Such a combination of simulator and tutor constitutes an intelligent tutoring system (ITS).

Research on intelligent tutoring and training systems has traditionally been concerned with imparting basic skills in mathematics, electricity, physics and computer programming (Wenger 1987; Sleeman & Brown 1982). These domains lack the complex interactions between subsystems that are characteristic of most engineering domains. Due to the inability to represent complexity, most ITS design principles have not been successfully extended from simpler, less constrained domains to complex engineering systems (Burns, Parlett & Redfield 1991; Frasson & Gauthier 1990; Psotka, Massey & Mutter 1988). Suitable methodologies for representing the system complexity are lacking.

The inability to represent system complexity stems primarily from the lack of a methodology to decompose and organize knowledge about large dynamic systems. We have developed a methodology that addresses the representation problem by decomposing, organizing and representing domain knowledge of complex dynamic systems for building functional, computer-based intelligent tutors. We separate the domain knowledge from pedagogical knowledge. Domain knowledge is further decomposed into system and troubleshooting task knowledge.

System knowledge at different levels is organized into a structure-function-behavior model. Trou-

troubleshooting task knowledge is organized to facilitate evaluation of student misconceptions. The organization of pedagogical knowledge assists in planning of pedagogical functions including inference of misconceptions and delivery of instructions using a blackboard-like control architecture.

The methodology for decomposing, organizing and representing knowledge is suitable for engineering domains (e.g., power plants, aircrafts, automobiles etc.) characterized by large size, high degree of interaction between subsystems and complex dynamics. The diagnostic task involves the identification of a malfunctioning component based on observed abnormal system behavior. Limited availability of gauges constrains the operators from observing *all* abnormal system behaviors.

We have implemented an experimental instructional system for a marine power plant. The instructional system comprises of a simulator and a tutor. A large number of failure conditions are simulated. The tutor is designed to improve the student's troubleshooting skills by providing practice and exposure to realistic situations.

In the next section, we provide a brief review of the general architecture of intelligent instructional systems. The problem in extending the existing ideas to a wider range of domains and the goals of current research are outlined. In the section that follows, which forms the core of this paper, we describe a methodology for organizing knowledge and an instructional system architecture that uses this methodology. We then describe the student-tutor interface. An experimental evaluation of the training system is presented next. Finally, we conclude with a discussion of the results and some observations.

Intelligent Tutoring In Complex Systems

Background

Even though research on intelligent tutoring systems has been in progress for over two decades, only a small fraction of the research deals with engineering domains. Surveys of intelligent tutoring systems can be found in (Sleeman & Brown 1982; Wenger 1987; Psotka, Massey & Mutter 1988; Frasson & Gauthier 1990; and Burns, Parlett & Redfield 1991).

SOPHIE, designed to teach troubleshooting in electrical circuits, was perhaps one of the first ITSs in an engineering domain (Brown, Burton & de Kleer 1982). Since SOPHIE, a great deal of progress has been made in computer-based training for electronic troubleshooting. The SHERLOCK family of tutors (Lajoie & Lesgold 1990; Lesgold 1990a; Lesgold 1990b; Lesgold et al. 1991), developed for a complex electronic troubleshooting job in the Air Force, has a far richer representation of the work environment than SOPHIE. Intelligent Maintenance Training System (IMTS) and its successors provide interactive environments for constructing domain-specific simulations and training scenarios for a wide variety of domains including electronics (Towne & Munro 1988). Finally, in the domain of marine steam power plant, we have developed ITSs.

SHERLOCK provides students with realistic means of practicing the task with context-specific

support and feedback. Instead of imposing a particular troubleshooting strategy it provides help that is relevant to the current performance of the students when an impasse is reached. In addition, a model of the student's competence and performance gives SHERLOCK the capability to provide personalized instructions.

Intelligent Maintenance Training System (IMTS) provides an interactive environment for constructing domain-specific simulations and training scenarios (Towne & Munro 1988). For instance, IMTS has been used to develop a maintenance training program for SH-3 Helicopter Blade-fold System. The Helicopter Blade-fold System is a moderately complex, electrically controlled hydraulic system. The maintenance training program uses Profile, a generic troubleshooting expert of IMTS, to aid operators in improving their diagnostic performance. The techniques for assessing and supporting the student's performance in this training program, along with the ones used in SHERLOCK, are the most extensive among all existing systems.

Another example from a complex system domain is MACH-III (Massey, de Bruin & Roberts 1988) used to train maintenance operators of HAWK radar systems to troubleshoot complex electronic devices at tactical installations. This system uses model-based qualitative reasoning techniques to generate explanations of normal and faulty radar operations for use in training.

In domains such as power plants, ITS research on operator training has produced STEAMER (Hollan, Hutchins & Weitzman 1984), The Recovery Boiler Tutor (Woolf 1986), and AHAB (Fath, Mitchell & Govindaraj 1990). STEAMER does not teach any specific task. Instead, it uses innovative graphical techniques to display the behavior of portions of the power plant. STEAMER has neither the means to evaluate the needs of the students nor can it provide help upon request to improve the student's understanding of the power plant. Thus, it is not really an intelligent tutoring system although it provides powerful graphical interfaces for interactive inspection of simulated faults in steam power plants.

The Recovery Boiler Tutor simulates the thermal and chemical processes in the boiler unit of a power plant. It has knowledge of boiler operating procedures for normal and emergency situations. The tutor uses this knowledge to teach the steps involved in controlling the situations arising from emergencies. The tutor also provides the students with the facility to interact with the simulator and stop the boiler processes to engage in activities needed to improve the understanding of the system.

AHAB addresses the complexities of a power plant such as size and interactions between sub-systems better than any other training system in its domain. Using qualitative approximation (Govindaraj 1987) for simulation and knowledge representation, and a discrete control model (Miller 1985; Mitchell & Miller 1986) of the operator's task, AHAB teaches symptomatic and topographic search strategies (Rasmussen 1986) for troubleshooting. It assists the student using context-specific symptomatic and topographic diagnostic tests and evaluates the performance of the student based on deviations from the strategy prescribed by its task model.

The rest of this section is divided into three parts. First, we identify the major constituents of the

instructional system architecture. Important characteristics of each constituent are discussed next. Finally, we identify the characteristics desired in a tutoring system for complex systems.

General Architecture of ITS

In general, all intelligent tutoring systems have a similar architecture (see "Tutor" in Figure 1), comprised of an expert module, a student module and an instructional module. In addition, a simulator provides the training environment. The expert module contains the domain expertise which is also the knowledge to be taught to the student. The student module contains a model of the student's current level of competence. The instructional module is designed to sequence instructions and tasks based on the information provided by the expert and student models. Also, the interface used to communicate knowledge to the student can be treated as a separate component of these systems. Together with the simulator and an interactive interface, the three components of the tutor (i.e., the expert, student, and instructional modules) comprise the architecture for the instructional system.

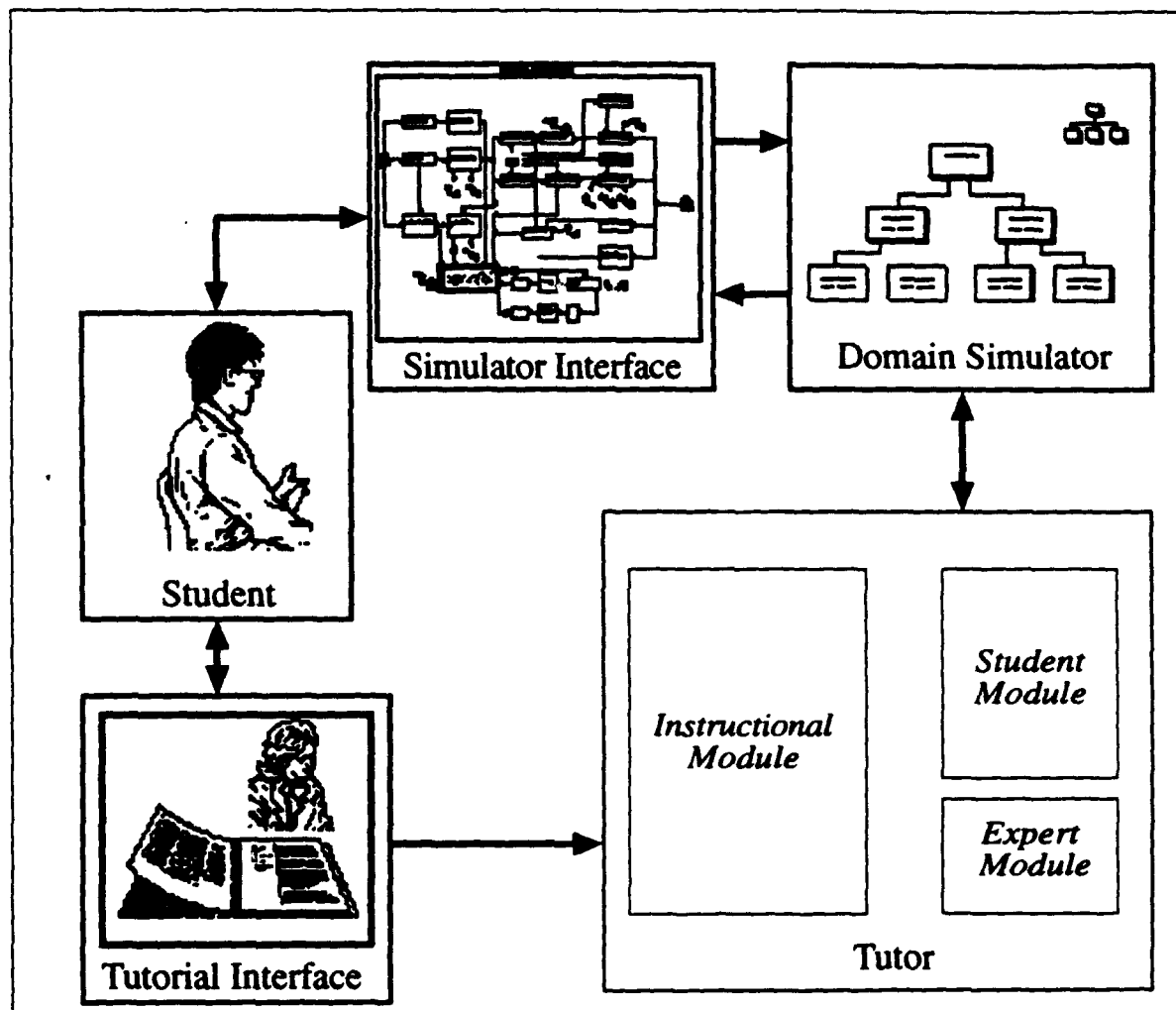


Figure 1. Major Components of Instructional System

The *expert module* is the heart of the ITS and provides the domain intelligence and expertise. The domain knowledge refers to the subject matter as it relates to the task for which it will be used. It is not limited to procedures for executing the task and includes material that provides the foundation and justification for the application of procedures. Thus, domain knowledge is of two types: *declarative* and *procedural* (Anderson 1988; Charniak & McDermott 1985; Nilsson 1980; Rich 1983; Winston 1980). Whereas declarative knowledge refers to objects in the domain, facts about them and their interrelationship, procedural knowledge refers to set of compiled rules for executing the task.

The *student module* in an ITS maintains a model of the student's current understanding of the domain. The student model is used to evaluate the student's need and help the instructional module in preparing appropriate individualized instructions. It stores actions taken by the student and has some means of representing the student's knowledge derived from recorded actions. Representation of data in such a student model must facilitate its comparison with the expert model of the task to enable evaluation of misconceptions in the student.

The *instructional module* of an ITS is responsible for several activities. Its primary function is to control the curriculum, that is, select the material to be presented and its form of presentation. In addition, the instructional module evaluates student's misconceptions based on observed actions. To achieve these objectives, the instructional module makes use of pedagogical rules pertaining to presentation methods, query response and conditions for tutorial intervention. It also incorporates an algorithm that facilitates comparison of knowledge in the expert and student models and a framework for evaluating misconceptions based on this comparison.

Domain Simulator and Interactive Interface

In complex real world systems, it is often impossible to generate situations merely for the purpose of training. Therefore, in spite of being costly and labor extensive, simulators provide the only practical alternative for any training program. Although simulators have, in the past, been used alone to provide instructions (Towne 1986), these simulators by themselves lack the ability to evaluate student misconceptions. However, these simulators are an important part of any training program and, when coupled with computer-based tutors, enhance the effectiveness of an instructional system.

A good interface makes the knowledge of the tutor transparent to the student and helps the student understand the complex structure, function, and behavior of the controlled system. In addition, a well designed interface addresses the external-internal task mapping problem (Moran 1983) and establishes a semantic link between the actions relevant to the task in the domain and the actions to be taken at the interface (Miller 1988).

Extensions to complex systems

The difficulties in the implementation of ITSs in complex real-world systems domains arises due to: (1) a lack of simulation techniques, and (2) the difficulty in organizing knowledge. Simulators

must have at least moderate levels of dynamic, structural and temporal fidelity (Su 1985; Su & Govindaraj 1986). For most real-world systems, due to their sheer size and complexity of processes and interactions involved, developing such a training environment is rather tedious. In the absence of an appropriate tool that supports rapid construction of these training environments, the task of building large simulators is made even more difficult. Environments such as RAPIDS and its successors (Munro & Towne 1993 (in press)) should help in overcoming this problem. Moreover, constructing simulators is computationally expensive. In an ITS, it is extremely important to conserve its computational resources for the tutor rather than consume it on the simulator. While the shortage of computational resources should not be a problem in the future, due to the rapid pace of advances in computer technology, there is still a need to explore techniques for rapid construction of training environments for instructional systems.

The second problem, that of organizing knowledge, is the focus of our research described in this paper. ITSs must represent knowledge in multiple levels of detail and abstraction. For most real-world systems, this makes the volume of knowledge to be represented overwhelming. Furthermore, this knowledge is interrelated and tightly coupled, and therefore cannot be stored as isolated modules. Knowledge must be integrated into proper contexts to assist in the development of good mental models. A framework that can help integrate the large volume of knowledge associated with the safe operation of real-world systems is needed.

In this paper, we describe a methodology for organizing knowledge. (An earlier, brief, version appears as (Vasandani & Govindaraj 1993 (in press))). Specifically, we describe a methodology for decomposing, organizing and representing system and task knowledge of a large complex dynamic system. Knowledge is organized to facilitate its use in intelligent instructional systems. Results of our research provides a pragmatic approach for extending the applications of ITS to real-world systems. Details of knowledge organization are discussed next.

Knowledge Organization

General

Organization of knowledge determines the *intelligence* of a training system. Absence of systematic organization of knowledge adversely affects the effectiveness of a computer-based tutor. In an effective ITS, the domain knowledge must be separated from teaching knowledge and made explicit. Having explicit domain knowledge makes it easily accessible and communicable to the student. Separation of knowledge also makes it possible for the tutor to present the domain knowledge in more than one way. This allows an ITS to function with more flexibility and present instructional material in different styles.

For a large, complex, dynamic, real-world system, the problems related to the knowledge and its representation in an instructional system are two-fold. First, the volume of knowledge is enormous. Second, its organization is critical for the success of the instructional system. Both the type and organization of knowledge in an instructional system also varies with the distribution of teaching

and learning responsibility between the student and the tutor (Rickel 1989). Tutors that attempt to maintain a balance of control between the student and the tutor (i.e., mixed-initiative tutors) have the largest amount of structured knowledge as compared to tutor-dominated traditional computer-aided instructional programs or student-dominated discovery learning environments (Sleeman & Brown 1982; Wenger 1987; Psotka, Massey & Mutter 1988). Thus, a consistent knowledge organization methodology can not only increase the pace of progress by cutting down on the development time but also ensure effectiveness of the instructional system.

We begin with a brief discussion of the simulator, and knowledge requirements for diagnostic problem solving. A methodology for decomposing and systematically organizing knowledge concerning complex dynamic systems follows. This methodology provides a framework for decomposing knowledge into smaller and easily comprehensible units for use in instructional systems. Implementation details are discussed later.

Simulation Via Qualitative Approximation

We use qualitative approximation for the design of moderate fidelity simulators. Basic principles of this approach were developed in (Govindaraj 1987). In qualitative approximation, the system states are represented by qualitative measures such as "pressure low" and "flow rate has been steadily decreasing." Exact numerical values are not used. Such qualitative state representation aids the operator involved in troubleshooting by eliminating the need to compare observed state values to nominal values. Also, large systems can be simulated with a moderate amount of computational power due to reduced computational requirements.

A hierarchical description of the system is employed. System components are grouped into a number of subsystems based on their function. For instance, an oil-fired steam power plant on a ship is comprised of the following primary subsystems: fuel oil, feed water, steam, lube oil, and control air. Some components might belong to more than one subsystem. For example, the condenser is part of the feed water subsystem as well as the steam subsystem. Components are classified into a number of generic types, which are then broken down into a small number of primitives. A condenser as well as an economizer, therefore, can be classified as heat-exchangers. This is a rather simple arrangement of the hierarchy based on the physical nature of components that form the system.

Knowledge Requirements for diagnostic problem solving

The knowledge required includes nominal values of the state variables and parameters, and operational principles of different types of system, e.g., thermodynamics and heat transfer for the fuel system, or electrical characteristics for a turbogenerator. Problem solving and compensation for failures require processing of information from various subsystems using efficient troubleshooting strategies. Therefore, an ITS must be capable of organizing and presenting knowledge about the system and the troubleshooting task at several levels of granularity or detail.

Four components of knowledge are necessary in an intelligent tutor for diagnostic problem solving

in complex dynamic domains. These components are: (1) a large amount of system knowledge organized to facilitate evolution of system states with time, (2) troubleshooting task knowledge, including knowledge about failures and student actions, (3) knowledge to infer a student's possible misconceptions from observed actions, and (4) pedagogical knowledge to realize the tutoring objectives.

Knowledge about the system and the troubleshooting strategies constitute an expert model of the operator's task. This knowledge must be organized in a manner that is easily accessible and communicable to the student. The instructional module uses this model to train students to use proper diagnostic problem solving strategies. Knowledge of student's actions can help the instructional module to infer possible misconceptions. Finally, knowledge of tutoring goals and how they are to be realized guides the instruction and its communication. In what follows, we describe a framework for decomposing and organizing knowledge. Figure 2 summarizes the components of knowledge. Each of the components is described next.

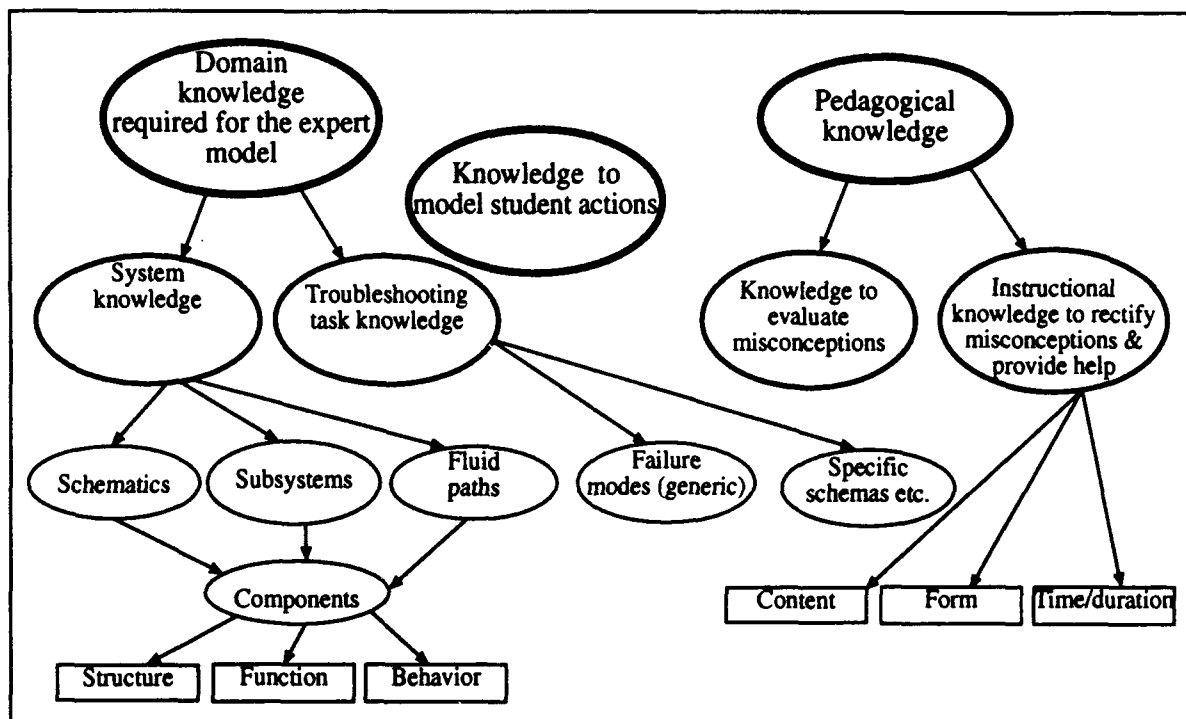


Figure 2. Summary of Knowledge Components

System Knowledge

Successful fault diagnosis in complex dynamic domains is aided by multiple representations of the system's functional properties (Rasmussen 1986). The expert model must therefore have access to multiple representations of the system knowledge. *Schematics*, *functional subsystems* and *fluid paths* are three possible means of representing the system knowledge. A *schematic* is a pictorial representation of the components in the system. A *functional subsystem* is a collection of compo-

nents responsible for a higher level system function. *Fluid paths* help in visualizing the system in terms of different fluids that flow through the system. The three representations are complementary rather than mutually exclusive. Since schematics are collections of components organized by physical or logical connectivity, a particular schematic may contain several subsystems and fluid paths. A detailed description of schematics, functional subsystems and fluid paths is provided next.

Schematics

A schematic presents a view into the structure of the system. Typically, a schematic shows the sequence in which certain components and gauges appear in a real system. It is also a structure that reveals the logical proximity of two physically unconnected components such as the burner and the stack in a combustion unit. A configuration of all components either responsible for a higher level function or sharing a common fluid is yet another example of a schematic.

During diagnostic problem solving, schematics are typically used to view the configuration of components and gauges. Scanning through the various schematics permits an operator to visualize the sequence of system processes as they occur in the system. In a steam power plant, for example, the schematics may display the stages of power generation in a sequence starting with the combustion of fuel, followed by steam generation, steam condensation and preheating of condensed steam for re-use in a closed loop water circuit. The operator's interaction with the system during a troubleshooting task involves probing gauge readings in the suspected areas of failure through schematics.

Grouping of components in schematics may depend upon some other factors such as frequency of interaction and level of dependency. There are parts of a system that commonly interact with each other. For instance, in a power plant, the performance of a steam generation unit is affected by the performance of the combustion unit. Hence, the steam generation unit and the combustion unit are displayed in a single schematic. Certain parts of a system do not significantly affect other parts of the system and thus are viewed in isolation. For example, problems related to lubrication are usually confined to lube oil path and rarely affect other fluid paths, unless left unattended for a long time. Finally, there are some failures in a system that occur more frequently than others. Components and gauges required for investigating such failures are confined, as far as possible, to a single schematic.

Functional subsystems

Functional subsystems are collections of components responsible for achieving specific higher level system functions. There are several higher level system functions that collectively contribute to the system goals. For instance, in a marine power plant, the functions are combustion, steam generation, power generation, steam condensation, feed water preheating, auxiliary steam use, saltwater service, lubrication and control air distribution.

A functional subsystem is described by information related to (1) fluid paths passing through the subsystem; (2) components through which a given fluid flows; (3) the order in which the components and gauges appear in each fluid path; (4) the connected subsystem on either side of the fluid

path; and (5) the schematic in which the subsystem may be found.

Fluid paths

In decomposing a system by fluid paths, all components on the same fluid path are represented in a group. Additional system knowledge based on fluid paths consists of (1) schematics in which the fluid is found, and (2) the subsystems through which the fluid flows. Examples of fluid paths in steam power plants are combustion air, fuel oil, superheated steam, feed water and saltwater.

Components

Each of the three system representations described above involves mechanical components and gauges. The lowest level of description of system knowledge is hence at the component level. System knowledge at the component level has three attributes: structure, function and behavior.

A component's structure, for the most part, refers to its connections to other components on the input and output side, the fluids carried by it, the gauges attached to it, and its association to a schematic or a functional subsystem. Functional knowledge about a component is its intended use in the system and its contribution to the higher level functions of the system. Knowledge of a component's behavior concerns its states. Since the behavior of a component is different under normal and failed modes, the behavior knowledge, like the structural knowledge, is different for the two modes.

Together, the structural, functional and behavior knowledge of a system and its components form an essential part of the expert's knowledge. Structural, functional and behavior knowledge are discussed next.

Structural knowledge

Most of the structural information for components is the same in normal and failed states. The structural information that remains invariant after a failure includes its connectivity relationship to other components, the fluids flowing through it, and its association to a particular subsystem and schematic. When a component fails, some structural information changes. For example, a valve with its control set to the open position but its blade stuck in the closed position represents a structural change for a valve when it is blocked shut. Such structural changes for failed components will be discussed later as a part of "troubleshooting task knowledge."

Functional knowledge

Functional information defines the purpose or role of a component in the system. Functional knowledge of a component depends upon its structure. For example, a pipe in the system may be modeled as a conduit, where the function of a conduit is to transport moving fluid from one of its ends to another. In an approximate representation, where friction may be ignored, it is reasonable to define the function of the conduit in the manner described above.

In general, a number of primitive function types, like the conduit, can be identified for a system. All the components of the system can be categorized as instances of one of the primitive types. For

continuous systems, examples of primitives based on functions include sink, source, source-sink, gain, controller, reactor, transducer, heat-exchanger and phase-changer.

Behavior knowledge

Normal and failed modes of a component affect the system differently. The manner in which the system state values are affected by the presence of a component, in both the normal and the failed states, constitutes the component behavior knowledge.

Normal behavior of components is responsible for normal state values during system operation. For example, normal behavior of the main condenser is responsible for a lower outlet temperature of the hot medium as compared to its inlet temperature. As the hot medium moves from inlet to outlet it undergoes a phase change from gas to liquid. The same normal behavior of the main condenser is also responsible for a corresponding increase in temperature of the cold medium as it flows from its inlet to outlet port. Behavior of all components can be explained by the laws of science, e.g., the law of conservation of energy explains the normal behavior described here.

Abnormal behavior describes the manner in which certain state values are affected by a failure in the component. For tutoring, the behavior information for a failed component includes contextual information about specific gauges affected by the failure. The explanations for the abnormal gauge readings in terms of cause-effect relationships also form a part of the component behavior knowledge represented in the tutor.

System knowledge, although essential, is not sufficient for the troubleshooting task. Troubleshooting task knowledge discussed next includes more than the operational knowledge of the system and its components.

Troubleshooting Task Knowledge

Troubleshooting task knowledge combines system knowledge and diagnostic strategies. It includes general knowledge of the types of failures in the system, detailed information on certain common failures, and cause-effect associations for familiar failures. The nature of this diagnostic problem solving knowledge is described here.

A mechanical component in a physical system such as a steam power plant can fail in more than one way. There are four common modes of failure in components: (a) blocked-shut, (b) stuck-open, (c) leak-in, and (d) leak-out (Fath, Mitchell & Govindaraj 1990). Faults in components fit one or more of these four mode types. Not all components, however, fail in all four different ways. Some components have multiple faults that fit the same failure mode category. For example, a clogged valve or a valve stuck in closed position are two different ways in which the valve may be blocked-shut.

Each failure mode exhibits a typical system behavior (Fath, Mitchell & Govindaraj 1990). The typicality of such behavior provides useful diagnostic information. If the system behavior suggests a particular mode of failure, then the list of suspected components can be reduced to those that fail

in that particular mode.

The typical system behavior may depend upon the phase of the fluid in the affected path. A blocked-shut mode of failure in a liquid path, for example, causes the liquid *level* downstream to be lower than normal and the level upstream higher than normal. A similar blocked-shut failure in a gas path, on the other hand, decreases the downstream gas *pressure* and increases the upstream pressure.

In any case, system behavior associated with each mode is manifested in the form of a typical pattern of abnormal state values. Patterns of such abnormal state values can be determined by the application of the laws of physics and thermodynamics, and recognizing these patterns of abnormalities during fault diagnosis often helps to identify the type of failure in the system.

System behavior associated with failure mode sometimes deviates from the expected abnormal behavior (Fath, Mitchell & Govindaraj 1990). The way in which the system components are configured is often responsible for such a deviation. For instance, a source-sink such as a deaerating feed tank located downstream in the blocked-shut feed water path may prevent further propagation of low feed water level. The deaerating feed tank imposes such a behavior on the system because it is an "infinite" source of feed water which can at least temporarily compensate for any loss in the water level. The expected abnormal behavior associated with a mode of failure may therefore be confined to the vicinity of the failed component. Furthermore, with the limited availability of gauges around the failed component, the abnormal behavior may not be observable. Knowledge of such deviations from the norm is essential for correct identification of the type of failure in the system.

Even when the failure mode is recognized from the system behavior, it may not be very useful. An expert needs more than just the knowledge about modes of failure and their associated system behavior. However, when the expert's troubleshooting knowledge also includes information on all possible modes of failure for each component, it can be helpful in at least reducing the list of suspected components.

Finally, to isolate the failed from the suspected components and to diagnose the fault, additional information such as the gauges affected by the failure and causal relationship between abnormal system states for every fault is required. Knowledge of the affected gauges and the system states for the individual faults can provide the verification of the final diagnosis.

There are other elements of the troubleshooting knowledge, accumulated through experience, that make fault diagnosis in a large complex system time-efficient (Govindaraj & Su 1988). This experiential knowledge, based on prior cases of solved and unsolved problems encountered by the operator, is usually responsible for the formation and rapid refinement of an initial set of hypotheses of either suspected components, subsystems, or fluid paths.

Such knowledge based on cases is particularly useful when unusual problems are encountered. While there is inadequate information to implement a full-fledged case-based reasoning system, knowledge of known cases useful for diagnosis is incorporated in a knowledge source associated

with the blackboard architecture discussed later in this paper.

Experiential knowledge is activated by the observation of obvious and non-obvious (i.e., discovered only upon investigation) symptoms. In a complex dynamic system, the size of the system and the effects of fault propagation make it impossible to uniquely associate a symptom to a specific fault. However, in such systems, observable symptoms still help to limit the search for the failed component to a specific location in the system. For example, the symptoms may indicate that a particular higher level function of the system has been affected by the fault. This helps to confine the search for the failed component to components comprising the subsystem responsible for the affected function. Symptoms may further help to categorize the faults, for example, they may separate those related to components with moving parts from those related to speed or load. Such a categorization of failures further reduces the search space for a failed component. For example, a search space generated by a set of all components with *moving parts* in the combustion system of a power plant is likely to be much smaller than the set of all components in the combustion subsystem.

An operator's fault diagnosis task is also aided by inferences based on failure schemas built through experience. These failure schemas are a part of experiential knowledge. The schemas represent some of the familiar ways in which the system fails. A schema is activated by a symptom and proposes a hypothesis or a partial solution to the diagnostic problem. This is similar to the use of symptomatic search during a troubleshooting task (Rasmussen 1986; see below).

The partial solution may be a diagnostic test that either provides a conclusive inference or activates another schema. For example, smoke in a boiler may activate a schema that recommends checking for smoke color. Black smoke may then trigger an *incomplete-combustion* schema while white smoke may trigger an *excessive-air-in-the-burner* schema. An abnormal fuel temperature with black smoke in the boiler may prompt the incomplete-combustion schema to specify desuperheated-steam or fuel path as the path suspected of containing the failed component.

Rasmussen (1986) has characterized the application of the troubleshooting task knowledge into two diagnostic strategies: symptomatic and topographic search. Symptomatic search is a simple and economical pattern matching strategy where a successful association between cause and effect is generated based on prior experience. An unsuccessful attempt with symptomatic search usually leads to topographic search. In topographic search, a hypothesis about the failed component is generated and tested by comparing a model of normal behavior of the suspected component with its behavior in the abnormally functioning system. Neither the symptomatic nor the topographic strategy is adequate in itself; instead, an expert often switches between the two strategies many times to complete the task.

We have provided an overview of an expert's troubleshooting knowledge and the diagnostic strategies. The system and the troubleshooting task knowledge discussed thus far are also normally the representation of the material to be taught by the tutor. However, the knowledge representation suitable for expert performance is not necessarily suitable for instruction or for evaluating student's

misconceptions (Clancey 1987). An alternative organization of the expert's task knowledge that may help evaluate a student's misconceptions is required.

Knowledge of Student's Actions

An important feature of an intelligent computer-based tutor is its ability to evaluate a student's misconceptions. This capability of the tutor evolves from a normative model of the student's actions. In a normative model of the student's actions, not all actions that occur at the student-tutor interface are valid. Examples of valid actions may range from requests for help to responses to queries and calls for schematics. In addition, in diagnostic problem solving, there may be some other actions performed by the student. These actions may include investigating components for gauges and checking their gauge readings. An action to investigate a component may be called an *investigative action* and a request to display the value of a particular gauge attached to the component an *informative action*. Most of the student's actions, such as the request for help, response to query, call for a change in schematic display and even investigative actions are self explanatory. These actions clearly express the intent of a well-motivated learner interacting with the tutor. However, the informative actions taken during diagnostic problem solving are associated with ambiguity concerning student's intent. We need context-specific knowledge and an understanding of the cognitive aspects of troubleshooting task to resolve these ambiguities.

In a troubleshooting task, the student maintains a set of failure hypotheses that explain the abnormal behavior of the system (Fath, Mitchell & Govindaraj 1990). A set of hypotheses is a list of components suspected to have failed. Each informative action taken by the student is an attempt to reduce the size of the set of failure hypotheses. The manner in which the list of suspected components may be revised depends upon the outcome of the diagnostic test associated with the informative action. The test results have a context-specific significance. For example, in a power plant, if the student has been alerted by a low condensate pressure alarm, it makes sense for her to check the pressure gauge on the condensate pump. If she does check the pressure gauge on the condensate pump, it is reasonable to assume that the condensate pump is probably one of the suspected components. If the pressure gauge shows a low reading, the student has reason to continue suspecting a malfunction in the condensate pump. On the other hand, if the pressure gauge reading is normal, the condensate pump may be omitted from the list of suspected components. However, when the student is alerted to a failure in the system by smoke in the boiler rather than a low condensate pressure alarm, checking for pressure across the condensate pump is inconsistent with the failure data. Thus, the knowledge of what are reasonable actions under various failure situations and how the test results ought to refine the set of failure hypotheses can help in evaluating the student's misconceptions.

A normative model of student's actions that describes the valid actions of a student for each failure condition can thus be used to evaluate students' misconceptions. The knowledge required to evaluate misconceptions using the normative model is described next.

Knowledge to Evaluate Misconceptions

The normative model describes what a student ought to do under a particular failure situation. When the student's action does not match actions suggested by the normative model, the reason can be attributed to many causes. Usually the causes are related to lack of knowledge, inappropriate knowledge or deficiencies in knowledge application skills. Evaluating a student's misconception means determining the probable cause for the deviant behavior. While suggesting remedies may be relatively straightforward when misconceptions are known with certainty, determining the misconception itself is a difficult task since the causes for a given misconception are often confounded.

In order to determine a student's misconception, the tutor needs to know the types of misconceptions that are associated with incomplete knowledge of the system or the task. Misconceptions can be categorized as those related to a lack of (1) structural knowledge of the system, (2) functional knowledge of system and components, and (3) knowledge of system behavior resulting from failures.

The lack of system structural knowledge makes the student investigate portions of the system unrelated to the failure. For example, if the abnormal system behavior in a power plant is initially observed in the boiler, the student is expected to investigate gauges mounted on the boiler or on the components in the vicinity of the boiler. If, however, the student fails to call up the schematic that contains the boiler or struggles to locate it in the schematic, it can be attributed to inadequate knowledge of system structure.

If, on the other hand, the student calls up the relevant schematic for investigations but checks components and gauges in the fluid paths unaffected by the failure, it indicates a lack of understanding of different system functions and their inter-relationships. For instance, if the observed abnormality concerns low water level in the boiler, persistent investigations along flue gas path is unlikely to yield any useful diagnostic information. Such an action is clearly an indication of the student's inability to integrate functional information about the boiler and the interactions between the fluid paths through the boiler.

Finally, pursuing a hypothesis that should have been rejected based on evidence gathered, or premature elimination of suspicion from a component due to insufficient evidence, suggests shortcomings in the knowledge of behavior related to failures. For example, if the pressure gauge on the condensate pump displays a normal reading, it is unreasonable to suspect a blocked-shut mode of failure in the condensate pump. Continued suspicion of a component in spite of evidence available to the contrary suggests inability on the part of the student to link failures to abnormal system behavior resulting from failures.

A mismatch between observed student actions and those predicted by the normative model often implies a number of confounding of causes. Therefore, some heuristic strategies are necessary to identify possible misconceptions and deliver individualized instructions. A rule-based knowledge structure is used in the tutor to identify three types of misconceptions based on observed student actions.

Structural misconception is inferred as the cause when the student investigates components in a schematic unaffected by the current failure. Knowledge of schematics affected by each failure, needed to evaluate the structural misconception, is obtained from the tutor's knowledge of the failures.

The tutor identifies a functional misconception when the most suspected subsystem or fluid path inferred from the student's action is unrelated to the failure being investigated. Most suspected subsystems and fluid paths are determined after each student action. A count is kept of the number of investigations made in each subsystem and fluid path. The subsystem and the fluid path with the maximum number of investigative-actions are also the most suspected if at least one of the last three investigations have occurred in that subsystem or fluid path. Otherwise, the most suspected subsystem or the most suspected fluid path is the one investigated last. Thus, after every action, the information concerning the most suspected subsystem and the most suspected fluid path in the student model is revised. Knowledge of subsystems and fluid-paths related to each failure, needed to evaluate the functional misconception, is obtained from the tutor's knowledge of failures.

Misconceptions concerning a student's knowledge of fault related system behavior is inferred when a student continues to pursue a failure hypothesis that should have been rejected based on the diagnostic evidence available. As in the identification of the first two types of misconceptions, the additional information required to evaluate behavioral misconception is available to the tutor. For example, probable evidence against each failure in terms of diagnostic test results is stored within the tutor's knowledge of failures and actual tests conducted by the student are stored in the student model. Thus, by comparison, the tutor can determine if a diagnostic test that suggests the elimination of a hypothesis has been conducted.

After evaluating a student's misconception, an ITS generates instructions to rectify the misconception and to improve the student's diagnostic problem solving skills. The selection of appropriate sets of instructions and their presentation is guided by pedagogical strategies outlined in the instructional module of the ITS.

Instructional Strategies

The instructional module of an ITS contains pedagogical knowledge that specifies how the tutor should respond to various student actions. Many of the instructional modules rely on a rule-based structure to create instructions (e.g., Burton & Brown 1982; Clancey 1987). More recently, (Woolf & McDonald 1984) and (Macmillan, Emme & Berkowitz 1988) have proposed architectures for dynamic instructional planners in adaptive environments. However, in any architecture, the key issues to be addressed are the instructional content, its form and time of presentation.

Instructional content depends upon the instructional objectives. Several units of instruction may be available that satisfy these objectives. Selection of a particular unit of instruction is governed by instructional strategies chosen for the tutor. Such strategies may, under different situations, include preference for hints or discussion of generalities as opposed to solutions or discussion of specifics.

Similarly, the form of presentation may be governed by another set of instructional rules. These rules may specify preference for either graphical or textual mode of presentation under various situations. These preferences may be based on context or norms formulated through experience by human instructors.

Finally, time of presentation of the instructional material is equally critical. There are usually two conditions under which the tutor is expected to deliver instructions. First, when explicit queries are raised by the student. Second, when the tutor infers a student's misconception. In the first case, the response should be immediate. In the second case, the response can be with or without intervention. Instructions without intervention are usually provided at the end of a training session. While non-intervention has some advantages because it does not disturb the student's thought process, intervention at critical stages of diagnostic activity may be an effective way of emphasizing a point.

With respect to tutorial intervention, both the model tracing approach (Anderson, Boyle & Reiser 1985) which calls for intervention as soon as the student's observed actions stray from the normative actions and the issue-based tutoring (Burton & Brown 1982) which encourages intervention at particular occasions may be useful.

We have described an architecture for building intelligent training systems for supervisory controllers in complex dynamic system domains. Figure 3 summarizes the organization of knowledge. However, knowledge organization that captures system structure, function, and behavior, troubleshooting task knowledge, knowledge to evaluate and rectify misconceptions, and instructional strategies are insufficient for the success of a tutoring system. Properly designed interactive interfaces also play a major role in imparting knowledge about the system and its operation during normal and abnormal situations. **Turbinia-Vyasa**, an implementation of the ITS architecture, is described next.

Implementation

The ITS implementation consists of a domain simulator, **Turbinia**¹, and a computer-based tutor, **Vyasa**². Together, **Turbinia** and **Vyasa** constitute an instructional system that trains operators to troubleshoot oil-fired steam-driven marine power plants. **Turbinia-Vyasa** is implemented in Macintosh Common Lisp with Common Lisp Object System and runs on Apple Macintosh II computers. **Turbinia** can simulate operation of a marine power plant under realistic failures.

This section is organized into four parts: (1) a brief description of the domain, including the task; (2) a description of **Turbinia**, the domain simulator; (3) a description of the computer-based tutor, **Vyasa**; and (4) implementation details of both **Turbinia** and **Vyasa**. The description covers the organization of knowledge in the instructional system and implementation details of knowledge

1. Turbines were first applied to marine propulsion by Sir Charles Parsons in 1897. *Turbinia*, an experimental vessel of 100 tons, was fitted with turbines of 2,100 hp driving three propeller shafts. It attained the then record speed of 34.5 knots (A. F. Burstall, 1965, *A history of mechanical engineering*, MIT Press, Cambridge, MA, p.340).

2. Ancient Indian sage, scholar and teacher.

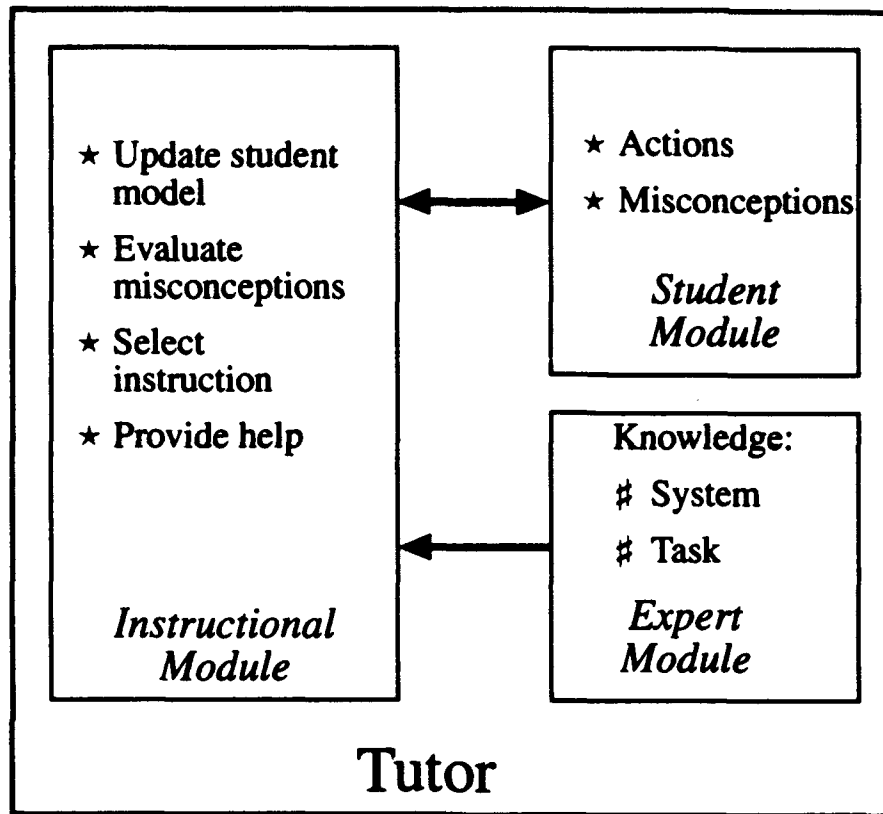


Figure 3. Summary of Knowledge Organization in an ITS

representation.

The domain, the task, and the operator

The domain of **Turbinia-Vyasa** is an oil-fired marine steam power plant. This power plant is installed on naval vessels to produce the power required to propel and operate the ship. Production of mechanical work in a steam-propelled marine power plant can be decomposed into four stages (Gritzen, 1980). Each stage is associated with one of the four phases in the steam cycle: generation, expansion, condensation and feed. Together, the four phases of the steam cycle form a closed loop. Operators must be familiar with these four phases and their interactions for efficient and safe operation.

Most failures in marine power plants, particularly those simulated by **Turbinia**, are usually not catastrophic in nature, and **Turbinia** simulates only such failures. Failures often involve a single malfunctioning component resulting in progressive deterioration of performance. Thus, troubleshooting for a failure involves identifying a single malfunctioning component. Investigations to identify the cause of the failure must begin as soon as symptoms of abnormal behavior are noticed.

Diagnostic problem solving task in a dynamic domain is often difficult due to three main reasons. First, the effects of a failure, in the form of abnormal system states, propagate due to system dy-

namics. Second, even when such a unique relationship can be defined it can only be done for steady state values. However, the system seldom attains a steady state. Third, even if the system attains a steady state it may take a long time to do so. Troubleshooting, on the other hand, must begin immediately, or else it may have possibly irreversible consequences.

The diagnostic task in a real system is further complicated by the operator's inability to observe all abnormal system behaviors. This is due to the limited number of available gauges. This limitation prevents the operator from accessing pressures, temperatures, and flow measurements across every component. Thus, the operator must utilize the available diagnostic information effectively to identify the malfunctioning component.

We have a limited, but pragmatic, pedagogical goal in the design of **Turbinia-Vyasa**. Our objective is to help improve the troubleshooting skills of marine engineers and naval personnel who learn to operate the power plant as a part of their curriculum. A brief description of the knowledge, skills and experience of a typical student who will be trained on **Turbinia-Vyasa** is provided next.

A novice operator in training, although unfamiliar with the faults in the power plant, has a basic understanding of the theory of power generation. The student is usually also familiar with the names and functions of the individual components but does not completely understand the integration of the components into the system, their role in achieving the higher level system goals, their interaction with other components and the dynamics of the system. Very few students are knowledgeable about the detailed structural layout of the power plant even though they may have a general idea about the locations of individual components. Therefore, the students who will use **Turbinia-Vyasa**, in addition to having practical experience and exposure to failure situations, need to consolidate their knowledge of the structure, function and behavior of the power plant and its components.

Next, we describe **Turbinia**, the marine power plant simulator. Detailed description of **Vyasa**, the tutor, follows. Complete details of the implementation and a description of the student's interaction with **Vyasa** can be found in (Vasandani 1991).

Turbinia: the simulator

Turbinia, the marine power plant simulator, was designed using qualitative approximation to represent system dynamics. It can simulate a large number of failures in a marine power plant and provides a good environment for teaching troubleshooting. It is an enhanced version of QSTEAM (Govindaraj 1987) and PEQUOD (Fath, Mitchell & Govindaraj 1990) that is more robust, modular and object-oriented. However, **Turbinia** retains the basic notion of hierarchical representation of components used in the earlier versions.

The primitives that form the basic units of the hierarchy in the simulated system are the simplest form of components performing a single operation or a function, e.g., providing a path for some fluid in the case of a conduit. The primitive hierarchy is shown in Figure 4. In **Turbinia**, approximately 100 components have been modeled to achieve fairly high degrees of structural and dynam-

ic fidelity even though the physical fidelity of the simulator is somewhat low.

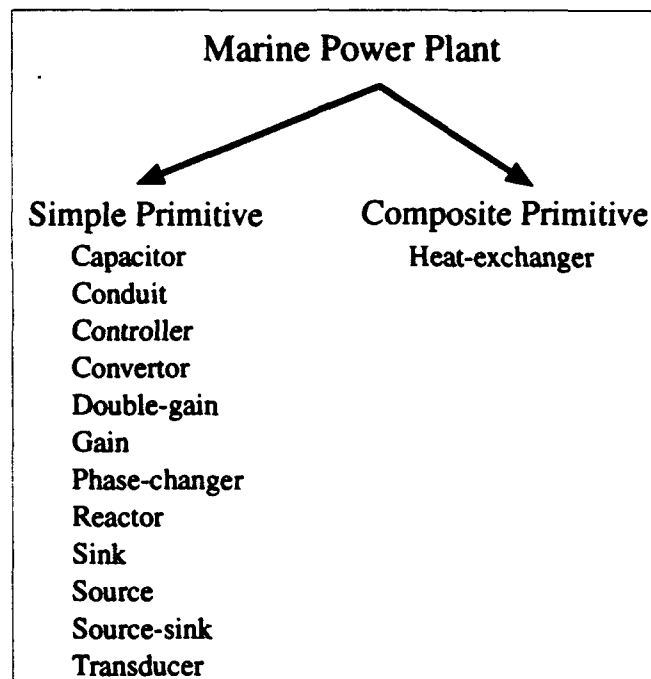


Figure 4. System Hierarchy

Vyasa: the computer-based tutor

Vyasa is the intelligent tutor that trains operators to troubleshoot **Turbinia**. **Vyasa** operates in two modes: passive and active. In the *passive* mode the student is solely responsible for initiating the communications. When the passive tutor is invoked, the simulation is temporarily brought to a halt and the student can access various segments of knowledge in the expert module. In the *active* mode, the tutor takes the initiative to provide instructions when it infers a possible misconception based on the student's actions. The instructions may be provided by the active tutor with or without intervention. The capabilities of the active tutor include all the capabilities of the passive tutor as well.

Vyasa uses the framework for knowledge organization outlined earlier. Knowledge in the tutor is comprised of:

- (1) system knowledge,
- (2) failure knowledge,
- (3) knowledge of student actions,
- (4) knowledge to update the student model,
- (5) knowledge to evaluate misconceptions, and
- (6) instructional knowledge

The expert module of **Vyasa** contains the system and failure knowledge. The rest of the knowledge is contained in the instructional module of **Vyasa**. A complete discussion of **Vyasa**'s knowledge

follows.

System Knowledge

System knowledge in **Vyasa** is comprised of fluid paths, functional subsystems, and schematics (Figure 5). Each fluid path connects components sharing a common fluid in the power plant. Each subsystem is a collection of components responsible for an important system function. Each schematic is a pictorial representation of a section of the power plant. These three constituents of system knowledge are interrelated. For instance, a fluid path may appear in multiple subsystems and schematics. Similarly, a subsystem may contain several fluids and may span over multiple schematics. Also, a single schematic may contain several fluid paths and subsystems. Each constituent of system knowledge is now described in further detail.

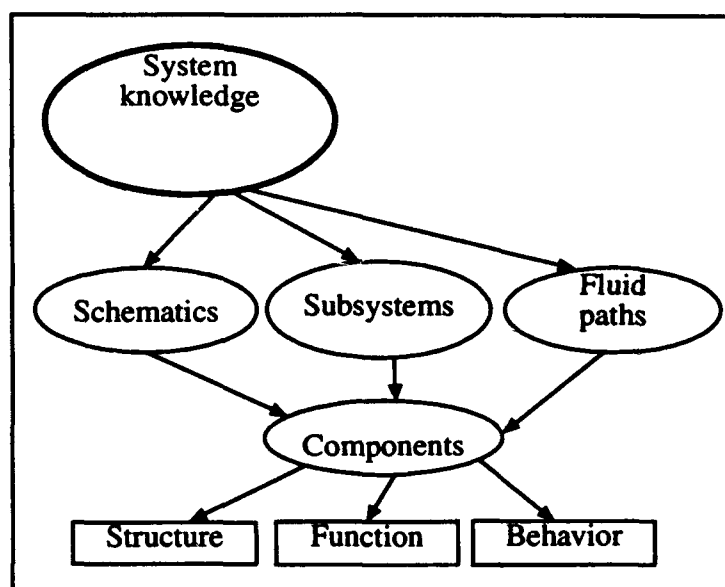


Figure 5. System Knowledge Decomposition

Fluid paths

Representation of a power plant as a collection of fluids is done by decomposition of the system into thirteen fluid paths. They are: combustion air, fuel oil, flue gas, feed water, steam, superheated steam, desuperheated steam, main condenser hot fluid, condensate, main condenser cold fluid, salt-water, control air, and lube oil. Among these thirteen paths, six represent different segments of a single continuous closed loop water path. Water that flows through this closed loop is called feed water prior to entering the boiler, steam at the boiler exit, superheated steam past the superheater, desuperheated steam at the desuperheater exit, main condenser hot fluid in paths leading to the condenser and condensate in the paths feeding to the deaerating feed tank.

The fluid path knowledge in **Vyasa** includes information such as the name of the fluid, and the name of the subsystems and schematics in which the fluid is found. Also included is a list of con-

nectors and components in each schematic that contains the fluid. An example is given in the appendix.

Functional subsystems

Functional decomposition of the power plant is done via nine subsystems. They are: combustion, steam generation, auxiliary steam use, power generation, steam condensation, feed water preheating, lubrication, control air, and saltwater service.

Steam generation, power generation, steam condensation and feed water preheating subsystems are responsible for the four major functions performed in the power plant during the four phases of the steam cycle, viz. generation, expansion, condensation and feed.

The combustion subsystem is responsible for burning the fuel-air mixture to release thermal energy for heating water in the boiler. The auxiliary steam use subsystem is responsible for operating the auxiliary units of the power plant. The interaction between all these subsystems to produce power is summarized in Figure 6.

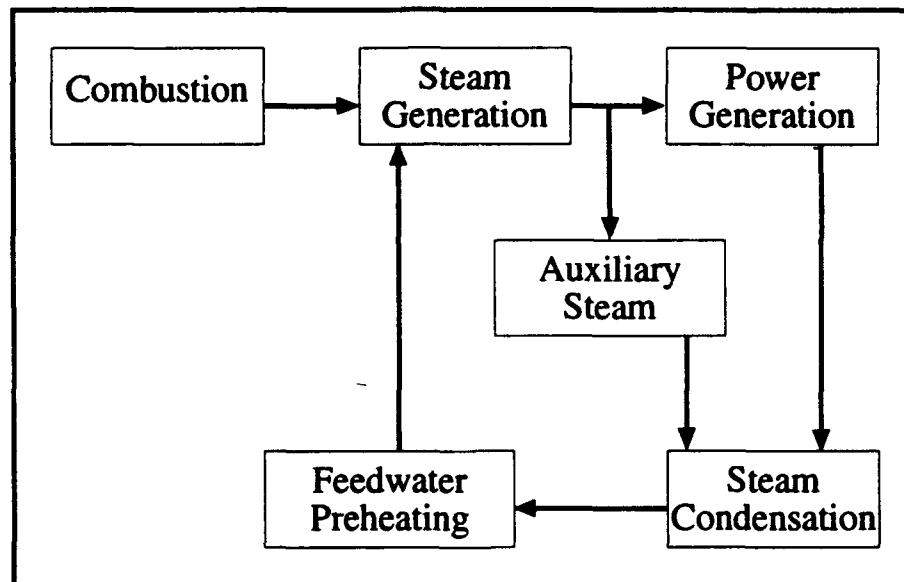


Figure 6. Interacting Subsystems of Marine Power Plant

The remaining subsystems perform other functions necessary for the safe operation of the power plant. The control air subsystem is responsible for distributing control air to many valves and regulators operated by control air. The lubrication subsystem lubricates moving parts and removes the heat produced by friction. The saltwater service subsystem distributes the cold sea water to remove heat from units dissipating heat.

The subsystem level knowledge represented in **Vyasa** consists of the name of the subsystem, its primary function, the names of fluids present in the subsystem and the names of the schematics in which the whole or part of the subsystem can be viewed. In addition, a list of connectors and com-

ponents that constitute the subsystem along each fluid path in each of the relevant schematic is also included.

Schematics

In **Vyasa**, the structural view of the power plant is provided by seven schematics: boiler, steam, feed water, fuel oil, control air, saltwater, and lube oil. Figure 10 illustrates an example. Each schematic contains one or more subsystems and fluid paths. The boiler, steam, feed water, and fuel oil schematics contain the main subsystems responsible for the production of power (shown in Figure 6) and all the six segments of the closed loop water path.

The schematic knowledge represented in **Vyasa** includes information such as the names of the components, subsystems and fluid paths, list of icons displayed, list of graphical objects that represent the components and connectors, and a list of gauges. In addition, the schematic knowledge of **Vyasa** includes information about regions of the schematic that are sensitive to mouse clicks, records of regions picked by the user, and instructions for highlighting or lowlighting the picked region.

Components

System knowledge at the component level concerns a component's structure, function and behavior. In **Vyasa**, the structural, functional and behavioral knowledge of components is organized at the level of detail necessary for the troubleshooting task. A component's structural knowledge refers to its input and output connections to other components, fluids carried by it, gauges attached to it and its association with a functional subsystem and schematic. The functional knowledge is a description of the purpose of the component in the system and its contribution to the higher level system function. The component's behavioral knowledge describes the manner in which the system state values are affected by the presence of the component in both the normal and failed states.

In addition, the system knowledge at the component level includes the modes in which the component can fail, the parameters that are used to compute the component's behavior in the failed state, and the component's link to the graphical object that represents it on the schematic interface.

Failure Knowledge

The failure knowledge in **Vyasa** is organized in terms of modes of failure and specific failures. Specific failures are faults simulated by **Turbinia**. Knowledge of failure modes helps the tutor to teach the student about typical abnormal behaviors associated with each mode. Knowledge of specific instances of failure helps the tutor in evaluating student's actions. Both components of failure knowledge are described below.

Modes of Failure

Vyasa has knowledge of four most common modes of failure for components of **Turbinia**: (a) blocked-shut, (b) stuck-open, (c) leak-in, and (d) leak-out. Knowledge of each mode of failure includes the typical system behavior associated with it, possible reasons for deviations from expected

abnormal behavior, and names of components in the system that are known to fail in that mode.

Knowledge of blocked-shut mode of failure, for instance, includes the upstream and downstream abnormal system behaviors expected in liquid and gas paths, the type of component that can curtail the propagation of abnormal behavior, and the names of the component in the power plant that are commonly known to fail in the blocked-shut mode. A summary of typical system behavior and the conditions that curtail the propagation of abnormal behavior for each of the four modes is shown in Table 1.

Table 1. Typical Abnormal System Behavior

Failure Mode	Fluid	State	Abnormal Behavior		Propagation limited by	
			Upstream	Downstream	Upstream	Downstream
Blocked shut	Liquid	Level	High	Low	Sink	Source
	Gas	Pressure	High	Low	Safety Valve	Source
Stuck open	Liquid	Level	Low	High	Source	Sink
	Gas	Pressure	Low	High	Source	Safety Valve
Leak in	Liquid	Level	High	High	Sink	Sink
	Gas	Pressure	High	High	Safety Valve	Safety Valve
Leak out	Liquid	Level	Low	Low	Source	Source
	Gas	Pressure	Low	Low		

Specific Failures

Knowledge of specific failures includes initial symptoms, cause and mode of failure. In addition, the subsystems, fluid paths, schematics, components and gauges affected by each failure are included in the tutor's knowledge of specific failures.

Vyasa also has access to "pre-defined" explanations of cause-effect associations that describe the propagation of abnormal gauge readings under each failure condition. These explanations, along with diagnostic tests that serve as evidence for or against the specific failures, form an essential part of the tutor's failure knowledge. A detailed example is given in the appendix.

Knowledge of Student Actions

A computer-based tutor must have some means of evaluating the intended purpose of student's actions. This capability enhances the tutor's ability to infer misconceptions. **Vyasa** infers student's misconceptions based on student's actions using its knowledge of the valid forms of interactions at the student-tutor interface.

A student interacts with **Turbinia-Vyasa** in three modes: troubleshooting mode, tutor dialog

mode, and diagnose mode. In the troubleshooting mode the student interacts with the simulator only. In the tutor dialog mode, the interaction is with **Vyasa**. In the diagnose mode, the student attempts to identify the failed component.

In all the three modes of interaction, **Vyasa** recognizes nine types of valid actions: call-for-schematic-action, investigative-action, informative-action, diagnose-request-action, diagnostic-action, help-request-action, resume-request-action, tutor-dialog-action and modal-dialog-action. A *call-for-schematic-action* is performed to call a new schematic or switch between schematics. An *investigative-action* is performed to view the gauges attached to a component. An *informative-action* usually follows the investigative-action and is taken to display the gauge readings. A *diagnose-request-action* is taken to switch to the diagnostic mode. The *diagnostic-action* is the action of identifying the failed component. The *help-request-action* and the *resume-request-action* switch the student to the tutor dialog and the troubleshooting modes respectively. *Tutor-dialog-actions* are all actions taken in the tutor dialog mode. Finally, *modal-dialog-actions* terminate interactions with dialogs.

Vyasa keeps a record of all recognized actions in a student model. This record is updated and used to determine student's misconceptions after every action.

Knowledge to Update the Student Model

The student model in **Vyasa** is a dynamic data structure that maintains a record of actions taken by the student. Each student action, if recognized as one of the nine types of valid actions, is time stamped and information relevant to the action is stored. In this manner, the student model keeps account of the schematics viewed, the order in which they were viewed, the sequence of subsystems and fluid paths explored, the components investigated, the gauges probed and their gauge readings at the time of the investigation.

After each student action, the subsystem and the fluid path most suspected by the student is determined. A count is kept of the number of investigations made in each subsystem and fluid path. The subsystem and the fluid path with the maximum number of investigative-actions are also the most suspected if at least one of the last three investigations have occurred in that subsystem or fluid path. Otherwise, the most suspected subsystem or the most suspected fluid path is the one investigated last. Thus, after every action, the information concerning the most suspect subsystem and the most suspect fluid path in the student model is revised.

In addition to recording the actions, the student model maintains a list of the student's past and current hypotheses concerning the failure. This information is a list of components with their respective modes of failure, which, in the opinion of the student, explains the observed abnormal system behavior. This information on student's failure hypotheses is directly elicited from the student when **Vyasa** functions in the active mode.

Knowledge to Evaluate Misconceptions

The knowledge to evaluate misconceptions gives **Vyasa** the ability to deliver individualized in-

structions. This knowledge has a rule-based structure. These rules are used to identify three types of misconceptions based on observed student actions.

The first misconception concerns deficiency in student's knowledge of system structure. **Vyasa** identifies this structural misconception when the student investigates components in a schematic unaffected by the current failure. Knowledge of schematics affected by each failure, needed to evaluate the structural misconception, is obtained from the tutor's knowledge of the failures.

The second misconception concerns deficiency in student's knowledge of system functions. **Vyasa** identifies this functional misconception when the most suspected subsystem or fluid path inferred from the student's action is unrelated to the failure being investigated. Knowledge of subsystems and fluid-paths related to each failure, needed to evaluate the functional misconception, is obtained from the tutor's knowledge of the failures.

The third misconception concerns deficiency in student's knowledge of fault related system behavior. **Vyasa** identifies this behavioral misconception when a student continues to pursue a failure hypothesis that should have been rejected based on the diagnostic evidence available. As in the identification of the first two types of misconceptions, the additional information required to evaluate behavioral misconception is available to the tutor. For example, probable evidence against each failure in terms of diagnostic test results is stored within the tutor's knowledge of failures and actual tests conducted by the student are stored in the student model. Thus, by comparison, the tutor can determine if a diagnostic test that suggests the elimination of a hypothesis has been conducted.

After **Vyasa** has evaluated a student's misconceptions, it uses its knowledge to deliver instructions to rectify the misconceptions. Knowledge concerning delivery of instructions is described under instructional knowledge.

Instructional Knowledge

Instructional knowledge concerns instructional content, its form and time of presentation. The instructional content is either extracted verbatim from the tutor's knowledge base or is generated using a template. The form of instructional presentation is either textual or graphical. The time of presentation of instructional material depends on the context and the student actions. Knowledge related to content, form and time of instructional presentation is discussed in further detail below.

Content

Information provided to the students by the tutor comes from units of instructional sets prepared in advance. However, the details to be inserted in the instructional sets is often context-driven. For example, when a student inquires about the behavior of a component, the information is always presented in the same format on a dialog box. It consists of relationships between input and output states of the component. Details of the input-output relationships depend upon the functional primitive that represents the component. Thus, although the details of the information presented are context-specific, they are extracted verbatim from the tutor's knowledge base.

The instructional template used for advice generation is the same on all occasions for the same type of advice. In all there are four such instructional templates used by the tutor. Each template is used for a different type of advice. One is used for suggesting hypotheses revision (see Figure 7), another to indicate lack of adequate evidence to pursue a hypothesis, a third to suggest a diagnostic test to strengthen or weaken a hypothesis; and a fourth to convey the tutor's inability to provide advice under existing conditions.

(Suspected Failure Mode)	(Suspected Component)
(Suspected Component) has probably not failed in a (Suspected Failure Mode) mode because the (Gauge) on (Component) shows a (Qualitative State Value) reading.	
OK	

Figure 7. Example of an instructional template

Form

Like content, the form of instructional presentation is also context-dependent. For example, answers to queries related to subsystems and fluid paths that can be visualized on the simulator interface are presented graphically. Where graphics is unlikely to enhance the understanding of the instructions or where graphical aid is computationally expensive the instructions are presented in the form of text. Therefore, when the student wants to know the components that constitute a particular subsystem, showing these components by highlighting them in schematics is preferred over listing the names of the components in a dialog box.

Form of presentation include more than just the graphics and/or text. There are certain instructions that convey an error message or require immediate attention. When these instructions are displayed, they are accompanied by a beep. Instructions of this type must be acknowledged before the student is permitted to proceed further. Such instructions disable the mouse, until the student performs a specified action.

Generation of help and instructions

Time and Duration

Apart from instructional content and its form of presentation, **Vyasa** has knowledge about time and duration of instructional presentation. Instructions that are given on request from the student are provided immediately in a dialog box or in the special tutor communication window. These instructions are displayed for an unspecified duration of time until the student makes a new request. In some cases, however, the student is required to acknowledge the receipt of instructions before making a new request.

In addition to the instructions presented on request, **Vyasa** delivers instructions on its own. **Vyasa** delivers these instructions with and without intervention depending on the context.

Instructions with Intervention

When the tutor identifies a misconception, instructions to rectify the misconceptions are presented immediately with a beep. The tutor has different sets of instructions for each of the three types of misconception it identifies. The instructional sets for structural and functional misconceptions inform the student that a region unaffected by the current failure is being investigated unnecessarily. The idea is to suggest that the student only investigate portions of the system relevant to the failure. Similarly, when behavioral misconceptions are identified, the student is instructed to refine the failure hypotheses based on the evidence available.

Since the instructions to rectify misconceptions are highly context-sensitive and may change with every new action taken by the student, they need to be presented only as long as they retain their context-sensitivity.

Instructions without Intervention

Instructions are also provided without intervention at the end of the training session. These instructions include explanations of abnormal gauge readings for the failure investigated during the session. These are causal explanations that animate the effects of fault propagation and are presented in a chronological order. For each failure, these explanations are stored as a pre-defined set in the failure knowledge of **Vyasa**.

This completes the description of the components of knowledge in **Vyasa**. Specific implementation details, with representative examples, are provided in the appendix. Complete details can be found in (Vasandani 1991). Coordination and control of different components of knowledge in **Turbinia-Vyasa** are discussed next.

Control and coordination

In **Vyasa**, the knowledge concerning evaluation and rectification of student's misconceptions is represented as rules. These rules are organized in the instructional module of the tutor in several units called *knowledge sources*. There are knowledge sources for recognizing and understanding student actions, updating the student model, evaluating misconceptions and presenting instructions to rectify the misconceptions. The overall tutoring objective of **Vyasa** when operating in the active mode is achieved by the coordinated efforts of these individual knowledge sources.

A blackboard-like control architecture (Hayes-Roth 1985; Nii 1986) coordinates the modules that contain various components of knowledge sources and performs high level control and planning of pedagogical functions. It consists of a blackboard object and several rule-based knowledge sources that can access information posted on the blackboard and make changes to it (Figure 8). The knowledge sources are invoked when preconditions necessary to activate them are posted on the blackboard. Together, the blackboard and the knowledge sources play an important role in helping **Vyasa** evaluate and provide help to rectify student misconceptions.

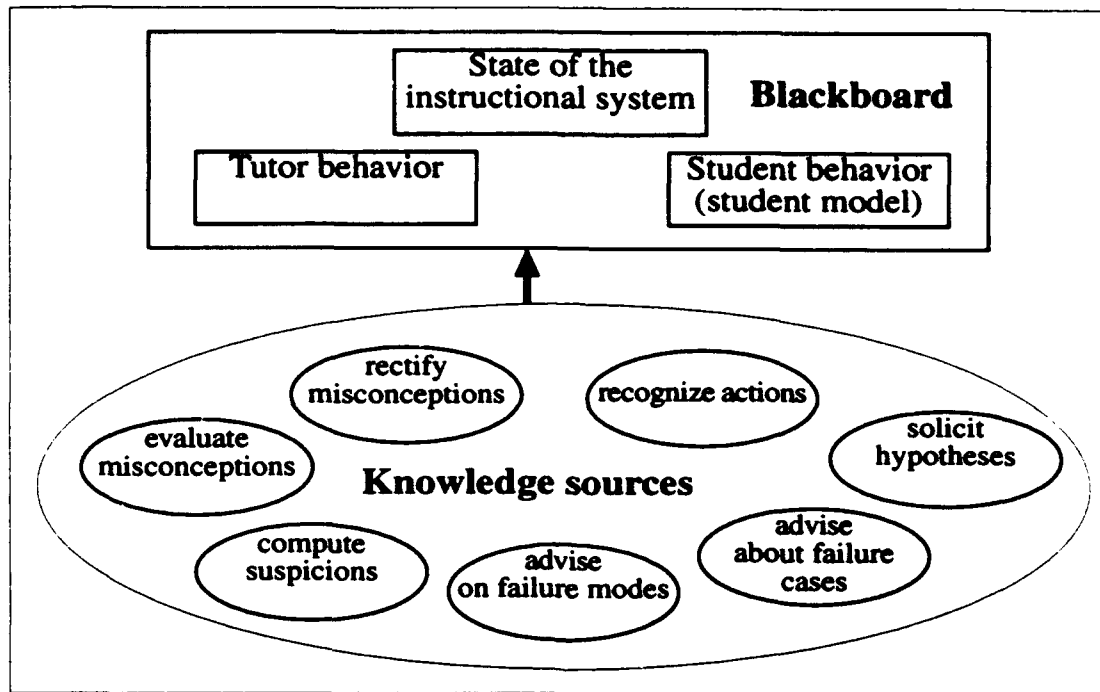


Figure 8. The Blackboard Control Architecture

The blackboard stores information concerning the state of the instructional system. The state of the instructional system is defined in terms of the tutor mode, the state of the simulation, the current displays, time spent by the student in the different modes of interaction and the pending events. The blackboard also stores complete information related to the student's last action and historical information necessary to determine the context for current and future action. Information concerning the system response to the student action is also recorded.

In addition, the blackboard captures the dynamic evolution of the tutor and student behavior. While most of the student behavior evolves dynamically, some of the tutor behavior is derived from the tutor's knowledge of the failures. The information obtained from tutor's knowledge of the failures remains unchanged for the problem solving session. This information concerns current failure and is used by the tutor to evaluate student performance.

The information posted by the knowledge sources is dynamic and concerns the student. It includes information solicited by the knowledge sources from the student such as the student's initial hypotheses and current hypotheses. It also includes summary of hypotheses refinement, evidence against current hypotheses, and the most suspected subsystem and fluid path as inferred from student actions. In addition, the existing misconceptions of the student and those rectified during the current session are posted by knowledge sources that evaluate and rectify misconceptions. A record of the actions taken since the various types of misconceptions were last identified is also maintained. Additional information concerning actions taken by the student is stored and updated dynamically by knowledge sources after every student action.

Several knowledge sources use the blackboard as a globally shared database and often compete with each other to modify information on the blackboard. Some of these knowledge sources also determine and execute the appropriate pedagogical functions of the tutor based on the current status of the instructional system. Two of the knowledge sources (Figure 8) help the students with their failure hypothesis.

The first of these knowledge sources is invoked when help is sought for failure hypothesis that matches one of the specific cases of failure known to **Vyasa**. For such hypothesized failures, knowledge concerning affected gauges is available to the tutor. This knowledge is used to determine whether the hypothesized failure is probable based on the observed symptoms. If so, the tutor suggests more tests to strengthen the student's belief in the hypothesis. If not, the tutor suggests tests to weaken the student's belief in the hypothesis. Also, since the knowledge source has access to the information displayed on the blackboard, it can determine if evidence has already been gathered to eliminate suspicion from the hypothesis. If so, the student is advised to drop the hypothesis from the list of suspected components.

The second knowledge source is invoked when the knowledge of the hypothesized failure is unavailable to the tutor. Instead, knowledge of the general modes of failure is used by the knowledge source to generate advice. For example, if the hypothesized failure involves blocked shut mode of failure, the knowledge source first determines the fluid path through the suspected component. Then, using its knowledge of modes of failure it determines the expected abnormal behavior associated with the fault. Next, using the knowledge of available gauges, the knowledge source determines if the expected abnormal behavior is observable under the current situation. If so, it suggests checking the relevant gauges to strengthen or weaken the student's belief in the hypothesis.

Summary

We discussed **Turbinia-Vyasa**, an implementation of the ITS architecture. The marine power plant domain, salient features of the troubleshooting task, and the educational background of the student trainees were discussed. Next, organization of knowledge in **Turbinia-Vyasa** was described. The discussion of knowledge organization included a description of the various components of knowledge represented in the instructional system. Finally, a control architecture was described that is used by the instructional system to plan the pedagogical functions of the tutor.

Complete details of the implementation can be found in Vasandani (1991). The interface of **Turbinia-Vyasa** and details of student interaction with the instructional system are described next.

Interactive Interfaces and Student-Tutor Interaction

The student-tutor interface of **Turbinia-Vyasa** has been developed on a dual screen Apple Macintosh II workstation. The configuration consists of a 19" color monitor on the left and a 13" color monitor on the right (Figure 9). A single button computer mouse that can point to all locations on both screens is used for input. All actions at the interface involve moving the mouse cursor to a desired location and clicking once on the mouse button. All valid user actions have appropriate re-

sponse while invalid actions are ignored by the system.

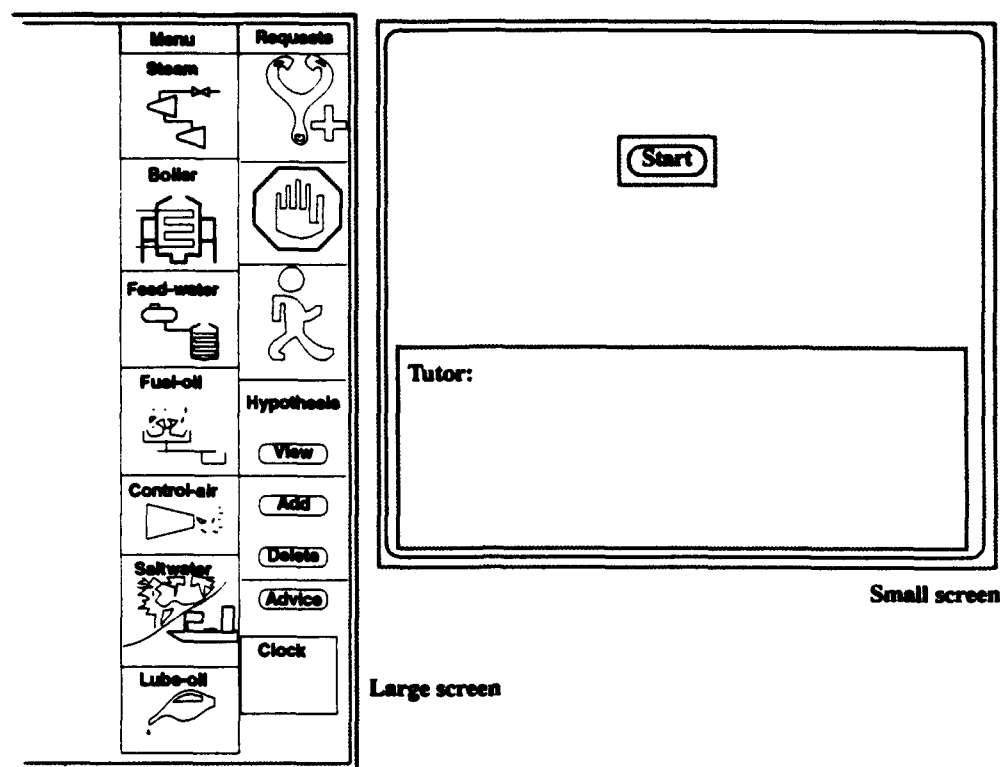


Figure 9. Screen configuration

The joint interface to **Turbinia-Vyasa** consists of an interface to the simulator **Turbinia**, and dialogs to interact with **Vyasa**. **Turbinia**'s interface consists of seven schematic windows, a schematic menu, a requests menu, a symptom dialog, several error dialogs and a clock. Student interaction with **Vyasa** is accomplished by multiple levels of hierarchically organized passive tutor help dialogs and a hypothesis menu.

Interaction with Turbinia-Vyasa

At the beginning of every training session, the large screen displays three menus and a clock. The three menus are: the *schematic menu*, the *requests menu* and the *hypothesis menu*. A *tutor dialog* is displayed on the bottom edge of the small screen. For sessions where the active mode of the tutor is not invoked, the *hypothesis menu* under the *requests menu* is not displayed. The student interacts with **Turbinia** through the seven schematics that display the physical connections between components of the power plant. (An example schematic, slightly altered from the actual display for improved clarity, is shown in Figure 10.) These schematics can be accessed by clicking on the icons in the schematic menu. When the student clicks on a displayed gauge to probe its reading, an icon appears near the gauge. This icon is a qualitative representation of the current gauge reading, which is either low, slightly low, normal, slightly high or high.

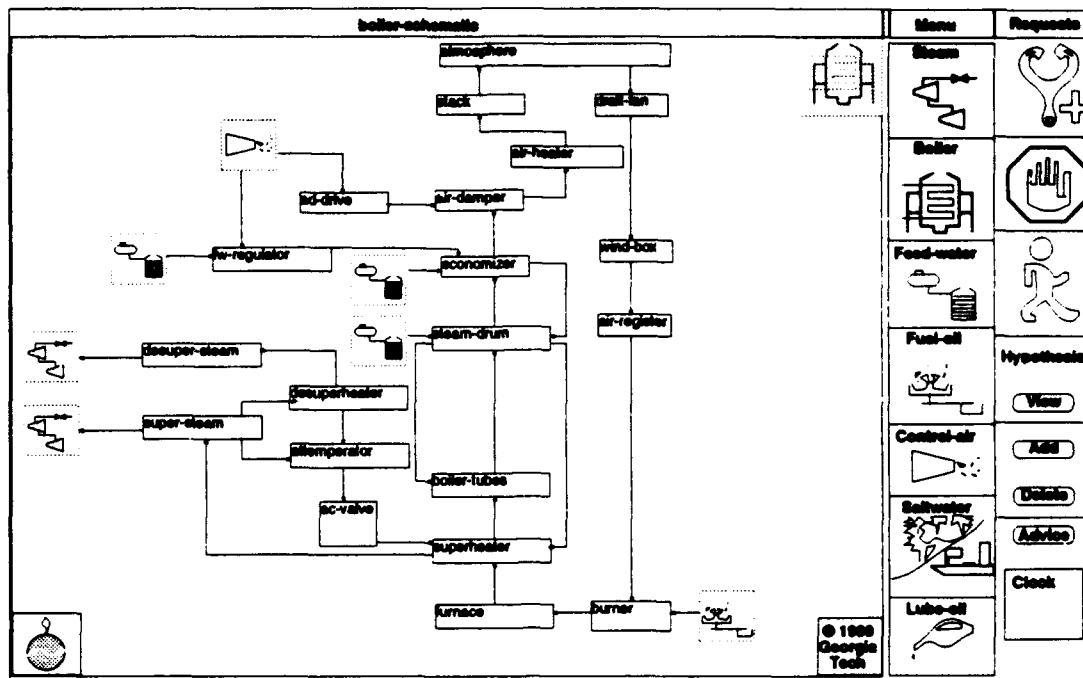


Figure 10. Boiler schematic

Interaction with Turbinia

Each icon in the schematic menu represents a different schematic. The same schematic icon that is used to access the schematic also appears in the top right corner of the schematic it represents. The features of the schematic interface of **Turbinia** are discussed next, mainly with reference to boiler schematic.

When the student clicks on a displayed gauge to probe its reading, an icon appears near the gauge. This icon is a qualitative representation of the current gauge reading. **Turbinia** uses five different qualitative representations of state values. These five are low, slightly low, normal, slightly high and high; each is represented by an icon as shown in Figure 11.

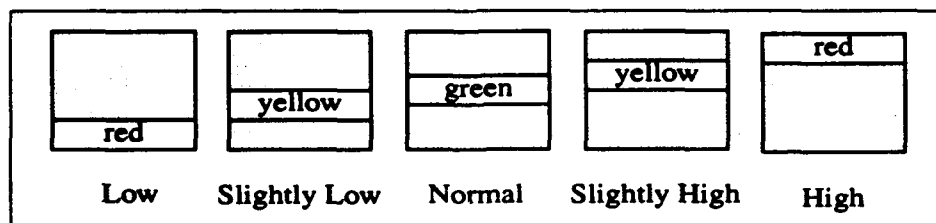


Figure 11. Qualitative state representation

Interaction with Vyasa

Passive Mode

When **Vyasa** operates in the *passive* mode, the student is responsible for initiating communica-

following the evaluation of a student's misconception. For instance, if the student investigates schematics, subsystems, or fluid paths unaffected by the failure, the tutor delivers the appropriate instructions to guide the student away from unaffected portions of the power plant (Figure 13). Such instructions are usually displayed for a fixed, but short, period of time since the instructions lose their context due to system dynamics and student actions.

Tutor: You seem to be investigating a schematic unaffected by the current failure

Figure 13. Example of instructions from Vyasa

The tutor in the active mode is also capable of helping the students with their hypotheses. Student's hypotheses concerning failures are either solicited by the tutor or voluntarily disclosed by the student. In either case, the manner of communicating the failure hypotheses to the tutor is identical. First, the tutor asks the student to select the suspected component responsible for the current abnormal system behavior. After the student selects a component, she is prompted to identify the failure mode. The student can use the hypothesis menu to *add* or *delete* hypotheses, *view* the hypotheses that are currently active, and ask for *advice* concerning any active hypothesis.

The student interacts with the instructional system at the end of every problem solving session to view the solution to the problem last presented. The solution and the interaction with the instructional system depends upon whether the student was using only the simulator or was also aided by the tutor. Students using just the simulator see only the solution as shown in Figure 14. Students aided by the tutor have the option of viewing the explanation for each observed abnormal behavior. The explanation containing causal reasons for each abnormal gauge reading are presented, one at a time, and in the order in which the gauges are affected by the failure. In presenting the reasons for each abnormal gauge reading, first the schematic which contains the affected gauge is displayed, then the affected gauge along with its gauge reading are made visible and finally an explanation for the abnormal reading is displayed in a dialog box on the small screen (Figure 15). Once the student has read the explanation and clicked on the "OK" button, the tutor proceeds to provide a similar explanation for the next affected gauge. This process continues until the tutor completes providing an explanation for each abnormal gauge reading caused by the failure.

This completes a description of the student-tutor interface of **Turbinia-Vyasa** and the valid forms of operator interactions at this interface. Additional details are given in (Vasandani 1991; Vasandani & Govindaraj 1993b). An experimental study to evaluate the ITS architecture implemented in **Turbinia-Vyasa** is described next.

Cause Feed-water regulator is stuck closed	Affected Subsystems (feed-water-purheating-subsystem steam-gas-exhaust-subsystem combustion-subsystem power-gas-exhaust-subsystem)	Affected Fluid Paths (feed-water fine-gas combustion-air fuel-oil superheated-steam desuperheated-steam steam main-condenser-hot fluid main-condenser-cold fluid)	Affected Schematics (steam-schematic boiler-schematic feed-water-schematic fuel-oil-schematic)
Symptom When speeding up the ship, boiler level drops low			

Figure 14. Solutions for students trained on simulator

The speed of the ship is increased by increasing the mass flow rate of steam to the turbines. When the steam demand from the boiler increases, the steam pressure in the drum decreases. This sets the boiler combustion control mechanism into operation. The job of the combustion control mechanism is to increase the quantity of combustion air and fuel to the boiler. Also, when the mass flow rate of steam from the boiler is increased, the boiler feed water control mechanism has to adjust the flow rate of feed water to maintain a mass balance of flow into and out of the boiler. When the feed water regulator is stuck, the feed water control mechanism is unable to increase the feed water flow rate. Such a failure may be regarded as an example of a blocked shut valve. Because the feed-water-regulator does not permit an increase in the feed water flow rate, the upstream water level in the deaerating-feed-tank rises above the normal value.

OK

Figure 15. Explanations for abnormal system behavior

Experiment

In the experiment, we investigated the performance of subjects trained with and without the tutor. There were two goals: (1) determining the effectiveness of the tutoring architecture and methods for knowledge representation and (2) establishing the usefulness of computer-based training programs over traditional means of training operators to troubleshoot complex dynamic systems. In addition, the experiment provided an opportunity to compare the effect of passive and active tutoring strategies.

Thirty paid volunteers, who were students at Georgia Institute of Technology and cadets with the Naval Reserve Officers Training Corps unit, participated as subjects. All except one subject were male. Subjects had a basic understanding of the theory of marine power plants. Among those selected for the experiment were twenty-four sophomores and six juniors. A few of the subjects had additional exposure to thermodynamics through course-work or had limited experience operating the marine power plants.

For every session completed in both the training and the testing phase each subject was paid \$6 per session. In addition, an award of \$25 was promised for the best troubleshooter in each of the three groups based on performance in the two data collection sessions. All subjects were told about the performance measures prior to the experiment. They were also informed that for the purpose of determining the award, the number of problems correctly diagnosed in minimum time was the only

measure to be considered.

Subjects were randomly assigned to the three experimental groups. The experiment consisted of two phases: training and data collection. In the training phase, subjects were exposed to one of the three instructional methods: (a) training on simulator alone (S); (b) training with the aid of a passive tutor (P); and (c) training with the aid of an active tutor (A). During data collection, trained subjects from all three conditions attempted to solve the same set of problems unaided by the tutor. The effect of training was then evaluated in the data collection phase where all subjects were exposed to identical problems on **Turbinia** without the aid of the tutor.

There were ten training sessions, each lasting no more than forty-five minutes. The sessions were run on consecutive days with typically one session per day. Occasionally, when a subject missed a day, the lost session was made up by extending the training period by a day. Under no circumstances was a subject permitted multiple sessions in a day.

The first training session for each group introduced the system using a single problem. Audio taped instructions, different for each group, were used during this session. These instructions introduced the subjects to the interface and valid forms of interactions.

After the first session, subsequent training sessions had three problems each. A subject had thirteen minutes to solve each problem. If the subject solved the problem in less than the allotted time, the next problem was immediately presented. Thus, if the subject solved one or more of the three problems in a session within the allotted time for each problem, the session could potentially be completed in less than forty-five minutes.

At the end of each problem the subject was provided the solution. While solutions presented to subjects with the tutor were accompanied by an explanation, no such explanation was provided to subjects using the simulator alone.

The data collection phase consisted of two sessions. These sessions were run on consecutive days immediately following the completion of training. During these sessions, the subjects interacted with the simulator only, unaided by any tutor, irrespective of their training condition. Thus, even the subjects in Groups P and A who were earlier aided by the tutor were unaided during the data collection sessions.

Each data collection session was approximately fifty minutes long and consisted of five problems. If the subject solved the problem within the ten minute time period allocated for each problem, the next problem was immediately presented. However, unlike the training sessions, no solution was provided to the student at the end of the problem. At the end of the data collection sessions, all subjects completed an exit questionnaire.

Results

The data were analyzed using the SAS General Linear Model and Type III sum of squares. The effect of training condition on the performance of the subjects is summarized Table 2. A brief dis-

cussion follows.

Table 2. Summary of Training Condition Effect

Performance Measures	Training Condition			Performance Comparison ($\alpha=0.05$) * Significant at $\alpha=0.1$
	Simulator (S)	Passive Tutor (P)	Active Tutor (A)	
Product Measures				
Percentage of problems solved	93.00	95.00	88.00	Not significant
Troubleshooting time (minutes)	2.62	3.43	3.69	Not significant*
Process Measures				
Number of informative actions per problem	10.72	8.18	8.83	Not significant*
Percentage of relevant informative actions	59.70	72.50	71.50	(S) < (P), (A)
Percentage of guesses	71.40	35.23	29.50	(S) > (P), (A)
Investigations (per problem) in unaffected				
Schematics	0.36	0.81	1.81	Not significant* (S) > (P), (A) (S) > (P), (A)
Subsystem	0.12	0.40	1.00	
Fluid-paths	0.23	0.35	0.98	
Nature of diagnosis (% of solved problems)				
Premature	26.80	53.70	19.35	(S) > (P), (A)
Timely	14.73	81.00	4.20	(S) < (P), (A)
Overdue	9.00	85.22	5.60	(S) > (P), (A)

There was no significant difference in the *number of problems solved* across the three training conditions. The relatively poor performance by subjects in Group A can be attributed to three factors. First, a single subject was responsible for five of the unsolved problems. Second, subjects in Group A were more inclined to leave a problem unsolved because they were reluctant to guess the failures. Third, the subjects in this group became somewhat dependent on the tutor to solve the problems and when the tutor was withheld from them, during the test sessions, their performance deteriorated.

The *troubleshooting time* was also not significantly different. This result is not at all surprising considering that the unaided group did not have a guided strategy to solve the problems and relied heavily on guessing. Guessing as opposed to abstract reasoning takes less time. However, at α level of 0.1, the effect of training condition on troubleshooting time was significant.

Even though the *number of informative actions* was statistically not significant, the data indicate that the subjects in Group S, in comparison to the subjects in other two groups, needed more diagnostic tests to solve the problems. In other words, subjects in Groups P and A utilized the diagnostic information more effectively and required smaller number of diagnostic tests to solve the problems.

The *percentage of relevant informative actions* taken by the subjects in the two aided groups was significantly higher, implying that those trained by the tutor were better able to identify the diagnostic tests that were useful for solving a problem.

The effect of training condition was significant with higher *percentage of guesses* for the unaided group in comparison to the two aided groups. Evidence of guessing strategy was noticed in 60% of the problems for Group S and only 39% and 30% of the problems for Groups P and A respectively. Also, the data indicate that the subjects in Group S often used guessing as a primary strategy whereas the subjects in Groups P and A started guessing only when they were running out of time.

Detailed analysis of the *number of unaffected schematics/subsystems/fluid-paths investigated* showed that subjects in the two aided groups performed significantly fewer investigations in unaffected subsystems and fluid paths. In other words, subjects in the two aided groups were able to better identify the location of the fault and investigate the relevant portions of the power plant.

The usual SAS analysis of variance was not possible for *the nature of diagnosis*, which consisted of comparing the three mutually exclusive categories of correct diagnoses (premature, timely and overdue) from each training group. Therefore, pair-wise comparisons were performed to detect significant differences across the three training conditions. Of the problems solved, subjects in Group S performed more premature diagnoses as compared to subjects in Groups P and A. Since the subjects in Group S relied rather heavily on guessing, it is not surprising that they got lucky more often. The results suggest that the subjects in the two aided groups either formed a better understanding of cause-effect associations or utilized it more effectively to diagnose faults. Also, for subjects in Group S, more diagnoses were overdue as compared to the two aided groups. This shows that the subjects in Group S were not as good at integrating diagnostic information as the subjects in the two aided groups.

From the results presented above and additional data analysis (Vasandani 1991), it was apparent that the tutor in both the passive and the active modes helped the students to develop useful troubleshooting strategies. Those trained by the tutor formed plausible failure hypotheses based on observed symptoms and systematically eliminated them by conducting appropriate diagnostic tests. In comparison, those trained without the tutor did not develop good troubleshooting strategies. They relied rather heavily on guessing the solution. Furthermore, the tutor helped the students to recognize and integrate crucial diagnostic information in a timely manner that the students without the tutor were unable to do. Students trained by the tutor were better-prepared for unfamiliar situations than those trained on the simulator.

The data also indicated that the effectiveness of a tutoring strategy depended upon the individual student. For example, the strategy of providing explanations for all observed symptoms for each problem was intended to help the students develop a proper causal model of fault propagation. Some students who learned to map salient symptoms to causes from these explanations became overly conservative. During troubleshooting they spent a lot of time eliminating all probable hypotheses linked to an observed symptom even when sufficient evidence in support of a highly probable hypothesis had been collected. Another tutoring strategy adopted by the active tutor was to provide help in building, refining, and eliminating failure hypotheses. In this capacity the active tutor came to be perceived by some students as an on-line associate. These students often took the

help of the active tutor to refine their failure hypotheses and thus became dependent on the tutor to solve problems. Performance of these students deteriorated when the active tutor was withdrawn.

Experimental results show that a simulator alone is inadequate for training purposes. However, a simulator in conjunction with an effective computer-based tutor can help develop efficient troubleshooting skills. Such a tutor must teach operators to identify useful diagnostic tests, use the results of these tests to formulate plausible hypotheses concerning failure, and systematically refine the hypotheses based on new diagnostic data until the cause of failure is identified. Operators trained by such a tutor are likely to rely less on guessing and more on abstract reasoning. Consequently, these operators are likely to provide incorrect diagnoses less often.

In real-world, where there is a cost associated with each incorrect diagnosis, less incorrect diagnoses can save valuable time and reduce troubleshooting costs. However, since not all students are equally receptive to every tutoring strategy, provisions must be made in training programs for individual preferences and differences in abilities and styles. Otherwise, students may become overly conservative or too dependent on the tutor for help. While conservative behavior may not necessarily be bad, too much dependence on the tutor is undesirable. Therefore, assistance provided by the tutor that directly helps students in solving the problems must be avoided to control the students' dependence on the tutor.

Also, use of certain features of the tutor, like hypothesis aiding, may be useful in other applications such as an on-line operator's associate. Possibility of success of hypothesis aiding in an on-line operator's associate application was indicated by the performance data of students who exploited this feature of the active tutor to successfully solve problems during training. They frequently provided the tutor with failure hypotheses and sought advice on each one of them. As a part of its counseling task, the tutor would check if evidence had already been gathered to reject the hypothesis, and if so, the student would be told about it. Thus, the students in fact assigned the tutor the task of filtering out the less likely alternatives and based on the tutor's advice refined their hypotheses till they could identify the failed component with reasonable amount of certainty.

Conclusions

A major impediment to successfully extending research results in ITS to real-world systems stems from a lack of suitable methodology for knowledge organization. We have developed a framework for organizing knowledge that should help remedy this situation. We described an architecture in which system and task knowledge are organized in a coherent manner to facilitate rapid construction of ITSs for complex dynamic systems. The framework decomposes system, task, and pedagogical knowledge for teaching diagnostic problem solving task in a marine power plant domain.

We have implemented a prototype instructional system, **Turbinia-Vyasa**, and evaluated it experimentally. Experimental results support the viability of designing and implementing effective tutoring systems for complex dynamic system domains. Instructional systems that integrate intelligent tutors with a simulator and provide access to multiple, complementary, system represen-

tations via direct manipulation graphical interfaces can contribute greatly to an effective training program. In our current research, we are developing a family of models to represent the knowledge about the system, tasks, and problem solving strategies employed by operators with different levels of expertise. We are studying both continuous and discrete event dynamic systems, including manufacturing systems and computer networks. Results of this research should lead to the development of enhanced architectures for computer-based intelligent systems to assist operators in control and coordination tasks that require different skill levels.

Acknowledgments

The research reported here has its roots in work sponsored by a previous grant from the Office of Naval Research (ONR). Drs. Marshall Farr and Henry Halff at ONR realized the need to study troubleshooting in complex, real-world domains, and provided financial and moral support for our work. Later, Drs. Michael Shafto and Susan Chipman saw the need to continue this support via contract N00014-87-K-0482 from Manpower, Personnel, and Training R & D Program to the Georgia Tech Research Corporation. Drs. Farr, Halff, Shafto, and Chipman were cheerleaders and promoters, and performed many other supporting roles that one does not normally associate with faceless government bureaucrats. Dr. Susan Chipman, the most recent Contract Monitor, helped with critical comments and suggestions on methodological issues relevant to cognitive science and training throughout the duration of this project and made efforts to publicize our research. We (especially TG) are grateful for all that they have done. We wish to thank the staff and cadets of the Georgia Tech Naval ROTC unit for their cooperation and help. We especially appreciate the help from Lt. William A. Marriot.

References

1. Anderson, J. R. (1988). The expert module. In M. C. Polson and J. J. Richardson (Eds.), *Foundations of ITS*, Lawrence Erlbaum Associates, Hillsdale, NJ.
2. Anderson, J. R., Boyle, C. F., and Reiser, B. (1985). Intelligent tutoring systems. *Science*, 228(4698), pp. 456-462.
3. Bureau of Naval Personnel (1957). *Engineering Operation and Maintenance*. Prepared by the Bureau of Naval Personnel.
4. Brown, J. S., Burton, R. R., and de Kleer, J. (1982). Pedagogical, natural language and knowledge engineering techniques in Sophie I, II and III. In D. Sleeman and J. S. Brown (Eds.), *Intelligent Tutoring Systems*, Academic Press, London.
5. Burns, H., Parlett, J. W., and Redfield, C. L. (Eds.) (1991). *Intelligent tutoring systems: Evolution in design*. Lawrence Erlbaum Associates, Hillsdale, NJ.
6. Burton, R. R., and Brown, J. S. (1982). An investigation of computer coaching for informal learning activities. In D. Sleeman and J. S. Brown (Eds.), *Intelligent Tutoring Systems*, Academic Press, London.
7. Charniak, E., and McDermott, D. (1985). *Introduction to artificial intelligence*. Addison-Wesley, Reading, MA.

8. Clancey, W. J. (1987). *Knowledge-based tutoring: The GUIDON program*, MIT Press, Cambridge, MA.
9. Fath, J. L., Mitchell, C. M., and Govindaraj, T. (1990). An ICAI architecture for troubleshooting in complex, dynamic systems. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-20 no. 3, pp. 537-558.
10. Frasson, C. and Gauthier, G. (Eds.) (1990). *Intelligent Tutoring Systems: At the crossroad of artificial intelligence and education*. Ablex Publishing Corp., Norwood, NJ.
11. Goldstein, I. L. (1986). *Training in Organizations: Needs Assessment, Development, and Evaluation*. Brooks/Cole Publishing Co., Pacific Grove, CA.
12. Govindaraj, T. (1987). Qualitative approximation methodology for modeling and simulation of large dynamic systems: Applications to a marine power plant. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-17 no. 6, pp. 937-955.
13. Govindaraj, T. (1988). Intelligent computer aids for fault diagnosis training of expert operators of large complex systems. In J. Psotka, L.D. Massey and S.A. Mutter (Eds.), *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Associates, Hillsdale, NJ.
14. Govindaraj, T., and Su, Y. -L. (1988). A model of fault diagnosis performance of expert marine engineers. *International Journal of Man-Machine Studies*, vol. 29, pp. 1-20.
15. Gritzen, E. F. (Ed.) (1980). *Introduction to Naval Engineering*. Naval Institute Press, Annapolis, MD.
16. Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence*, vol. 26 (3), pp. 251-321.
17. Hollan, H. D., Hutchins, E. L., and Weitzman, L. (1984). STEAMER: an interactive inspectable simulation-based training system. *AI Magazine*, 5(2), pp 15-27.
18. Johnson, W. B. (1988). Pragmatic considerations in research, development, and implementation of intelligent tutoring systems. In Polson, M. C. and Richardson, J. J. (Eds.), *Foundations of intelligent tutoring systems*. Lawrence Erlbaum Associates, Hillsdale, NJ.
19. Kearsley, G. Overview. In Kearsley, G. (Ed.) (1987). *Artificial Intelligence and Instructions: Applications and Methods*. Addison- Wesley, Reading, MA.
20. Lajoie, S. P., and Lesgold, A. (1990). Apprenticeship training in the workplace: Computer coached practice environment as a new form of apprenticeship. In *Machine-Mediated Learning*.
21. Lesgold, A. (1990a). Tying development of intelligent tutors to research on theories of learning. In H. Mandl, E. De Corte, S. N. Bennett, and H. F. Friedrich (Eds.), *Learning and Instruction: European research in an international context*. Vol. 3. Pergamon, Oxford.
22. Lesgold, A. (1990b). *Intelligent computer aids for practice of complex troubleshooting*. FAA symposium on Training Technology.
23. Lesgold, A., Lajoie, S. P., Bunzo, M., and Eggan, G. (1991). SHERLOCK: A coached practice environment for an electronics troubleshooting job. In J. Larkin, R. Chabay, and C. Scheftic (Eds.), *Computer assisted instruction and tutoring systems: Establishing communication and collaboration*, Lawrence Erlbaum Associates, Hillsdale, NJ.
24. Macmillan, S. A., Emme, D., and Berkowitz, M. (1988). *Instructional Planners: Lessons*

- Learned. In J. Psotka, L.D. Massey and S.A. Mutter (Eds.), *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Associates, Hillsdale, NJ.
25. Massey, L. D., de Bruin, J., and Roberts, B. (1988). A training system for system maintenance. In J. Psotka, L.D. Massey and S.A. Mutter (Eds.), *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Associates, Hillsdale, NJ.
 26. Miller, J. R. (1988). Human-Computer interaction and intelligent tutoring systems. In *Foundations of ITSs*, Polson and Richardson, Eds. Lawrence Erlbaum Associates, Hillsdale, NJ.
 27. Miller, R. A. (1985). A systems approach to modeling discrete control performance. In W. B. Rouse (Ed.), *Advances in Man-Machine Systems Research* Vol. II. JAI Press Inc., Greenwich, CT.
 28. Moran, T. P. (1983). Getting into a system: external-internal task mapping analysis. *Proceedings of the ACM - CHI Conference on Human Factors in Computing Systems*. Boston, MA, pp 45-49, 1983.
 29. Mitchell, C. M., and Miller, R. A. (1986). A discrete control model of operator function: A methodology for information display design. *IEEE Transactions on System, Man, and Cybernetics*, vol. SMC-16(3), pp. 343-357.
 30. Munro, A., and Towne, D. M. (1993, in press). Chapter in D. M. Towne, T. de Jong, and H. Spada (Eds.) *The Use of Computer Models for Explication, Analysis and Experiential Learning*. NATO ASI Series F, Programme AET, Springer-Verlag, Heidelberg.
 31. Naval Training Command (1973). *Engineering Administration*. United States Government Printing Office, Washington D.C.
 32. Nii, H. P. (1986). Blackboard systems. *AI Magazine*, vol. 7-2 and 7-3.
 33. Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA.
 34. Psotka, J., Massey, L. D., and Mutter, S. A. (Eds.) (1988). *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.
 35. Rasmussen, J. (1985). The role of hierarchical knowledge representation in decision making and system management. *IEEE Transactions on System, Man, and Cybernetics*, vol. SMC-15(2), pp. 234-243.
 36. Rasmussen, J. (1986). *Information processing and human machine interaction: An approach to cognitive engineering*, North-Holland, New York, NY.
 37. Rich, E. (1983). *Artificial Intelligence*. McGraw-Hill, New York.
 38. Rickel, J.W. (1989). Intelligent Computer-Aided Instruction: A survey Organized Around System Components. *IEEE Transactions on Systems, Man, and Cybernetics*, vol 19, No. 1, pp. 40-57.
 39. Rouse, W. B. (1982). Models of human problem solving: Detection, diagnosis and compensation for system failures. *Automatica*, vol. 19, pp. 613-625.
 40. Sheridan, T. B., and Johannsen, G. (Eds.) (1976). *Monitoring behavior and supervisory control*, Plenum, New York, NY.
 41. Sleeman, D., and Brown, J. S., (Eds.) (1982). *Intelligent tutoring systems*, Academic Press, Orlando, FL.

42. Su, Y.-L. (1985). Modeling fault diagnosis performance on a marine power plant simulator. Doctoral dissertation, Center for Human-Machine Systems Research, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.
43. Towne, D. M., and Munro, A. (1988). Intelligent maintenance training system. In J. Psotka, L. D. Massey and S. A. Mutter (Eds.), *Intelligent Tutoring Systems: Lessons Learned*, Lawrence Erlbaum Associates, Hillsdale, NJ.
44. Towne, D. M., and Munro, A. (1990). Model-building tools for simulation based training. In *Interactive learning environments*, 1, pp. 33-50.
45. Towne, D. M. (1986). The generalized maintenance trainer: Evolution and revolution. In W. B. Rouse (Ed.), *Advances in Man-Machine Systems Research* Vol. III, JAI Press Inc. Greenwich, CT.
46. Vasandani, V. (1991). *Intelligent Tutoring for Diagnostic Problem Solving in Complex Dynamic Systems*. Doctoral dissertation, Center for Human-Machine Systems Research, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.
47. Vasandani, V., and Govindaraj, T. (1993, in press). Knowledge structures for a computer-based training aid for troubleshooting a complex system. In D. M. Towne, T. de Jong, and H. Spada (Eds.) *The Use of Computer Models for Explication, Analysis and Experiential Learning*. NATO ASI Series F, Programme AET, Springer-Verlag, Heidelberg.
48. Vasandani, V., and Govindaraj, T. (1993b). Integration of interactive interfaces with intelligent tutoring systems: an implementation. Submitted for publication.
49. Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*, Morgan Kaufmann Publishers, Los Altos, CA.
50. Wickens, C. D. (1984). *Engineering psychology and human performance*, Charles Merrill, Columbus, OH.
51. Winston, P. H. (1980). *Artificial Intelligence* (Second Ed.). Addison-Wesley, Reading, MA.
52. Woods, D. D. (1986). Cognitive technologies: The design of joint human-machine cognitive systems. *AI Magazine*, pp. 86-92.
53. Woolf, B. P. (1986). Teaching a complex industrial process. *Coins Technical Report 86-24*, Computer and Information Science, University of Massachusetts, Amherst, MA.
54. Woolf, B. P., and McDonald, D. D. (1984). Building a computer tutor: Design issues. *IEEE Computer*, 17(9), pp. 61-73.

Appendix: Knowledge Representation and Implementation

In **Turbinia-Vyasa**, components of knowledge are encapsulated using objects. Most of the knowledge is represented in a declarative form and is amenable to changes. Procedures that manipulate data are not stored separately but are mostly encapsulated within the objects. Object-oriented programming features such as inheritance and polymorphism have also been effectively employed. Objects that are instances of the same class have similar representations and share methods that create and manipulate data.

Knowledge concerning the thirteen fluid paths is represented in instances of an object class called **FluidPath**. **FluidPath** has six variables: *fluid-path-name*, *in-schematic*, *in-subsystem*, *components*, *connectors*, and *fluid-path-schematic-association*. *Fluid-path-name* stores the name of the fluid path. Variables *in-schematic* and *in-subsystem* store the names of the schematic and subsystems in which the fluid path is found. A list of component names that lie along the fluid path is stored in the variable *components* and a list of instances of the type connector that represent connections between these components is stored in the variable *connectors*.

An example of a fluid path in the simulated marine power plant is the fuel oil path. The fuel oil path is found in fuel oil and boiler schematics. Components that lie along the fuel oil path are fuel oil settling tank, fuel oil service pump, fuel oil hp regulator, fuel oil heater, fuel oil discharge strainer, fuel oil control valve, master fuel oil valve and the burner. All except the burner in the fuel oil path are displayed on the fuel oil schematic. The object ***FuelOilPath*** is shown in Table A1. Instances of all the thirteen fluid paths are created prior to run time and are initialized from data in input files.

Table A1. Description of ***FuelOilPath***

Object: *FuelOilPath* ; Instance of: FluidPath	
Instance Variables	Value
<i>fluid-path-name</i>	fuel-oil-path
<i>in-schematic</i>	(fuel-oil-schematic boiler-schematic)
<i>in-subsystem</i>	(combustion-subsystem)
<i>components</i>	(fuel-oil-settling-tank fuel-oil-service-pump fuel-oil-hp-regulator fuel-oil-heater fuel-oil-discharge-strainer fuel-oil-control-valve master-fuel-oil-valve burner)
<i>connectors</i>	((#<connector 4003344> #<connector 4003276> #<connector 4003208> -----))
<i>fluid-path-schematic-association</i>	((fuel-oil-schematic #<FluidPathStructureWithinSchematic 4043316>) (boiler-schematic #<FluidPathStructureWithinSchematic 4043116>))

Knowledge concerning the nine subsystems is represented in instances of an object class called **FunctionalSubsystem**. **FunctionalSubsystem** has seven variables: *subsystem-name*, *in-schematic*, *fluid-paths*, *components*, *connectors*, *function* and *subsystem-schematic-association*.

An example of a functional subsystem in the simulated marine power plant is the combustion subsystem. The combustion subsystem spans over the fuel oil and boiler schematics. In the fuel oil schematic, the combustion subsystem has only fuel oil flowing through it. In the boiler schematic, the combustion subsystem has fuel oil as well as combustion air fluid paths in it. The combustion subsystem is comprised of fuel oil settling tank, fuel oil service pump, fuel oil hp regulator, fuel oil heater, fuel oil discharge strainer, fuel oil control valve, and the master fuel oil valve in the fuel oil schematic; and forced draft fan, air heater, windbox, air register and the burner in the boiler sche-

matic. Of the components in boiler schematic, the burner is located in the fuel oil path and the rest lie along the combustion air path. The *CombustionSubsystem* object is shown in Table A2. Instances of all the nine functional subsystems are created prior to run time and are initialized from data in input files.

Table A2. Description of *CombustionSubsystem*

Object: *CombustionSubsystem*; Instance of: FuntionalSubsystem	
Instance Variables	Value
<i>subsystem-name</i> <i>in-schematic</i> <i>fluid-paths</i> <i>components</i> <i>connectors</i> <i>function</i> <i>subsystem-schematic-association</i>	combustion-subsystem (boiler-schematic fuel-oil-schematic) (fuel-oil-path combustion-air-path) (fuel-oil-settling-tank fuel-oil-service-pump fuel-oil-hp-regulator fuel-oil-heater fuel-oil-discharge-strainer fuel-oil-control-valve master-fuel-oil-valve forced-draft-fan air-heater windbox air- register burner) (#<connector 4020096> #<connector 4020028> ----) "To mix the combustion-air with fuel and ignite it in the burner to release thermal energy" ((fuel-oil-schematic #<SubsystemStructureWithinSchematic 4042304>) (boiler-schematic #<SubsystemStructureWithinSchematic 4042504>))
Object: #<4042504>; Instance of: SubsystemStructureWithinSchematic	
<i>fluid-paths</i> <i>components</i> <i>connectors</i> <i>subsystem-fluid-path-association</i>	(fuel-oil-path combustion-air-path) (forced-draft-fan air-heater windbox air-register burner) (#<connector 4020028> #<connector 4020640> ----) ((combustion-air-path #<SubsystemStructureWithinFluidPath 3773792>) (fuel-oil-path #<SubsystemStructureWithinFluidPath 3773992>))
Object: #<3773992>; Instance of: SubsystemStructureWithinFluidPath	
<i>components</i> <i>connectors</i>	(burner) (#<connector 4020708>)

Knowledge concerning the seven schematics is represented in instances of an object class called **Schematic**. Table A3 shows ***BoilerSchematic***, an instance of **Schematic** with most of its instance variables initialized from an input data file. **Schematic** has seventeen class variables which are described in the table.

All components are instances of classes of objects defined by the primitive class hierarchy (see Fig-

Table A3. Description of *BoilerSchematic*

Object: *BoilerSchematic*; Instance of: Schematic	
Instance Variables	Value
<i>ccl-window</i>	#<Object #280, "boiler-schematic", a *color-window*>
<i>title</i>	"boiler-schematic"
<i>schematic-name</i>	boiler-schematic
<i>components-in-schematic</i>	(ac-valve windbox tubes super-steam superheater stack feed-water-regulator furnace forced-draft-fan economizer drum desuper-steam desuperheater burner attemperator atmosphere air-register air-heater ad-drive air-damper)
<i>fluid-paths-in-schematic</i>	(control-air steam desuperheated-steam superheated steam flue-gas feed-water fuel-oil combustion-air)
<i>subsystems-in-schematic</i>	(control-air-subsystem steam-generation-subsystem combustion-subsystem)
<i>list-of-graphic-components</i>	(#<GeneralGraphicObject4002864> #<GeneralGraphicObject 4002656> -----)
<i>list-of-graphic-connectors</i>	(#<connector 4005248> #<connector 4005180> -----)
<i>list-of-graphic-icons</i>	(#<SchematicIconGraphicObject 4006436> #<SchematicIconGraphicObject 4006636> -----)
<i>list-of-visible-graphic-gauges</i>	(#<GaugeIconGraphicObject 4005736> #<GaugeIconGraphicObject 4005936> -----)
<i>list-of-graphic-gauge-readings</i>	(#<GeneralGraphicObject 4005316> #<GeneralGraphicObject 4005516> -----)
<i>picture-handle</i>	#<Mac Handle, Unlocked, Size 2512 #x1DE5B8>
<i>high-light-function</i>	#<Compiled-function active-regions/default-high-light-fn>
<i>low-light-function</i>	#<Compiled-function active-regions/default-low-light-fn>
<i>button-event-function</i>	#<Compiled-function active-regions/button-event-fn>
<i>list-of-active-regions</i>	(#<ActiveRegion 4005392> #<ActiveRegion 4005532> -----)
<i>pick-region</i>	(#<ActiveRegion 4000416>)

ure 4). These components are instantiated prior to run time using a data file that contains structural, functional, behavioral information about components along with knowledge of failures in the component.

At the top of the primitive class hierarchy is an object class called **Primitive**. The nine class variables of **Primitive** are shown in Table A4. Values attached to these variables define the structure, function and behavior of the instantiated object.

The three types of gauges in **TurbInia**: pressure, temperature and level, are instances of **Gauge**. The seven variables of **Gauge** are described in Table A5. Three of these variables store information related to a gauge such as its type, number, and reading. The names of components on either side of the gauge, and the resource ID of the gauge-icon that represents it are also stored in the instance

Table A4. Description of Object Class Primitive

Class: Primitive All components are instances of Primitive or its subclasses	
Class Variables	Description
<i>linked-graphic-objects</i>	list of paired associations. Each pair contains the name of a schematic and an instance of GeneralGraphicObject that represents the primitive in the schematic.
<i>general-structure</i>	contains an instance of PrimitiveGeneralStructure
<i>specific-structure</i>	contains an instance of PrimitiveSpecificStructure
<i>function</i>	describes the function of the primitive
<i>propagation-behavior</i>	pointer to an appropriate propagate-states method
<i>update-states-behavior</i>	pointer to an appropriate update-states method
<i>normal-behavior</i>	instance of NormalBehaviorProperties
<i>abnormal-behavior</i>	list of instances of AbnormalBehavior
<i>failures</i>	pointers to instances of SpecificFailureCase

variables of a gauge.

Table A5. Description of Object Class Gauge

Class: Gauge	
Class Variables	Description
<i>type</i>	pressure, temperature, flow or level
<i>gauge-number</i>	gauge number
<i>on-input-side-of-components</i>	components that have the gauge on their input side
<i>on-output-side-of-components</i>	components that have the gauge on their output side
<i>gauge-reading</i>	gauge reading
<i>gauge-icon-id</i>	resource ID of icon that represents the gauge
<i>linked-graphic-objects</i>	list of paired associations. Each pair contains the name of a schematic and an instance of GaugeGraphicObject that represents the gauge in the schematic.

Representation of failure knowledge in **Vyasa** is discussed next. There are two types of representations used for failure knowledge: one to represent general abnormal system behavior associated with each failure mode and the other to represent specific failures.

The system behavior associated with component failure modes is described in instances of **Failure-Mode** class of objects (Tables A6, A7). There are four failure-mode objects, one for each of the four failure types: *blocked-shut*, *stuck-open*, *leak-in*, and *leak-out*. Knowledge about a failure mode represented in these objects includes information about the upstream and downstream system behavior along the gas and liquid paths, the name of the failure mode and the resource ID of the icon used to represent the failure mode.

Knowledge of the individual faults in the component is stored in objects of class **SpecificFailure-Case**. Variables of **SpecificFailureCase** are described in Table A8.

Table A6. Description of Object Class FailureMode

Class: FailureMode	
Class Variables	Description
<i>name</i>	name of failure mode
<i>icon</i>	resource ID of icon that represents the failure mode
<i>gas</i>	an instance of ExpectedAbnormalBehavior
<i>liquid</i>	an instance of ExpectedAbnormalBehavior

Table A7. Description of Object Class ExpectedAbnormalBehavior

Class: ExpectedAbnormalBehavior	
Class Variables	Description
<i>upstream-behavior</i>	list of affected state upstream and its qualitative value
<i>downstream-behavior</i>	list of affected state downstream and its qualitative value
<i>upstream-behavior-limited-by</i>	primitives upstream that curtail propagation of abnormal behavior
<i>downstream-behavior-limited-by</i>	primitives downstream that curtail propagation of abnormal behavior

Table A8. Description of Object Class SpecificFailureCase

Class: SpecificFailureCase	
Class Variables	Description
<i>failure-number</i>	a number given to failure. Each failure is identified by this number
<i>symptom</i>	symptoms initially observed
<i>cause</i>	name of the failed component
<i>failure-mode</i>	mode of failure
<i>upstream-behavior</i>	upstream affected state and its qualitative value
<i>downstream-behavior</i>	downstream affected state and its qualitative value
<i>upstream-behavior-limited-by</i>	component upstream that curtails propagation of abnormal behavior
<i>downstream-behavior-limited-by</i>	component downstream that curtails propagation of abnormal behavior
<i>upstream-behavior-gauge</i>	upstream gauge number and its qualitative value that shows the abnormal system behavior
<i>downstream-behavior-gauge</i>	downstream gauge number and its qualitative value that shows the abnormal system behavior
<i>affected-subsystems</i>	names of affected subsystems
<i>affected-fluid-paths</i>	names of affected fluid paths
<i>affected-schematics</i>	names of affected schematics
<i>affected-components</i>	names of affected components
<i>affected-gauges</i>	affected gauges along with cause-effect explanations
<i>hypothesis-strengthening-tests</i>	diagnostic tests that strengthen the belief in this failure
<i>hypothesis-weakening-tests</i>	diagnostic tests that strengthen the belief in this failure

Each instance of **SpecificFailureCase** contains information relevant to the failure such as the symptom, the name of the failed component and the mode of failure. The expected upstream and downstream abnormal system behavior due to the fault, the gauges that show this behavior, and the names of the components that curtail the propagation of this behavior away from the malfunctioning component are also stored within the object. In addition, the encapsulated knowledge of specific failures includes information concerning affected schematics, subsystems, fluid paths, and gauges along with explanations of cause-effect associations (see Table A9). The diagnostic tests that strengthen or weaken the likelihood of this specific failure are also stored along with the rest of the information. Details of the gauge readings and information used for end-of-problem help are shown in Table A10.

Table A9. Description of Failure Two

Object: *FailureTwo*; Instance of: SpecificFailureCase	
Instance Variables	Value
<i>failure-number</i>	2
<i>symptom</i>	"When speeding up the ship, boiler level drops low"
<i>cause</i>	"Feed-water regulator is stuck closed"
<i>failure-mode</i>	"blocked-shut"
<i>upstream-behavior</i>	((flow-or-level high))
<i>downstream-behavior</i>	((flow-or-level low))
<i>upstream-behavior-limited-by</i>	(deaerating-feed-tank)
<i>downstream-behavior-limited-by</i>	(deaerating-feed-tank)
<i>upstream-behavior-gauge</i>	(#<flow-or-level-gauge 3955028> high)
<i>downstream-behavior-gauge</i>	(#<flow-or-level-gauge 3939780> low)
<i>affected-subsystems</i>	(feed-water-preheating-subsystem steam-generation-subsystem combustion-subsystem power-generation-subsystem steam-condensation-subsystem)
<i>affected-fluid-paths</i>	(feed-water flue-gas combustion-air superheated-steam desuperheated-steam steam main-condenser-hot-fluid main-condenser-cold-fluid condensate)
<i>affected-schematics</i>	(steam-schematic boiler-schematic feed-water-schematic)
<i>affected-components</i>	(drum tubes economizer superheater)
<i>affected-gauges</i>	affected gauges along with cause-effect explanations is shown on the next page
<i>hypothesis-strengthening-tests</i>	((56 low) (58 high))
<i>hypothesis-weakening-tests</i>	((56 normal) (56 high) (58 normal) (58 low))

Table A10. Details of Failure Two

List of affected gauges along with cause-effect explanations for *FailureTwo*
((58 (slightly-high high) "The speed of the ship is increased by increasing the mass flow rate of steam to the turbines. When the steam demand from the boiler increases, the steam pressure in the drum decreases. This sets the boiler combustion control mechanism into operation. The job of the combustion control mechanism is to increase the quantity of combustion air and fuel to the boiler. Also, when the mass flow rate of steam from the boiler is increased, the boiler feed water control mechanism has to adjust the flow rate of feed water to maintain a mass balance of flow into and out of the boiler. When the feed water regulator is stuck, the feed water control mechanism is unable to increase the feed water flow rate. Such a failure may be regarded as an example of a blocked shut valve. Because the feed-water-regulator does not permit an increase in the feed water flow rate, the upstream water level in the deaerating-feed-tank rises above the normal value.") (56 (low) "The blocked-shut feed-water-regulator causes the water level in the steam drum, downstream, to fall below the normal level.") (7 (slightly-low) "The steam pressure in the boiler drum also decreases when the steam demand is increased. The steam pressure decreases further as the level of water in the drum continues to fall. The combustion control mechanism tries to increase the steam generation rate to build up the steam pressure. Even then the steam saturation pressure in the drum may remain below normal.") (64 (slightly-high) "Since the water level in the drum continues to fall, the saturation pressure of steam in the drum keeps on decreasing. The combustion control mechanism tries to increase the steam generation rate to build up the steam pressure. Therefore, the fuel oil flows into the burner at a rate which is higher than the normal rate for this operating condition.") (1 (slightly-high) "The combustion control mechanism is also responsible for pumping more air into the burner to support combustion of excess fuel.") (66 (slightly-low) "As the fuel flow rate is increased, the flow level in the settling tank falls below normal.") (2 (slightly-high) "The higher than normal combustion air pressure propagates towards the burner.") (3 (slightly-high) "The higher than normal combustion air pressure propagates past the burner.") (46 (high) "Since the available heat energy is now used to heat less feed water, more heat is consumed in superheating steam at constant pressure in the superheater.") (44 (slightly-high) "Since the available heat energy is now used to heat less feed-water in the economizer, the feed-water temperature at drum input is higher than normal") (42 (slightly-high) "With less feed-water to heat in the economizer, the flue gas temperature at the air-heater outlet rises") (48 (slightly-high) "Higher flue-gas temperature causes an increase in combustion-air temperature during preheating in the air-heater") (45 (slightly-high high) "Higher superheated-steam temperature propagates to desuperheater as higher desuperheated-steam temperature") (61 (fluctuating) "Level fluctuates in the deaerating-feed-tank--distillate-tank--atmos-drain-tank feed-back loop to compensate for level variations in the deaerating-feed-tank") (67 (fluctuating) "Level fluctuates in the deaerating-feed-tank--distillate-tank--atmos-drain-tank feed-back loop to compensate for level variations in the deaerating-feed-tank") (55 (slightly-high high) "The higher superheated-steam temperature also causes the steam temperature at lp-turbine exit to be higher than normal"))