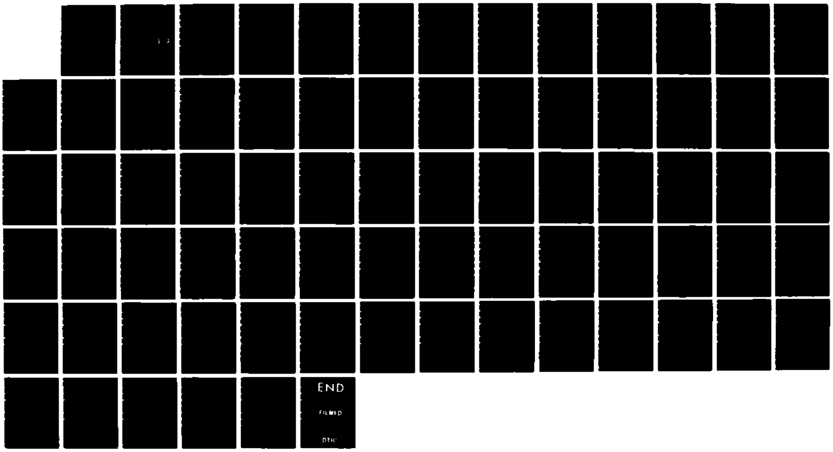
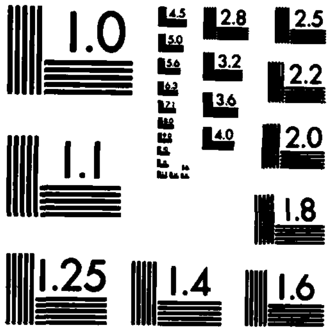
AD-A158 035 A REACTIVE LEARNING ENVIRONMENT FOR THE EQUIPMENT PROBLEM SOLVING TECHNIQUES (EPST) SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH D G MCKENNEY 1/1

UNCLASSIFIED AUG 85 AFIT/CI/NR-85-68T F/G 5/9 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

UNCLASS

SECURITY CLASSIFICATION OF

AD-A158 035

(1)

REPORT I	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR 85-68T	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Reactive Learning Environment for the Equipment Problem Solving Techniques (EPST) System	5. TYPE OF REPORT & PERIOD COVERED THESIS/DISSERTATION
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) David Gene McKenney	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: The University of Utah	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433	12. REPORT DATE August 1985
	13. NUMBER OF PAGES 70
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASS
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)	
18. SUPPLEMENTARY NOTES APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1	[illegible] LYNN E. WOLAVER Dean for Research and Professional Development AFIT, Wright-Patterson AFB OH 5 AUG 1985
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) ATTACHED	

85 8 13 104

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

This thesis describes a user-friendly program that allows students to troubleshoot equipment in a reactive learning environment. The system allows students to develop problem solving strategies while troubleshooting faulty equipment through the use of video scenes with graphic overlays. The student interface contains instructional strategies which access information stored in a database consisting of semantic networks and frames. Student troubleshooting involves moving within equipment, changing device settings, setting up test equipment, and obtaining equipment readings.

Important features of the student interface include the simulation of devices based on first order effects, the use of production rules to describe device state transitions, and the use of an emulation algorithm to ripple out device state changes. The student interface also allows users to follow wire connections, set up test equipment, record hypotheses, and get on-line help. In addition, the groundwork is provided for the future addition of an Advisor module (or Coach) to monitor the student's progress throughout the troubleshooting process and give advice.

The student interface is general in nature. The content of the database can be changed to represent any type of equipment, but the student interface does not have to be redeveloped or modified for the new database. The student interface is written in "C" on a Unix operating system, which permits the program to run on both large and small computer systems.

68T

AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to ascertain the value and/or contribution of research accomplished by students or faculty of the Air Force Institute of Technology (AU). It would be greatly appreciated if you would complete the following questionnaire and return it to:

AFIT/NR
Wright-Patterson AFB OH 45433

RESEARCH TITLE: A Reactive Learning Environment for the Equipment Problem Solving Techniques (EPST) System

AUTHOR: David Gene McKenney

RESEARCH ASSESSMENT QUESTIONS:

1. Did this research contribute to a current Air Force project?

( ) a. YES ( ) b. NO

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not?

( ) a. YES ( ) b. NO

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency achieved/received by virtue of AFIT performing the research. Can you estimate what this research would have cost if it had been accomplished under contract or if it had been done in-house in terms of manpower and/or dollars?

( ) a. MAN-YEARS ________ ( ) b. $________

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3. above), what is your estimate of its significance?

( ) a. HIGHLY SIGNIFICANT ( ) b. SIGNIFICANT ( ) c. SLIGHTLY SIGNIFICANT ( ) d. OF NO SIGNIFICANCE

5. AFIT welcomes any further comments you may have on the above questions, or any additional details concerning the current application, future potential, or other value of this research. Please use the bottom part of this questionnaire for your statement(s).

NAME GRADE POSITION

ORGANIZATION LOCATION

STATEMENT(s):

68

A REACTIVE LEARNING ENVIRONMENT FOR THE EQUIPMENT PROBLEM SOLVING TECHNIQUES (EPST) SYSTEM

by

David Gene McKenney

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
The University of Utah
August 1985

To my wife, Annamarie

ABSTRACT

This thesis describes a user-friendly program that allows students to troubleshoot equipment in a reactive learning environment. The system allows students to develop problem solving strategies while troubleshooting faulty equipment through the use of video scenes with graphic overlays. The student interface contains instructional strategies which access information stored in a database consisting of semantic networks and frames. Student troubleshooting involves moving within equipment, changing device settings, setting up test equipment, and obtaining equipment readings.

Important features of the student interface include the simulation of devices based on first order effects, the use of production rules to describe device state transitions, and the use of an emulation algorithm to ripple out device state changes. The student interface also allows users to follow wire connections, set up test equipment, record hypotheses, and get on-line help. In addition, the groundwork is provided for the future addition of an Advisor module (or Coach) to monitor the student's progress throughout the troubleshooting process and give advice.

The student interface is general in nature. The content of the database can be changed to represent any type of equipment, but the student interface does not have to be redeveloped or modified for the new database. The student interface is written in "C" on a Unix operating system, which permits the program to run on both large and small computer systems.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank those organizations and people without whose help and guidance this research could not have been completed. My deepest appreciation goes to the U.S. Air Force, which provided both the opportunity and financial support that allowed me to pursue an advanced degree.

I would like to thank Dr. Richard Brandt and Barbara Knapp for their help in developing and expanding some of the ideas contained in this thesis. Thanks also to my advisor, Dr. Richard Brandt, and the other committee members, Drs. Thomas Henderson and Robert Kessler, for their support and advice during preparation of this manuscript. I would like to especially thank Sandra Peck for her careful reading of the manuscript and her helpful comments on style and content.

I would also like to thank everyone in the Computer-Based Education Software System (CBESS) group, especially those individuals who developed some of the underlying CBESS components the student interface uses. Thanks to Eric C. ("No I *don't* do windows") Brown for developing the Interaction Driver software and to Brad ("They will be up Real Soon Now") Davis for his work in developing sequences and windows. A special thanks to Rick Paulsen, who developed the EPST database software, without which the student interface could not run.

This work was partially supported by the Navy Personnel Research Development Center under Contract Number N00224-83-C-1759.

CHAPTER 1

INTRODUCTION

Equipment Problem Solving Techniques (EPST) is a Computer-Aided Instruction (CAI) system developed at the University of Utah as a component of the Computer-Based Education Software System (CBESS) [3, 12]. EPST is an interactive, image-based, computer-controlled trainer simulator capable of simulating a wide variety of electronic and electromechanical equipment. The major objective of EPST is to give students experience in troubleshooting electronic equipment and developing troubleshooting strategies. The EPST student interface developed in this thesis provides the basis for a reactive learning environment that *goes beyond* the traditional "branch and test" environment normally found on microcomputers. The student interface combines recent advances in cognitive science and Artificial Intelligence to allow students to not only learn about how to troubleshoot faulty equipment, but also to actually develop problem solving strategies while troubleshooting.

This thesis addresses several design features of the student interaction program (student interface), particularly those that provide aid to students in forming hypotheses, developing troubleshooting strategies, and obtaining information on how actual electronic equipment works. Device representation and emulation are also discussed.

1.1 Motivation

Much of the motivation for designing EPST stems from the search for a low-cost, image-based classroom trainer/simulator that can simulate the operational and maintenance functions of a wide variety of electronic equipment. Such a trainer is needed because extensive training and practice is required for electronics technicians to acquire operational and maintenance skills. The equipment normally used for this training and practice is actual operational equipment, which is expensive, involves hazards to both people and machines, and often involves time-consuming procedures.* Another problem with using actual equipment is that equipment is often in short supply, resulting in delays or training on inappropriate or obsolete equipment [13]. Previously developed trainer/simulators have only provided a practice environment, relying on instructors to provide instruction and on-line help.

EPST, on the other hand, is a trainer/simulator that permits the emulation of the characteristics of a wide range of equipment and systems. EPST presents realistic performance symptoms similar to those of real equipment during either normal or failed operating conditions and provides, when needed, instruction and on-line help that would normally come from an instructor. The main purpose of the EPST student interface design is to create not just a learning-by-doing practice environment, but a reactive learning environment that helps students learn appropriate problem solving strategies. The EPST student interface allows students to examine their own thinking and learning strategies and assists students in forming and testing their strategies. It also helps students learn more about the specific equipment they are troubleshooting and its related test equipment by providing on-line specific help through access to knowledge stored in the EPST database. Thus, the EPST student interface improves students' understanding of the operation of electronic equipment, their ability to recognize equipment problems, and their

*Fault insertion and conversion maintenance is time-consuming.

ability to troubleshoot these problems.

1.2 Outline of Thesis

The rest of this chapter describes the overall EPST system and related work. The chapters that follow describe the student interface designed for EPST and the reactive learning environment it creates. Chapter 2 describes the make-up of the EPST database, which has been designed using a combination of semantic networks and frames that provide a powerful framework for storing and accessing knowledge. This database design allows for the easy, compact storage of knowledge about different pieces of faulty/test equipment yet permits quick access to this knowledge by the program or student while troubleshooting. This type of storage and access was not available in earlier trainers. The database design also makes it possible for EPST to provide a more realistic troubleshooting environment by allowing students to actually follow wire connections and access test equipment during troubleshooting. The database creates a solid framework on which to add future enhancements such as an advisor or coach.

Chapter 3 describes the student interface in detail, defining the overall structure of the reactive learning environment and describing student movement during troubleshooting. This environment allows students to select and view different sections of the faulty equipment and to create and test hypotheses. Students not only view sections of the faulty equipment and change its control settings, but also view and set up test equipment, set up test connections, and actually take test equipment readings. Allowing students to use test equipment in this way provides more realistic troubleshooting that results in more training in the proper use of test equipment. Chapter 3 also describes a method that allows students to follow physical (wire) connections in order to trace erroneous signals from device to device. This method uses the input and output triangle nodes stored in the semantic networks in the EPST database. Finally, after a student has finished a problem, the student has the opportunity to view and compare his procedures with those of an expert.

Chapter 4 discusses how the EPST student interface aids students in forming and testing hypotheses. It describes how students can think out and record their hypotheses and proposed tests, and how this information will be made available during troubleshooting and for instructor evaluation. EPST can also suggest possible hypotheses at certain points in the troubleshooting session and provide various types of on-line help to aid students in formulating their own hypotheses. The types of on-line help available to the student are described.

Chapter 5 discusses device representation and emulation, and how production rules are used. The emulation algorithm is given along with an example of how it works. The final chapter, Chapter 6, summarizes the results of this thesis and outlines areas for further research.

1.3 The EPST System

EPST is a versatile educational tool that consists of relatively machine independent software which allows Subject-Matter Experts (SMEs) who know nothing about computer programming to create instructional material by entering informational content into an appropriate EPST database. EPST uses a combination of semantic networks, production rules, and frames to represent knowledge in the database. Emulation in EPST is based on first order knowledge, where each device is defined in terms of inputs and outputs, and production rules are used to determine the outputs of a device based on its inputs. A representation of how individual devices are connected to each other is kept in a semantic network.

The major objective of EPST is to give students experience in troubleshooting electronic equipment through the use of video scenes with graphic overlays representing the status of actual equipment. During authoring and on computers that cannot support video, menus are used to display information normally found on video pictures. The current EPST prototype uses only menus, since the video interface and graphics package is still being developed.

For ease of discussion, EPST is normally divided into two parts: EPST Database and User Interaction. The EPST Database is described in detail in Chapter 2. The EPST User Interaction consists of two parts: authoring and student interaction. The EPST authoring mode enables SMEs who are not computer programmers to develop, enter, and modify data in the EPST database. The EPST student interaction mode allows students to obtain practice in problem solving techniques by interacting with the EPST database that was created in the authoring mode.

1.3.1 Authoring Mode

The authoring mode consists of two steps: EPST database creation and creation of video images with graphic overlays.

1.3.1.1 EPST database creation. Creating an EPST database requires an SME to:

1. specify the equipment to be used for troubleshooting practice,
2. specify the test equipment that may be used in troubleshooting,
3. specify the problems that may be presented for troubleshooting practice,
4. enter the information into the database by defining the semantic networks, and
5. test the database to ensure content accuracy.

Entry and modification of the information needed in these five areas is performed in the authoring mode which is described in Paulsen et al. [19].

1.3.1.2 Creating video images and graphic overlays. In this step, an SME will specify the video images needed for displaying the different parts of the equipment. Creating video images is described by Brandt [4].

1.3.2 Student Interaction

Student interaction allows students to develop and use problem solving strategies while troubleshooting faulty equipment. Students continuously develop, test, and modify hypotheses during troubleshooting. To test hypotheses, students can:

1. change control settings on the simulated equipment,
2. set up test equipment,
3. take test equipment readings,
4. observe responses obtained from the changed control settings and test equipment, and
5. ultimately identify and replace the element that is causing abnormal symptoms.

On-line help is available to aid students in forming hypotheses. At the end of a troubleshooting session, students can compare their solution paths to those of an expert. The student interface designed for student interaction is described more fully in Chapter 3 and in Paulsen et al. [19].

1.4 Related Work

EPST is based on the frame-based Electronic Equipment Maintenance Trainer (EEMT) [7] designed for the Navy by the Cubic Corporation Defense Systems Division. The EEMT system is a two-dimensional trainer/simulator designed to reduce reliance on the use of actual equipment trainers in Navy technical schools. It is an outgrowth of the Generalized Maintenance Trainer/Simulator (Rigney Trainer) developed by Dr. Joseph Rigney and others at the Behavioral Technology Laboratory, University of Southern California [12, 14]. GMTS showed the feasibility, effectiveness, and broad application of a trainer/simulator in the field of electronic training systems. Except for some differences in hardware selected for implementation, EEMT functions identically to GMTS. For this reason, normally both systems are jointly referred to as EEMT.

EEMT allows students to practice troubleshooting skills on equipment through learning-by-doing, but it does so in a limited way. EEMT:

1. does not allow students to learn about their own thinking and learning strategies;
2. does not provide help to students in forming or testing their hypotheses;
3. does not provide specific help to students on the equipment they are troubleshooting;
4. allows students to thrash about and spend many hours pursuing wrong paths.

One reason for these limitations is the way the subject matter knowledge is represented and organized in EEMT. All the system knowledge is implicitly programmed into the database with no means of accessing it, so there is no way for a student to view this knowledge and learn from it, nor for the software to use the knowledge to create a reactive learning environment for the student.

Several projects have been associated with trying to solve these limitations in electronic troubleshooting programs, including one of the more successful ones, SOPHIE of Brown et al. [5]. The SOPHIE program teaches electronic troubleshooting for a particular electronic device, the IP-28 regulated power supply. SOPHIE creates a reactive learning environment that evaluates student's hypotheses, critiques measurements, handles any question presented in the context of electronic troubleshooting, and uses a simple coach to track and advise the student during troubleshooting. While SOPHIE was developed for use in troubleshooting simple circuits, EPST deals with a more complex troubleshooting environment. A less robust program, TASK, developed by Search Technology Inc., also teaches the fundamentals of troubleshooting simple circuits, but does not evaluate students' hypotheses, nor provide a coach.

Concurrent research in simulating circuits for electronic troubleshooting systems is being done by Randall Davis at MIT [8]. Davis also uses reasoning based on first principles (first order effect). Davis' reasoning uses knowledge of structure and behavior, where structure is the information about the interconnection of modules, and behavior refers to the black box description of a component.[*] Reasoning from first principles offers many advantages, including making it easier to construct and maintain the overall system.

A thesis by David Matty describes a similar use of first order knowledge to describe module behavior in a Constraint Driven Synthesis system [17], where module behavior is defined in terms of inputs and outputs and outputs are derived from inputs using procedural definitions. David Matty's current research also deals with using first order knowledge to simulate behavior.

The EPST algorithm used for emulation and the theory of how production rules are evaluated to determine outputs from inputs was originally developed by EPST group members and other members of the CBESS group. Lee Coller, a member of the EPST group, has implemented a prototype of a production rule evaluator, which searches a list of production rules for an applicable rule. A similar system is also described in a thesis by Michael Lemon [16].

There are numerous publications that deal with issues in human-computer interface design. One of the more complete articles is by Beverly and Robert Williges [22], which compiles in one document various dialogue design considerations from a variety of sources. Many of the ideas presented in the Williges' paper and in other articles [10, 11, 15, 18] were used in designing the EPST student interface.

[*]Black box description means: How is the information leaving the component related to the information that entered it?

CHAPTER 2

EPST PROJECT DESCRIPTION

2.1 Database Structure

EPST uses a combination of semantic networks and frames to represent the knowledge content in the EPST database. Combining these two representations allows the knowledge to be stored efficiently, permitting fast access and easy storage. Elaine Rich [20] defines frames and semantic nets as general-purpose structures in which particular sets of domain-specific knowledge can be embedded. Using semantic networks in the EPST database provides a strong framework that exploits ISA and IS_PART_OF* relationships to represent knowledge about equipment and its parts. These semantic networks allow the representation of different pieces of equipment in terms of spatial, physical, and logical dependencies,** without having to store explicitly all of the implied relations. This advantage, along with being able to use property inheritance, allows decreased storage space and the use of set search strategies for finding needed information. Semantic networks also provide EPST the flexibility to allow students to access and set up test equipment, and to trace wire connections on faulty equipment.

In Artificial Intelligence, *frames* are used to describe a collection of attributes that a given object possesses. The EPST database design uses frames

*The ISA and IS_PART_OF relations are discussed in Section 2.1.1.

**Physical refers to actual wire connections between two pieces of equipment, while spatial refers to pieces that are located next to each other on the equipment but may not be physically connected. Logical refers to pieces of equipment that are related in the context of troubleshooting.

to store attributes of different parts of the equipment, which reduces the number of triangles and search time. If a pure semantic network were used without the addition of frames, each piece of knowledge would be represented as a triangle relation, consisting of a subject, relation, and object. For example, a production rule would be stored as **ProductionRule1** *IS_PRODUCTION_RULE_OF* **Device3**. If a large piece of equipment with many devices were stored in such a pure semantic network, a great number of triangles would be needed.* By using frames instead of relations to store equipment attributes, the EPST database design greatly reduces the number of triangle relations needed, thus eliminating much of the overhead cost associated with storing and searching these triangles during run time. Frames also allow knowledge to be localized by storing information with only that part of the equipment that needs to know the information.

If only small pieces of equipment with few devices were used in EPST, a pure semantic network would work well. But EPST is designed to handle equipment with a relatively large number of devices. Frames also provide a way to store instructional information that can be used if an advisor or coach is added.

The rest of this chapter discusses how semantic networks are used as the basic structure for the EPST database, how frames are used to store additional information, and how the design of the IS_PART_OF tree affects the learning environment.

2.1.1 Semantic Network

EPST uses semantic networks to represent the physical layout and relationships between devices on different pieces of equipment. Two trees are used in constructing semantic networks for the EPST database: an **ISA** tree and

*EPST uses 16 bit indices for all nodes, and is currently limited to about 32,000 nodes. Nodes are described in Section 2.1.1.

an **IS_PART_OF** tree. An ISA tree relates parts on a conceptual basis and is referred to in EPST as a parts tree. An IS_PART_OF tree relates parts in terms of their spatial, physical, and logical placement on the equipment and is referred to in EPST as an equipment tree. During troubleshooting, all steps a student takes to access different parts of the equipment are on the equipment IS_PART_OF tree. Therefore, the design of each equipment IS_PART_OF tree strongly affects the learning environment and the student's troubleshooting strategies.

Triangles are used in semantic networks to represent knowledge and are composed of three parts: a subject, a relation, and an object. EPST uses triangle nodes to represent these triangles in the database, each triangle node being unique. The subject and object can be any node in the database, while the relation must be a permitted relation. EPST permits three types of triangle nodes, each defined by the types of relations allowed. The three types of triangles nodes are shown in Figure 1. Three different types of triangle nodes are used in EPST to decrease the *search time required during run-time*. For example, if information is needed about an input, then only the input triangle nodes need be searched instead of all triangle nodes. If a triangle node is being searched for that has a standard relation, such as the IS_PART_OF relation, then only the standard triangle nodes need to be searched. By searching only a subset instead of all the triangle nodes for the current equipment tree, search time required is reduced, thus improving response time.

EPST uses first order knowledge, whereby each device is defined in terms of the production rules that describe how outputs are obtained from inputs. This information is stored with the generic device in the parts tree. The equipment tree contains the spatial, physical, and logical relationships between devices. That part of the equipment tree consisting of the IS_PART_OF relations contains the logical and spatial relationships, while the remaining relations in an equipment tree semantic network show physical (wire) connections. All this information is readily available to authors for error checking, to the EPST

1. **INPUT RELATION** - An input relation triangle node represents a triangle where the relation is an input relation. An example of an input relation would be, **NAND2** *input_A* **+5 volts**, where NAND2 is the subject, input_A is the relation, and +5 volts is the object.* The number of input relations allowed is large, since the relation itself will be the actual name of the input. Different devices may have the same input relation.

2. **OUTPUT RELATION** - An output relation triangle node is similar to an input relation triangle node except it represents a triangle where the relation is an output relation. An example of an output relation would be, **NAND2** *output_Q* **+5 volts.**

3. **STANDARD** - A standard triangle is a triangle that is not an input or output triangle. An example of a standard relation would be, **2_INPUT_NAND** *isa* **NANDGATE**. The types of standard relations used are:

ISA
IS_PART_OF
INSTANCE-OF
IS-TESTPOINT-OF
IS-CONNECTED-TO

Figure 1: Triangle Nodes Used in EPST

student interface program for use in simulating devices, and to students through on-line help.

<u>**2.1.1.1 Parts database.**</u> A parts tree is a semantic network that defines devices and probes and categorizes them using the ISA relation. The ISA relationship is used in this case to represent the relationship between objects in a hierarchical taxonomy. All devices in the equipment tree are instances of an item in the parts tree. Each parts tree is composed of the ISA nodes and two types of nodes: part nodes and category nodes. Part nodes describe the operation of devices and are represented as leaf nodes on the parts tree. Category nodes are used to classify part nodes and are represented as nonterminal nodes on the parts tree. While several levels of category nodes

*The object in this case is not another device but a quantifier.

may exist, the first level will always categorize all devices under either Mechanical, Electrical, Electro-mechanical, Probe, or as uncategorized. A sample parts tree is shown in Figure 2.

Subject-Matter Experts define the operation of equipment devices by entering leaf part node data, and then classifying the nodes under a category node. Since each equipment device is an instance of a part node, all static information and knowledge about each equipment device is localized in the respective part node in the parts tree.[*] Any information that changes, such as current state information, is stored with the individual nodes (instances of the part nodes) in the equipment tree. The EPST database design simplifies authoring [19], since an author need enter only once the information known about each part device and how it may be connected to other devices. Authors use instances of these part nodes to build equipment trees.

__2.1.1.2 Equipment databases.__ An equipment tree is a semantic network that contains a spatial, physical, and logical representation of the equipment layout. All nodes on an equipment tree correspond to actual equipment parts or devices and are related by their spatial, physical, or logical placement on the equipment. Spatial placement refers to pieces that are located next to each other on the equipment but are not physically connected. For example, a device may be represented as a sibling of another device simply because it is physically located on the same panel with the device, even though it may not be wired to anything on the panel. Physical connections refer to actual "wire" connections between devices. Physical links exist where wires normally would be. These links are represented by input and output triangle nodes and by standard triangle nodes that use the IS-TESTPOINT-OF and IS-CONNECTED-TO relations. Examples of standard triangle node "wire" links are:

[*]The current version of the EPST database does not use inheritance in the parts tree to store information. Using inheritance in the parts tree is discussed as a future research topic in Chapter 6.

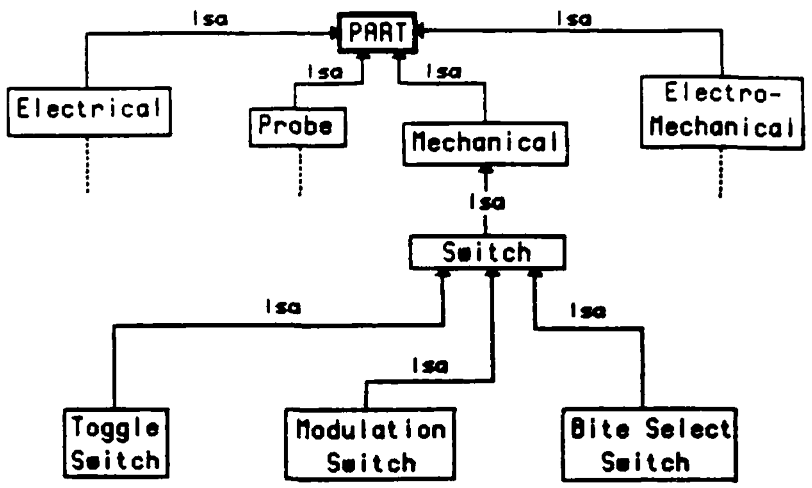


Figure 2: Sample Parts Database

TP1 *is_testpoint_of* **PANEL2**

TP1 *is_connected_to* **QNT_1.**[**]

Logical placement refers to the way the equipment is subdivided logically into panels and devices by the SME when constructing the equipment tree. For example, devices and panels may be *grouped together on a panel* because they are task related. This logical structure of the equipment tree is very important, since it may influence the student's cognitive model of the equipment structure. This idea is discussed in Section 2.2. A sample equipment tree with a portion of its parts tree is shown in Figure 3.

The general structure of an equipment tree uses the IS_PART_OF relation and is composed of three types of nodes: a scene node, panel nodes, and device nodes. The scene node is a sketch, diagram, or picture that represents an overall view of the equipment being simulated, including all external panels. Each equipment tree contains only one scene node. Panel nodes represent equipment panels or sections of the equipment that provide access to devices, testpoints, or other panels. Device nodes represent equipment devices

[**]Quantifier nodes, such as QNT_1, are used to represent values in the database. In this example, Qnt_1 is the name of the quantifier that contains the value of TP1.

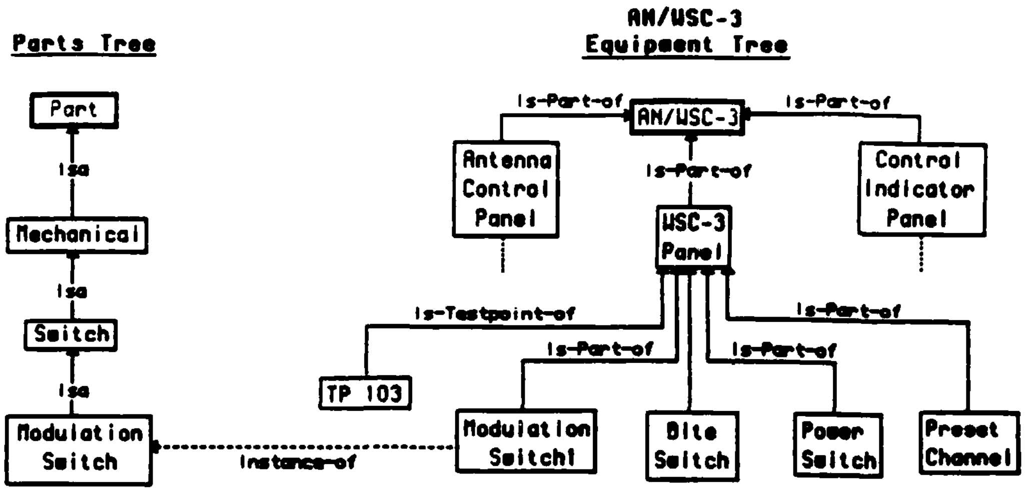


Figure 3: Sample Equipment Database

depicting switch settings, indicator readings, or any other observable state information and provide access to testpoints. Every device node on an equipment tree is connected to its respective part node (leaf) on the parts tree by the INSTANCE_OF relation. A device node is the only type of node that contains state information. A probe is a special case of a device. It is at the device node that the student normally interacts with the trainer/simulator by changing device settings, setting up testpoint connections, and obtaining testpoint readings. Testpoint readings can also be obtained from testpoints on a panel node.

When an equipment tree is created, nodes are arranged in an hierarchical structure, with the scene node shown as the root node, panel nodes as nonterminal nodes, and device nodes as leaf nodes.

A fourth kind of node contained in equipment trees are testpoint nodes. They are connected to panel and device nodes by the IS_TESTPOINT_OF relation. Testpoint nodes are used to access quantifiers and are also shown as leaves on the equipment tree.

2.1.2 Frames

Each part node and each instance of a part node has a frame associated with it. This frame is referenced by an index number stored in the node. Within each frame, slots are used to store such things as production rules, states, and data for graphics and video. Additional help information not usually associated with equipment, such as possible hypotheses and tests to be performed, is also stored in frame slots.

All information that can be inherited by all instances of a device is stored in the frame attached to the device part node in the parts tree. Other frame information that pertains to an actual instance of the device is stored in the frame attached to the respective instantiated device node in the equipment tree.

The EPST SUIDD document [19] describes the structure of frames for the different types of nodes and the record format for the different slots. The types of slots used in EPST frames are described in Table 1 and a portion of a frame attached to a device part node is shown in Figure 4.

2.2 How the Equipment Tree Affects Learning

The IS_PART_OF tree has a direct influence on the student's cognitive model of the equipment. How a Subject-Matter Expert (SME) logically divides the equipment up into panels and devices when building an equipment tree has a direct influence on the strategy the student uses and how the student remembers relationships between panels and devices on the equipment.

While practicing troubleshooting, the student will develop a cognitive model of the layout of the equipment based on how the logical organization (using the IS_PART_OF relation) was defined in the equipment tree. For example, if the initial scene state is divided up into three panels, the student will form a cognitive model in which there are three main panels or parts on the equipment. If it is usually necessary to inspect one of these three panels first for certain problems, then by defining the first level of the equipment tree as consisting of only three panels, the student is forced to select one of these

Table 1: Slots Used in EPST Frames

Slot	Description
Errors	The set of error records associated with the part node.
Graphics Information	The graphic record associated with the part, scene, panel, device, or probe node.
Hypothesis	The set of possible student hypotheses associated with panel and device nodes.
Node	The index of the corresponding atom node in the semantic network.
Positions	The set user of input records associated with the device node.
Production Rules	The set of production rule records associated with the part node.
Replacement Information	The replacement information record associated with the panel or device.
Selection Area	The set of selection area records associated with the panel, device, probe, or testpoint node.
States	The set of state records associated with the part node.
Tests	The set of possible tests associated with panel or device nodes.
Video Image	The video image associated with the scene, panel, or probe node.

panels depending on the problem. An SME may also group certain panels or devices together because they are task related. Such grouping helps students remember relationships among different parts of the equipment.

On past trainers, students were constrained during troubleshooting to follow only what EPST defines as the IS_PART_OF relation. While this helps develop a cognitive model of equipment layout and relationships between panels and devices, it does not present a realistic troubleshooting environment. For example, when troubleshooting an actual piece of equipment, a student may find a faulty input to a device, and want to trace the input (wire) to its source. In previous systems (using only the IS_PART_OF links), the student would have to determine which device the input wire came from, move up the IS_PART_OF

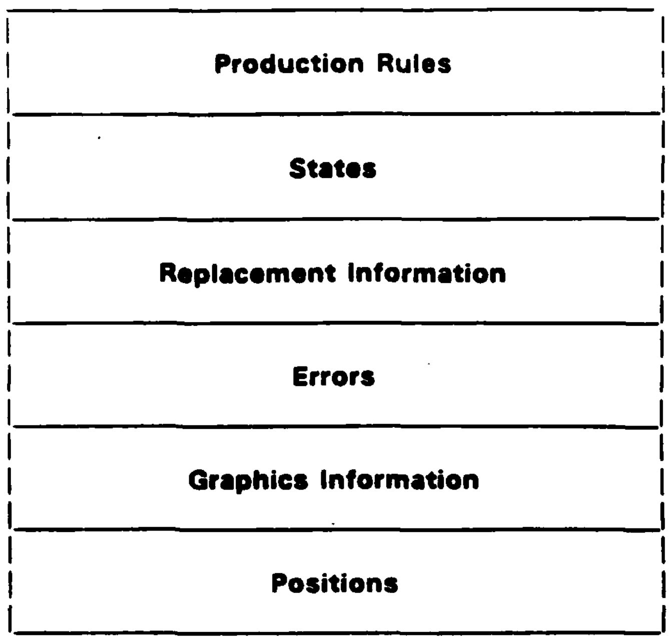


Figure 4: Device Part Node Frame

tree until reaching a common ancestor of both the current device and the desired device, and then move back down the tree to the desired device.

Since EPST explicitly stores wire links as input and output triangle nodes, the student can actually follow "wires" during troubleshooting and trace down a bad input source. This is a significant change from past trainer/simulators, and is due to the fact that EPST can represent wire connections as well as the logical layout of panels and devices. An example of how a student can follow wire connections is presented in student movement in Chapter 3.

CHAPTER 3

STUDENT INTERFACE

The EPST student interface is the basis for a reactive learning environment, one that allows students to develop problem solving strategies while practicing the troubleshooting of faulty equipment. John Seeley Brown [6] describes a reactive learning environment as one that allows students to try out their hypotheses, see the results of their tests, analyze their data, find counterexamples to their hypotheses, and experiment with different solutions. For the reactive learning environment to be productive, the troubleshooting environment created should be easy to learn and use, so that it does not detract from the more important issue of learning. The EPST student interface also lets students explore their partial understanding of how a system works with complete safety. Students are able to formulate, test, and witness the consequences of their ideas without worrying about possible catastrophic consequences.

Troubleshooting faulty equipment involves developing a strategy, and then using this strategy to access different parts of the equipment, change equipment switch settings, obtain testpoint readings, and replace bad panels or devices. During basic problem solving, human cognition tends to invoke a sequence of actions based on various patterns of knowledge. These steps in problem solving are based on cognitive rules that specify which actions should be performed under a given set of conditions. These steps are called *productions*, which usually consist of an overall goal and a set of subgoals used to reach the overall goal. The ACT theory of cognition [1] uses goal-directed productions, where the conditions needed to reach the overall goal are goals themselves. This type of goal directed cognition has been the key to effective teaching and tutoring of equipment troubleshooting and problem solving in general [2].

Based on this goal-directed approach to problem-solving, the general troubleshooting process the student should follow during EPST student interaction consists of the following four steps:

Step 1. Student develops and enters an hypothesis, identifying information used in forming the hypothesis.

Step 2. Student proposes a test or tests (goals and subgoals), based on the hypothesis, that should prove or disprove the hypothesis.

Step 3. Student performs the tests and gathers information both explicitly* and implicitly.**

Step 4. Student analyzes information obtained from the tests and uses it to form conclusions about the hypothesis. The student returns to step 1 and repeats the process until a solution is reached.

The different parts of the EPST student interface will be described in terms of:

1. entry of hypotheses,
2. student interaction modes,
3. student movement,
4. measurements,
5. scoring,
6. on-line help, and
7. solving the problem.

Entering hypotheses and the use of on-line help are discussed in detail in Chapter 4. The other five areas are discussed in the remainder of this chapter.

---

*Information received through readings from test points.

**Information received through observations of meters, lights, switches on the equipment, and the effect of replacement parts.

3.1 Student Interaction Modes

EPST student interaction allows six different student interaction modes.[*] The teaching, practice, and freeplay modes are the modes that are used in the reactive learning environment. The overall troubleshooting sequence the student follows in all three modes will be the same. The teaching and practice modes provide a structured learning environment while the freeplay mode provides an unstructured environment. The main differences among the three modes are the options and specific help[**] available to the user while troubleshooting the equipment. The reasons for having a structured and unstructured learning environment are:

1. The structured environment allows an instructor to adapt the system to individual teaching style and course flow.
2. The structured environment sets the stage for the later addition of an interactive advisor/coach module.
3. The unstructured environment allows students to use the system with no constraints, experimenting freely.

The three modes are defined in the following sections.

3.1.1 Teaching Mode

The teaching mode allows <u>structured</u> practice in troubleshooting malfunctioning equipment. This mode allows the instructor to preselect:

1. the faulty equipment database to be used by the student,
2. the troubleshooting problem to be solved by the student,
3. the troubleshooting problems available for student selection,[***]
4. the order in which the troubleshooting problems will be done,

[*] The six modes are: tutorial, teaching, practice, freeplay, tryout (for authors), and test.

[**] The types of specific help available are described in Chapter 4. Examples of specific help are how far away the student is from the bad device, how a device works (production rules), and the input/output values for a device.

[***] This restricts students to selecting only from a preselected list specified by an instructor.

5. the options available to the student in the Finished state if the malfunction was not corrected, and

6. the types of specific help available to the student.

In the teaching mode, an instructor can create a controlled learning environment, by preselecting certain constraints that control which problems a student can do and what is available to the student in the problem.

3.1.2 Practice Mode

The practice mode is the same as teaching mode except the only specific help available is the production rules associated with how a device works. This allows the student to troubleshoot with virtually no help available, simulating a test environment.

3.1.3 Freeplay Mode

The freeplay mode allows unstructured practice in examining or troubleshooting malfunctioning equipment. The student can examine and traverse an EPST database with no restrictions. All defined help will be available.

3.2 Student Movement in EPST

In EPST, students examine or troubleshoot faulty equipment by moving among ten different EPST system states. Movement is controlled by selecting an item from a menu, by selecting a command from the command line, or by selecting the next panel or device node to view. A state diagram of student movement in the EPST system is shown in Figure 5.

While traversing the different states shown in Figure 5, users can interrupt the EPST system at any time by pressing the ESCAPE <ESC> or HELP <?> key. The escape interrupt allows students to quit the troubleshooting session at any time.* The help interrupt allows the student to access any on-line help that

*The escape interrupt in future releases of EPST also will allow students to send comments and notes to Subject-Matter experts and instructors.

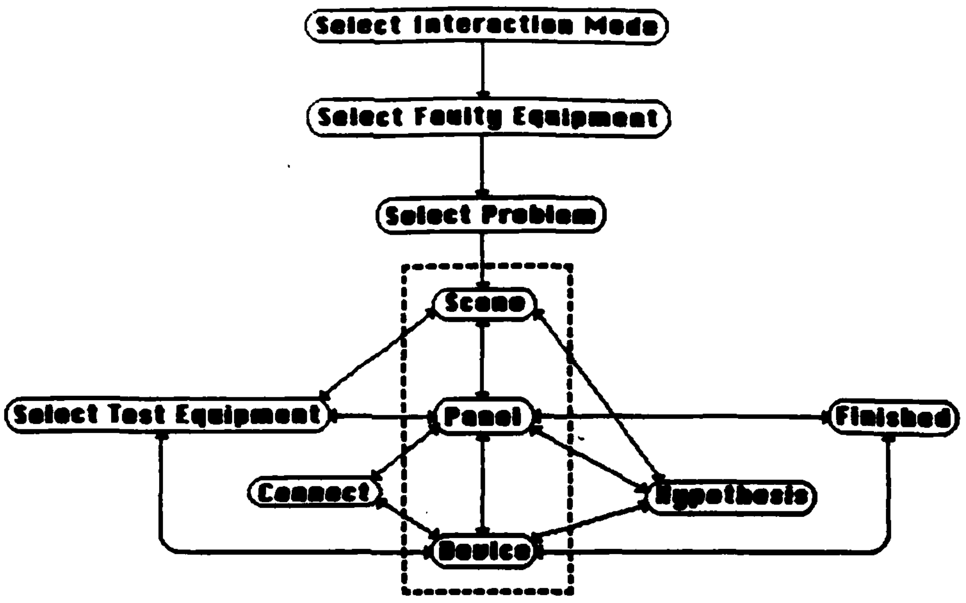


Figure 5: State Diagram Showing Possible Student Movement in EPST

is available for that session. The types of on-line help are described in Chapter 4.

3.2.1 EPST State Definitions

The EPST system states, defined in Paulsen et al. [19], are briefly described in Table 2. The Select Interaction Mode, Select Faulty Equipment, and Select Problem states are used to set up the troubleshooting problem for the user and initialize the necessary variables. Actual equipment troubleshooting is done in the remaining seven states, where the Scene, Panel and Device states are used to actually examine or view parts on a piece of equipment.* The Scene, Panel, and Device states in the EPST system, enclosed by dotted lines in Figure 5, correspond to the scene, panel, and device nodes on an equipment tree. Students can view either the faulty equipment tree or a test equipment tree, but only one tree at a time. Thus, when a student is moving among the

*If the video picture or graphics is not available in a scene, panel, or device node, a menu containing a textual listing of the panels, devices, testpoints, probes, or device settings available for selection will be displayed. This menu will also be used for testing the database before video is available.

Table 2: EPST States

State	Description
Connect	Allows the student to change test equipment connections and get test equipment readings.
Device	Displays a selected device node that is on the current equipment tree.
Finished	Evaluates whether the student solved the problem or not.
Hypothesis	Allows the student to update an hypothesis file with an hypothesis, proposed tests, data obtained from tests, or conclusions.
Panel	Displays a selected panel node that is on the current equipment tree.
Scene	Displays a video picture showing a full view of the equipment system currently under examination.
Select Faulty Equipment	Used to select a faulty equipment database for troubleshooting practice.
Select Interaction Mode	Used to select a student interaction mode.
Select Problem	Used to select a troubleshooting problem
Select Test Equipment	Allows a student to select a piece of test equipment in order to view its equipment tree or set it up for testpoint readings.

Scene, Panel, and Device states in the EPST system, the student is conceptually moving among the scene, panel, or device nodes on an equipment tree.

The remaining four states, Select Test Equipment, Connect, Hypothesis, and Finished, are used by the students to aid in troubleshooting and developing problem solving strategies. The Select Test Equipment state allows students to select a new equipment tree for viewing, since the student may view only one equipment tree at a time. The Connect state allows users to view existing test equipment connections and make new connections or change old ones. Setting up test equipment and obtaining test equipment readings allows students to test the validity of their hypotheses. The Hypothesis state allows users to record their hypotheses and any associated information. It serves as a scratch pad for students to record hypotheses, proposed tests, and results from those tests. The Finished state evaluates the user's solution to the troubleshooting problem and allows the user to view other solutions. These last four states are accessed by selecting a command from the command line, and in all but the Finished state movement out of these states is actually a return to the Scene, Panel or Device state from which the original command was selected.

3.2.2 Student Position Representation

During troubleshooting, a student's current position is always represented as a scene, panel, or device node on the current equipment tree. A student can view only nodes on one equipment tree at a time. A current pointer (**curr_equipment**) is kept which always points to the current database record, and is updated whenever a new database is selected in the Select Test Equipment state or as a result of selecting the Jump command.

The current node information is stored in the associated database record (**database_rec**) in the curr_node field. Thus, when a user moves back and forth between equipment trees during a problem, the current node on the current piece of equipment can always be displayed.

3.2.3 Student Movement on Equipment Trees

Student Movement in EPST provides a more realistic simulation of a troubleshooting environment than was available in previous trainer/simulators, such as EEMT [7]. EEMT allowed students to access different parts of the faulty equipment only in terms of the hierarchical structure (Scene - Panel - Device) and students could not access test equipment at all. In contrast, EPST not only allows students to access different parts of the faulty equipment by traversing up and down a similar hierarchical structure, but also allows access to different parts of the test equipment in the same way. EPST allows students to move between the faulty equipment and test equipment, freely setting up test equipment and test connections in order to get test equipment readings. EPST also allows students to follow actual wire connections, which has not been possible in the past. Allowing students to set up test equipment and follow wire connections are major advantages of EPST since they provide a more realistic trainer and should help students develop a better cognitive model of troubleshooting.

Students initially examine or troubleshoot faulty equipment by traversing through the faulty equipment tree semantic network, viewing scene, panel, and device nodes. The Scene - Panel - Device (IS-PART-OF) hierarchical structure was used as the basis for student movement in EPST because of the proven success of the use of this type of structure in EEMT and GMTS systems [13], and because EPST was designed to replace EEMT in Navy training schools. Movement within an equipment tree semantic network is accomplished by selecting

1. the next panel or device node to be viewed,
2. the **Up** or **Scene** command, or
3. the **Follow_wire** command.

The user moves down the IS-PART-OF relation in the equipment tree by selecting the next desired panel or device node from the current node's video/graphics display.* The areas available for selection by the student will be stored in the selection slot in the frame attached to each child of the current node. In all cases, the sta..dard triangles are searched for all triangles that have the current node as the object, the IS_PART_OF or IS_TESTPOINT_OF relation as the relation, and any node as the subject. When menus are used, the subjects of all triangles found will be included in the menu as the panels, devices, or testpoints available for selection from the current node. When video/graphics is used, the selection areas of the subjects of all triangles found will be used to determine the selection areas from the current node. Referring to Figure 3, the

*User will select from a menu display if video/graphics is not available.

panels that would appear on the AN/WSC-3 Scene Node Menu would be the Antenna Control Panel, WSC-3 Panel, and Control Indicator Panel.

Moving up the IS-PART-OF relation is controlled by selecting either the **Up** or **Scene** command. The Up command symbolically moves the student up one node on the equipment tree to the parent of the current node. This is accomplished by searching the standard triangle nodes for the node that contains the current node as the subject, the IS-PART-OF relation as the relation, and any node as the object. When this triangle node is found, the object of the triangle will contain the index to the parent node of the current node. For example, if the current node in the AN/WSC-3 equipment tree, shown in Figure 3, is the Bite Switch, the triangle that will be found is [**Bite Switch** is-part-of **WSC-3 Panel**]. The WSC-3 Panel will then be displayed to the student.

The **Scene** command symbolically moves the student to the root node of the equipment tree and displays the scene node. An index to the scene node is kept in each equipment tree database record to avoid having to do extensive triangle searches, especially when the student is many nodes away from the scene node.

The **Follow_wire** command is used to allow students to follow wire connections both forwards (tracing the output) and backwards (tracing an input). When selected, the Follow_wire command displays a list of all inputs and outputs for the current device, along with their values. The student can then use a Negation Strategy [9] for problem solving, where the student traces known correct signals, determining where the fault has been inserted. If the student can narrow down the faulty device to a certain path of devices, the student can examine the devices along this path and determine whether the outputs for each device are consistent with the inputs. If the outputs are normal, then the student can usually conclude that the device is operating correctly. The student will then select the next device along the path, and determine if it is good. Once the student finds a device where the output is not consistent with the inputs, then the faulty device has been found. The student will then replace the device and confirm that the problem was solved.

When a student wants to trace an input, the student will select the desired input from the menu. The device whose output is connected to the input is then displayed. If the student selects to view this device, the device becomes the current device and its inputs and outputs are shown. If the student wants to trace an output, then the desired output is selected from the menu. A list of devices whose inputs are connected to the output is displayed, and the student may select one of these devices to view. If a new device is selected, then it becomes the current device, and its inputs and outputs are shown.

The way the knowledge is represented (stored) in the database, in the form of input and output triangle nodes, allows EPST to easily find which outputs are connected to inputs. For example, consider the simple connection of two AND gates, shown in Figure 6. The output of device 1 is used as an input to device 2. The value of output_A on device 1 is therefore the same as the value of input_1 on device 2, and is stored in a quantifier node (QNT_1). This information is stored in the semantic network in the form of input and output triangle nodes. These triangle nodes would symbolically look something like:

Triangle 1:	**Device1**	*Output_A*	**QNT_1**
Triangle 2:	**Device2**	*Input_1*	**QNT_1.**

In actuality, the subject, relation, and object fields contain indices to the specific nodes that contain the names or values. For example, QNT_1 actually contains the index of the quantifier node which contains the value. Using this structure, there is only one copy of the value (the number in QNT_1) to worry about for updating. The common part of the two triangle nodes above is the Qnt_1 node. In order to find the output whose value is used for input_1 on device2, EPST searches the output triangles of the current equipment tree for any outputs that have the same quantifier pointer (like triangle 1 above). After finding the

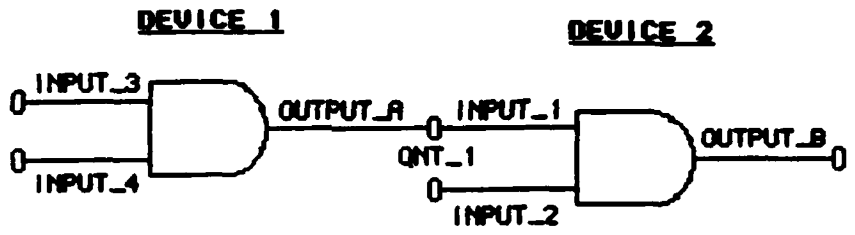


Figure 6: Connection of Two **AND** Gates

triangle, the source device and output are known, and are displayed to the student. Since all physical (wire) connections between devices are represented in this manner in the EPST database, students can follow wires from one device to another (either forward or backward), until they find the source of an erroneous signal.

Movement between the faulty equipment and test equipment is accomplished by selecting the **Test_equip** command or **Jump** command. The **Test_equip** command displays the Select Test Equipment state, which allows the student to select any piece of test equipment available for the current problem. If a piece of test equipment is selected, that piece of equipment becomes the current piece of equipment, and the student will view the current node for that test equipment tree. The current node is stored in the database record for the equipment tree, and is either the scene node if the tree is being viewed for the first time, or the last node that was viewed on the test equipment tree. The **Jump** command is used to jump back and forth between the current node on the faulty equipment tree and the current test equipment tree. The current test equipment tree is the last equipment tree that was viewed by the student. The Jump command is available only after at least one test equipment tree has been viewed.

3.2.4 Student Path Information

The ability of the system to store and replay the student's solution paths is very important for a reactive learning environment. It allows a student to view what he has done. In EPST, the student path information is stored in a sequential list so that student movement can be analyzed after a troubleshooting problem is completed. The hypothesis file is available to the student during the troubleshooting session. This student path information will also be used later by instructors to find student errors and weak points in the database.

Being able to store the student's solution path is even more powerful when an advisor or coach is added.* The advisor or coach can determine which aspects of the audit trail should be enhanced in order to help students "discover" their misconceptions or shortcomings [6].

EPST student path information is stored in a global list, **problem_path**, which has been initially defined as a linked list. This data structure allows for the unknown length a student path *could have*, and allows for storage of different amounts of information. Each time an equipment tree node (Scene, Panel, or Device) is displayed, the equipment database index and node name are added to the *problem_path* list. When the Select_test_eq, Connect, Hypothesis, and Finished states are entered, the name of the entered state will be added to the list.

This initial attempt at storing path information is simplistic and will be used mainly for debugging programs. At a later time, a more sophisticated path storing mechanism can be added that can record everything a student has done, so that the information can be used by both the student and an advisor/coach during troubleshooting.

*The addition of an advisor/coach module is discussed in Chapter 6.

3.3 Obtaining Test Equipment Measurements

Another important part of the EPST student interface is that it allows students to take measurements by setting up test equipment and actual testpoint connections. Earlier trainers, such as EEMT, did not allow students to set up test equipment. They were allowed only to select testpoints that were preconnected to test equipment and a reading then was displayed. This meant that the system could use only connections that had been predefined and preset in the database. Allowing students to make any test connection they choose provides a more reactive learning environment since it gives students the freedom to experiment. It also provides a better simulation of a real world troubleshooting environment.

In EPST, students traverse the test equipment tree in the same way as they move through the faulty equipment tree, except they cannot traverse the input/output wire connections. Students move up or down the test equipment tree using the IS_PART_OF relation, viewing panels and devices, changing device settings, and connecting test probes to the faulty equipment if they are available on the current piece of test equipment. Then, in the Connect state, students set up testpoint and test equipment connections and are able to view test equipment readings. Students are not confined to using only predefined testpoint connections to test equipment, but are free to use any testpoint and test equipment and take any testpoint reading they desire. They may set up wrong testpoint and test equipment connections or have several pieces of test equipment connected to the faulty equipment at one time. Students also can have several different probes from different pieces of test equipment hooked up to a single testpoint. Since EPST will use graphic overlays, not all possible test point connections to test equipment have to be predefined by the Subject Matter Expert when creating the database.*

*Some predefined connections could be used in the advisor module to help give advice to students on what tests are appropriate at certain points in the troubleshooting process.

3.3.1 Connect State

The Connect state allows students to change test equipment connections and get test equipment readings. The student selects the Connect state by selecting the Connect command while in another state. The Connect state displays a menu containing current test equipment probe connections and the current connection being set up.

A probe connection in the Connections Menu includes the name of the test equipment, the test equipment probe-input,* probe, and testpoint that make up the connection. Probe connections to test equipment probe-inputs are predefined by SME's in the authoring mode and can not be changed by students.

The first connection in the Connections Menu will be the current connection being set up. The current connection being set up consists of the current piece of test equipment and the last probe selected. If the student selected a testpoint just prior to entering the Connect state, that testpoint will be displayed in the current connection, connected to the last probe selected on the current piece of test equipment. The current connection being set up will always be shown in a different color from the rest of the Connections Menu and marked by ">>."

The current connection is changed by selecting a command from the command line. The Connect state commands available for changing current connections are briefly described in Table 3.

When the New_Probe or Select_new_test_equipment command is selected, a menu is displayed in the lower right-hand corner of the screen, being overlaid on top of the Connections Menu if necessary, from which a new probe or new piece of test equipment is selected. The Probe Menu will contain the probes available for the current test equipment. The Test Equipment Menu will contain the test equipment available for troubleshooting the faulty equipment. After the

---

*The probe-input is the name for the place on the test equipment where the probe is connected.

Table 3: Connect State Commands for Changing Current Connections

Command	Description
Adjust_test _equipment	Moves student to the Select Test Equipment state where the student selects a piece of test equipment in order to view its equipment tree and set it up for test equipment readings.
Make _connection	Makes the current connection on which the student was working a completed connection.
New_probe	Allows student to select a new probe for a connection.
Return	Returns student to the state from which the Connect state was entered.
Select_new _test_equipment	Allows student to select a new piece of test equipment to connect to.

student selects one of the available probes or test equipment from the menu, the selected item is placed in the current connection at the top of the Connections Menu under the appropriate heading. If the probe is changed, the probe-input associated with the selected probe also replaces the old probe-input in the current connection. If a new piece of test equipment is selected, the current probe is erased. After the student selects a new probe or piece of test equipment, the Probe or Test Equipment Menu is erased and the student selects another command from the Connect state command line. Once the student has finished changing the current connection, the student completes the changed connection by selecting Make_connection from the command line.

<u>3.3.1.1 Make_connection command.</u> Selecting the Make_connection command makes the current connection on which the student has been working a completed connection and adds the completed connection to the Connections Menu. The newly completed connection will be marked by the keyboard cursor in the Connections Menu. Any previously completed connection for the same test equipment that contained the same probe as the newly added connection will be deleted. Once the connection is completed and the Connections Menu is updated, the updated Connections Menu and new command line will be displayed.

The commands available for use after the connection is completed are described briefly in Table 4. When selecting one of these commands, the connection that has the cursor next to it will be the connection that the student will see the value for. After selecting both the desired connection and the Measurement command, the student moves to the state corresponding to the panel or device node last visited on the test equipment tree and sees the video/graphics picture of the panel or device and the test equipment reading. After the desired connection and the Value command are selected, the test equipment reading will be displayed. The Quantifier command will appear on the command line only in the Tryout mode, and is used only by an SME. Selection of the Quantifier command allows an SME to view the value of the quantifier for the test equipment connection made.

3.4 Scoring in EPST

EPST provides several mechanisms for recording how well students do on particular problems. These mechanism are: time, cost, and problem score. These scoring mechanisms allow instructors to quantitatively judge how well students are doing, and also provides a sense of gaming or competition to the system for student motivation during practice sessions. The time kept represents the time spent by the student on the troubleshooting problem. The time is recorded at the beginning and end of the problem, and the difference between the two is the time spent. The cost and problem score are defined in EPST as generally as possible, allowing different system users to modify them to suit their needs. The cost for a particular troubleshooting problem normally consists of only the total cost of all replacement parts used. The replacement cost of a device is stored in the frame associated with the generic description of the device in the parts tree. The replacement cost of a panel is stored in the frame attached to the panel node in the equipment tree. Other costs or bonuses could be added, such as giving the student a bonus for solving the problem under a certain time or for using fewer test connections.

Table 4: Connect State Commands Available After Completing Connection

Command	Description
Measurement	Displays the test equipment reading and moves the student to the state corresponding to the last node viewed on the test equipment tree.
Quantifier	Allows an SME to view the value of the quantifier for the test equipment connection.
Return	Returns student to the Connect state, redisplaying the connect command line.
Value	Displays the test equipment reading only and leaves student in current state.

Problem scoring is associated with the different types of on-line help available. In trying to design a way to score the types of help used, two different ways are commonly used. The first way to score a student is to start the student with a score of zero, and each time on-line help is used, add a set amount of points to the student's score. The student's goal is to finish with the lowest score possible. The second way is to start the student with some maximum score, say 200 points, and subtract a set deduction from the student's score each time on-line help is used. The student's goal in this case is to finish with the highest score possible. EPST allows both of these types of scoring, allowing an SME or instructor to actually specify the values to be used. An external file (called Score_definitions) exists that can be edited by an SME or instructor in order to allow them to specify the initial value of problem_score and set the values of the individual on-line help deductions. If the help deductions are given a positive value, it will be added to the problem score if that type of help is called. A negative value will result in the value being subtracted. Defining scoring in this way allows system users to modify the scoring to suit their needs and allows them the freedom to experiment with different combinations of scoring. The amount of the deductions can also be modified as students gain more experience.

3.5 Solving the Problem

After the student reaches a solution, makes the necessary changes to the faulty equipment (either causing a state change within the faulty equipment that corrects the problem or replacing a panel or device), and verifies that change(s) to the equipment have corrected the *malfunction*, the student proceeds to the Finish state. The Finish state evaluates whether the student solved the problem or not, and gives the cost, time, and score associated with the solution if it was correct. The Finished state displays the student's scorebox containing information about the student's solution to the troubleshooting problem and a menu containing options on what the student can do next. The options displayed in the Options Menu depend on the student's interaction mode. The options available allow the student to:

1. view an expert's solution to the problem,
2. view an expert's hypothesis file,
3. view the student's hypothesis file, and
4. view any help information available for the troubleshooting problem.

Allowing students to view an expert's hypothesis file and problem solution helps students learn different problem solving strategies and techniques. Students can also review their hypothesis file and compare it to the expert's. After completing the Finished state, the user may start another problem or exit the system.

CHAPTER 4

HYPOTHESIS FORMULATION

Developing strategies is an important concept for teaching students skills in troubleshooting equipment. A maintenance person who is out on a ship thousands of miles at sea needs to be able to develop strategies to fix unusual problems on familiar equipment or on new pieces of equipment for which no specific training has been available. That person just cannot take a time consuming hit or miss approach, especially when the piece of equipment that has failed is vital to the operation of the ship. The repair person must be able to methodically develop a test strategy in order to find and repair the problem as quickly as possible. Providing help in forming hypotheses and developing strategies can help teach these skills.

Brown [5] says in order to facilitate this style of learning in a reactive learning environment, the student must be encouraged to formulate, test and witness the consequences of his own ideas and must be freed from worry about possible catastrophic consequences. He also says that the system should be designed to criticize the student's ideas. A program that helps students form ideas and then criticizes those ideas allows students to learn from their mistakes.

4.1 Goal-Directed Hypotheses

Hypotheses should relate to goals and subgoals. Usually the overall goal when troubleshooting a piece of faulty equipment is to fix the equipment. An example of a subgoal might be: determine whether a certain device is faulty or not. This subgoal can be reached by using testpoints to look at the device inputs and outputs, and determine whether they are correct or not. If an input is wrong, the student knows the device is good. The student could then trace the input to the source of the erroneous signal. If the inputs are all good, but the output is bad, then the student would deduce that the device is faulty and replace it. In either case, the subgoal has been reached. It is important for the student to set such goals and be aware of what is needed to achieve them.

4.2 Help in Forming Hypotheses

What EPST now offers is limited help to the students in forming hypotheses. The Hypothesis state provides a scratch pad for students to use in developing hypotheses and storing information associated with the troubleshooting problem. By providing this type of on-line scratch pad, EPST allows students to see their hypotheses in written form, making it easier for them to develop and modify their goals and subgoals. The Hypothesis scratch pad also allows students to store information, so students do not overload their working memory.*

Students analyze and improve their troubleshooting strategies by formulating hypotheses, then testing and witnessing the consequences of them. On-line help is available to provide additional information about the equipment to students to aid them in forming hypotheses. On-line help can also present possible hypotheses to the students, depending on their position in the faulty equipment tree.** An example of on-line help that can be offered is the device_hint help command, which helps a student decide which type of device should be checked first. Since the faulty device for the current problem is known, information in the parts and equipment trees can be used to give hints to the student on which device is faulty. For example, if a modulation switch is

*Working memory, according to the ACT theory, stores what the problem solver currently knows about the problem [2].

**Possible hypotheses can only be given if they have been entered by an SME. These hypotheses are stored in frames attached to equipment tree nodes.

the faulty device, the hint could come from one level higher on the parts tree (refer to Figure 2) and tell the student that a switch is faulty. A less direct hint could come from two levels higher and state that a mechanical part is bad. These hints would be given when asked for by the student or, when an advisor or coach is added, when deemed necessary by the advisor/coach module. Thus, by combining what is known about the faulty equipment, and where the student is, EPST provides limited advice to students and helps them form hypotheses and develop problem solving strategies.

In the future, student learning can be enhanced by an advisor module that critiques student hypotheses and offers advice, as is done in SOPHIE. This idea is discussed under future research in Chapter 6.

The different types of on-line help available in EPST are discussed in the remainder of this chapter.

4.3 On-Line Help

In order to provide the student with a realistic learning environment, it is necessary to provide as much help as possible, similar to what could be received from an instructor if one were present. The way in which knowledge is represented in the EPST database permits various types of on-line help to be made available to the student.

EPST's student interface was designed to provide many different forms of on-line help to students, depending on the mode of student interaction. Help available in each mode is defined in Paulsen et al. [19], and in Chapter 3, Section 3.1. The student can access on-line help by selecting the help interrupt. The commands available under the help interrupt are:

1. Definition
2. General
3. Specific
4. Highlight
5. Problem
6. Cost.

The Definition and General help commands will be implemented system wide in CBESS, and are not discussed at length in this thesis. Each of the other commands listed above is explained briefly below.

4.3.1 Definition Command

If a student encounters a word and is unsure of the meaning, the Definition command allows the student to see a full definition of the word, provided it is in the Language Skills Computer Assisted Instruction (LSCAI) database and the LSCAI database is present.

4.3.2 General Command

The General command provides help in using the CBESS system (including the interface) and is not related to the subject matter. Examples of help available under the General command includes descriptions of: how to enter various types of answers, how to use lesson control keys, and where to find special keys on the keyboard.

4.3.3 Specific Command

The Specific command allows students to access any on-line help that is available for helping the student troubleshoot, develop hypotheses, or develop problem solving strategies. It provides specific information on devices, the faulty equipment, and the troubleshooting problem. Much of the help offered in specific help would not be available in an actual troubleshooting environment but would be available in the classroom, where an instructor would be present to help students, offer advice, and give hints that help focus students efforts towards a certain solution. The net effect is increased learning.

The commands available in specific help are:

1. Path Information
2. Faulty Device Hints
3. Production Rules
4. Replacement Conditions

5. Inputs/Outputs

6. Error Description

7. Possible Hypothesis

8. Possible Tests.

This list of help options is by no means exhaustive of all the possible types of help that can or should be given to a student. It represents the types of information that can be accessed from the EPST database. In the future, help options can be expanded to include instruction and troubleshooting techniques, among others. The amount of specific help available to the student during troubleshooting is limited by the interaction mode the student is currently in, and is controlled at the student interface by the commands that are made available to the student. The specific help commands are described below.

4.3.3.1 Path information. Path information is designed to help direct the student towards finding the faulty node by confirming for the student whether or not the current node on the faulty equipment tree is on the *fault path.*[*] Since the program knows the faulty node for the current problem, the student interface determines the "fault path" and the path from the scene node to the current node using the IS_PART_OF relation. These two paths are compared. If the paths are exactly the same (the student has found the faulty node) or if the current node is on the "fault path," then a message will be displayed indicating that the current node is on the "fault path" and within a certain number of nodes of the faulty node. If the two paths differ (the current node for the faulty equipment tree is not on the "fault path"), a message will be displayed indicating that the student is not on the "fault path" but is within a certain number of nodes of the faulty node.

With both messages, the distance (number of nodes) part of the message

[*]The "fault path" is defined as a concatenated list of nodes that describes the most direct path from the scene node to the faulty node using only the IS_PART_OF relation. For example, if the Bite Switch shown in Figure 3 is the faulty node, the "fault path" would be "(AN/WSC-3) / (WSC-3 Panel) / (Bite Switch)."

will be an approximation of how close the current node is to the faulty node, instead of the exact distance. An approximation is used to avoid giving too much information to the student, especially when the student is close to the faulty node. The actual distance from the current node to the faulty node is used to provide the approximation. The actual distance to the faulty node is calculated using the distance the student would have to travel on the equipment tree using the IS_PART_OF relation. The distance approximations currently used are:

a. within three nodes of the faulty node,

b. within six nodes of the faulty node,

c. within nine nodes of the faulty node, or

d. greater than nine nodes away from the faulty node.

4.3.3.2 Faulty device hints. The Faulty Device Hints command allows a student to receive descriptive hints as to which device is faulty. There is a sequence of four types of hints available to the student, each successive type being more descriptive. The four hints available are:

Category	The type of category the device falls under. All devices on the parts tree are categorized under either Mechanical, Electrical, Electromechanical, Probe, or uncategorized.
Type	The type of device it is, such as a switch or circuit board.
Generic	What kind of device it is.
Faulty device	The actual name of the faulty device.

If the modulation switch shown in Figure 3 was the faulty device, the hints given would be:

Category:	The faulty device is Mechanical.
Type:	The faulty device is a switch.
Generic:	The faulty device is a modulation switch.

Faulty device: The faulty device is the modulation switch on the WSC-3 Panel.

Each successive hint is more descriptive and would result in a larger score change. In other words, the Category hint would result in a low change to the student's score while the Faulty device hint would result in a large change.

The information needed to provide these hints is retrieved from successive searches of the triangle nodes for the faulty equipment. Since the faulty device is always known by the program for each problem, the faulty device hint simply uses this information. The generic device hint is found using the INSTANCE_OF relation. The standard triangle nodes are searched for the triangle that contains the faulty node as the subject and INSTANCE_OF as the relation. The object of this triangle will be a pointer to the name of the generic device. The type hint is found by searching for the triangle node that has the generic device as the subject and ISA as the relation. The object of this triangle node will be the type of device. The Category hint is found in the same way, using the type of device as the subject and ISA as the relation.

4.3.3.3 Production rules. Production rules describe device state transitions and are used during emulation to determine the outputs of a device based on the inputs. This specific help command displays the production rules defined by the Subject Matter Expert for the current device. This information is helpful in allowing students to determine whether a device is working normally or not. Examples of production rules and how they are used are described in detail in Chapter 5.

4.3.3.4 Replacement conditions. Selecting the Replacement specific help command allows the student to view any replacement conditions that must be met before a device or panel can be replaced. As a minimum, usually the power needs to be turned off, but there may be other conditions that must be met before replacing the panel or device in order to avoid damage to the equipment or personal injury. These conditions are checked by the program whenever a student tries to replace a device or panel. A warning message is printed out and replacement is prevented if the conditions have not been met. The replacement conditions for a device are stored with the generic device description found in the parts tree. This information is found by following the INSTANCE_OF relation to find the generic device, and then looking in the frame to get the information.

4.3.3.5 Inputs/outputs. This command displays a list of input/output values for the current device or for any device selected by the student. It allows a student to view the inputs/outputs of any device at any time. Using this type of help eliminates excessive time spent trying to move through the equipment tree to a certain device in order to use the Examine_inputs command. It also allows a student to compare inputs/outputs from similar devices. The list of inputs/outputs is retrieved by searching all input and output triangle nodes for the triangles that have the selected node as the subject. The relation of these nodes has the index of the name of the input/output and the object contains the index of the quantifier node that contains the current input/output value.

4.3.3.6 Error description. This command will actually display an English description of what the equipment error is for the current problem, without telling the student which panel/device is faulty. The English description of the equipment error is stored with each problem.

4.3.3.7 Possible hypotheses. This information will be entered by an SME or instructor, and will be connected to certain nodes on the equipment tree and stored in the frame associated with the node. The hypotheses can be used for designated problems or for all problems associated with the equipment tree. If hypothesis information is asked for and the current equipment tree node does not have any possible hypotheses information associated with it, the program will use inheritance to find the hypotheses information of the closest ancestor node that contains such information.

4.3.3.8 Possible tests. The Possible Tests command is similar to the Possible Hypotheses command described above. This command displays possible tests to perform based on the current node and problem.

4.3.4 Highlight Command

The purpose of the Highlight command is to prevent student frustration incurred after repeatedly selecting panels, devices, or device positions[*] from a video screen that are not available for selection. When the Highlight command is selected, the possible selection areas will be drawn in reverse video. A student can than see what is available for selection and make an appropriate choice. The Highlight help command will be available when video/graphics are available. When menus are used instead of video to display the scene, panel, and device nodes, the items in the menu are the only choices allowed. The information that is used in creating a menu will be used to highlight the different panels and devices on the video/graphics screen.

4.3.5 Problem Command

Selecting the Problem command displays the complaint and/or symptoms associated with the current troubleshooting problem. This is the same information that is presented to the student at the beginning of a problem, and is available anytime in EPST, without any score deductions. The complaint and/or symptoms for each problem are stored in the problem record.

4.3.6 Cost Command

Selecting the Cost command displays the replacement cost of the current panel or device being viewed when the help interrupt was selected and/or the cumulative cost for troubleshooting the faulty equipment thus far. If a panel or

[*]Video pictures will display actual pictures of devices that are currently on the equipment. Because of the cost involved, most devices that have had some device positions deactivated are not replaced. Therefore, positions may appear on the video picture that are not actually hooked up to anything.

device is not replaceable, only the cumulative cost for the current troubleshooting session is given. The replacement cost for a device is stored with the generic device description found in the parts tree. This information is found by following the instance-of relation to find the generic device, and then looking in the frame to get the information.

CHAPTER 5

EMULATION IN EPST

In EPST, each device is defined in terms of inputs and outputs, and production rules are used during emulation to determine the value of the outputs of a device based on its inputs. Keeping all information local and using reasoning from first principles offers many advantages, including making it easier to construct and maintain the overall system.

Related research in simulating circuits for electronic troubleshooting systems is being done currently by Randall Davis at MIT [8]. Davis also uses reasoning based on first principles (first order effect). This reasoning uses knowledge of structure and behavior, where structure is the information about the interconnection of modules, and behavior refers to the black box description of a component.

5.1 Use of Production Rules to Simulate Devices

Production rules are associated with part nodes and describe device state transitions. They are used in emulation to determine new outputs of devices based on the current state of a device and its inputs. The use of Production Rules are described in detail in Paulsen et al. [19], and some of the examples contained in this section are taken from that document.

5.1.1 How Information is Stored in Production Rules

A production rule consists of the following items:

Initial state	The state a device must be in for the production rule to be applicable.
Bit vector	The error conditions under which the production rule is

applicable.

Boolean expression	The premise that, when evaluated, indicates whether or not the production rule is applicable.
Conclusion	The state the device should go to if the production rule is applicable. If the state should not change, the conclusion will state "no-change."

5.1.2 Order of Production Rules

The order of production rules is important because the system uses the first applicable rule it finds.[*] As a result, SMEs need to ensure that rules are placed in the production rule list in the proper order.

For example, one rule-ordering strategy that SMEs can use is to place rules that operate on boundary states[**] towards the front of the list so that later rules can cover the general case. An example of this for a 4 bit counter is:

```
If in state 15 and operating normally and count and load are
   high, then state 0.
```

A later rule would say:

```
If in any state and operating normally and count and load are
   high, then state = state + 1.
```

Another strategy SMEs can use is to specify conditions where nothing happens first, so that later production rules need not worry about them. For example, the first rule for digital synchronous devices can often be:

```
If in any state and under any conditions and (clock <> rise),
   then no change.
```

[*]There is implicit knowledge here, since later rules need not check for situations that are covered in earlier rules.

[**]A boundary state is a state at which something special happened. For example, in state 15 a 4 bit counter would go back to state 0 rather then continue to state 16.

Rules later in the list would not have to verify that the clock rose. Using such strategies can often simplify production rules.

5.1.3 Production Rules and Error Conditions

There are two types of production rules, "normal" rules and "error" rules. Normal rules are used when a device is operating normally. Error rules are used when an SME has specified that a device is operating under an error condition. A production rule can fall into both these categories.

Each production rule has a bit vector that indicates when the rule is valid. The least significant bit of the vector is set when the rule applies under normal operating conditions.* The other bits in this bit vector signify error conditions specified by the SME. If a bit is set, the rule is valid under that error condition.

5.1.4 Sample Representation of a Device in EPST

Consider a d-flip-flop with an asynchronous clear. The following two states are defined:

```
State 0:  Q low, Q_BAR high

State 1:  Q high, Q_BAR low.
```

A subset of the production rules would be (in order):

```
1. IF
     in any state, and
     operating normally, and
     low(clear)
   Then
     state (0)
```

Implicit assumption for following rules: clear is high

```
2. IF
     in any state, and
```

*Associated with each instantiated device is a 16 bit field for giving the error conditions. If the least significant bit is set, then the device is operating normally. The other bits represent different error conditions. A device can only have one error at a time.

```
    operating normally, and
    not(rise(clock))
  Then
    no change
```

Implicit assumption for following rules: clock is rising

```
3. IF
    in any state, and
    operating normally, and
    low(d)
  Then
    state (0)

4. IF
    in any state, and
    operating normally, and
    true[empty premise]
  Then
    state (1).
```

The strategy used in defining the above rules is simple. Each rule was defined in order because:

1. *If the asynchronous clear is asserted,* the flip flop goes into state 0 regardless of what the other inputs are.
2. If the first rule does not apply, then nothing can happen unless the clock rises.
3. This rule tells what happens when d is low. Clock is not mentioned because the previous rule already indicated that the clock had risen.
4. If none of the above applied, then the d input must be high, therefore the flip flop goes into state 1.

5.1.5 Advantages of Using Production Rules

There are several advantages to the way EPST uses production rules. First, each device need only know about itself and its production rules, and not how it affects every other device. An author can easily define the way each device works in terms of inputs and outputs, and define the production rules to represent this. This information is bound to the conceptual device node, and is accessible to the program, author, and students.

Second, this representation saves the Subject Matter Expert (SME) much time and effort. For each device, the SME must define one concept node with its associated production rules. This device definition then can be used with any number of instantiations of the device with no extra effort on the part of the SME.

Third, the amount of space required to store a single set of production rules for all instantiations of a device versus separate production rules for each instantiation is far smaller.

Finally, EPST makes it easy to modify a device once a database has been created. If all the input and output connections remain the same, all that has to be done is to redefine the device in the parts tree. If the inputs and/or outputs change, the authoring mode can be used to add, delete, or edit inputs and/or outputs.

5.2 Emulation Algorithm

The emulation done in EPST is not a true simulation of the operation of the equipment, but an approximation of what actually happens. Emulation is done by performing actions on devices. An action on a device is triggered whenever one of its inputs changes, either as a result of an action on another device or as a result of the user performing an action that affects the device directly, such as throwing a switch. The devices awaiting an action are added to an agenda, called a waiting actions list, which lists the devices for which production rules need to be evaluated. The production rules for each device on the waiting actions list are evaluated, and any changes to the outputs of the device are recorded. After the waiting actions list is empty, each output that was changed is examined, and all devices that have the changed output as an input are placed on a new waiting actions list. This new waiting actions list is then acted upon. The process continues until the waiting actions list is empty.

Thus changes ripple out through all the affected devices by the emulation algorithm. The data structures used in this algorithm are:

Waiting Actions List
List of devices for which production rules need to be evaluated.

Waiting Changes List
List of quantifiers which need to be updated when the waiting actions list is empty.

Changed Quantifiers List
List of digital quantifiers that were changed by the last set of actions. This list allows the program to go back and change rise and fall values to high and low.* This list only contains digital quantifiers.

The algorithm assumes that there is a discrete system with no cycles. The algorithm for emulation is shown in Figure 7. By using an agenda in the emulation algorithm, the possibility exists for allowing students to have a single step mode, where emulation could be stopped after each pass through the algorithm. This is not possible in real life, but may be useful for teaching purposes. The current version of EPST does not permit this single step mode.

5.3 Sample Emulation

To better understand the emulation algorithm, consider the simplified example shown in Figure 8, where there are six devices, numbered 1 - 6. These six devices are connected by inputs and outputs, where the outputs of some devices are inputs to other devices. Therefore, if the output of a device changes due to a state change within the device, the corresponding input to the other devices that use the changed output as an input will also change. The resulting changed input may or may not cause a state change within the other devices. Device 1 is connected to devices 2, 3 and 4 in such a way that if

*EPST treats digital transactions (a transition from low to high or high to low) differently from other transactions in EPST. The reason is that EPST may need to know both the current value of a digital input and whether that input has risen or fallen. During emulation, if a quantifier has the value low and a new value of high is to be placed in the quantifier, EPST places the value rise in the quantifier and adds the quantifier to the Changed Quantifiers List. Similarly, if the quantifier value is to be changed from high to low, EPST places the value fall in the quantifier and adds the quantifier to the Changed Quantifiers List. When the current Waiting Actions List is empty, EPST changes rise values to high and fall values to low [19].

1. Emulation starts when an individual input quantifier changes. Devices which use this quantifier as an input are placed in the Waiting Actions List. If the quantifier is a digital quantifier, it is placed on the Changed Quantifiers List.

2. If Waiting Actions List is empty then EXIT.

3. For each device in the Waiting Actions List DO:

   a. Evaluate the production rules for the device, using the state of the device and the current input values as inputs.

   b. Record new state of device in state_fld of node.

   c. The outputs and their values are placed in the Waiting Changes List.

4. The Changed Quantifiers List is traversed, and the quantifiers updated to their new values (*rise* values changed to *high* and *fall* values changed to *low*).

5. For each output quantifier on the Waiting Changes List DO:

   a. Update the quantifier of each output with the new value.

   b. For each output quantifier that is changed, all devices that have this quantifier as an input are placed in a new Waiting Actions List.

   c. If the output value is digital, place the quantifier on the Changed Quantifiers List.

6. Any changed screen graphics are updated.

7. Jump to step 2.

Figure 7: Emulation Algorithm Used for Rippling Out Device State Changes

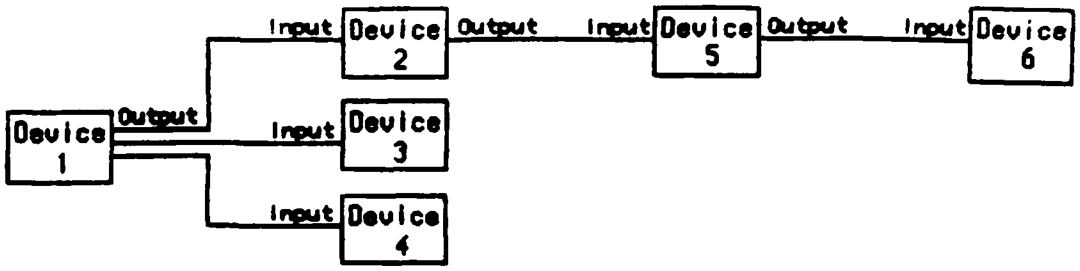


Figure 8: Simulating Devices Example

device 1 changes states, it may or may not cause a corresponding change to the state of the other three devices. Similarly, devices 2 and 5 are connected to devices 5 and 6, respectively. So if a state change in device 1 results in a new input to device 2 which causes a state change in device 2, then the output of device 2 will change, resulting in a new input to device 5. If the new input to device 5 causes a state change, device 6 will have a new input which could cause a state change within device 6.

If we assume that the state of a device always changes whenever one of its inputs changes, then EPST would emulate a state change in device 1 as shown in Figure 9. In this example, the waiting actions lists have been numbered for purposes of clarity only, since only one waiting actions list is actually used. Four passes through the emulation algorithm are required, because the algorithm repeats for each new waiting actions list until an empty list is encountered.

```
**Input to device 1 changes -> Add device 1 to list(1)

PASS 1          Waiting Actions List(1)
                       Device 1

**Output of Device 1 changes -> Add Devices 2,3,4 to list(2)

PASS 2          Waiting Actions List(2)
                          Device 2
                          Device 3
                          Device 4
**Output of Devices 2,3,4 change -> Add Device 5 to list(3)
PASS 3          Waiting Actions List(3)
                       Device 5

**Output of Device 5 changes -> Add Device 6 to list(4)

PASS 4          Waiting Actions List(4)
                        Device 6

**Output of Device 6 changes -> Finished
```

Figure 9: Sample Emulation in EPST

CHAPTER 6

FUTURE RESEARCH

This final chapter summarizes the results of our research and discusses some possible areas for future research.

6.1 Summary of Thesis

The EPST student interface provides a user-friendly reactive learning environment that helps students practice troubleshooting faulty equipment and develop troubleshooting strategies. Students develop, record and set up test connections to test their hypotheses, eventually determining and replacing the faulty component on the equipment. On-line help is available to help students form their hypotheses and to provide general information on the faulty equipment. EPST uses a combination of Semantic Networks and Frames to represent knowledge in the database, which provides for easy storage and fast access. The knowledge contained in the Semantic Networks is used to represent equipment structure and the relationships between devices in terms of inputs and outputs. Frames are used to store attributes associated with different pieces of equipment. Students access different parts of the faulty equipment by following spatial, physical, or logical links.

Devices in EPST are described in terms of structure and behavior. The structure of devices is defined in terms of inputs and outputs and the behavior of devices, which determines how outputs are derived from inputs, is defined in production rules. An emulation algorithm that uses an agenda is used to emulate the characteristics of devices. EPST was designed for the Navy for use in its technical schools, and an experimental prototype was written in "C."

6.2 Reiteration of Design

In designing any type of system, a cycle of design (DESIGN - TEST - EVALUATE - REDESIGN) should always be used to ensure a user effective product. This cycle of design was used on a small scale in designing the EPST prototype, but now some long-term testing is needed to really see how well the reactive environment works in allowing students to practice troubleshooting faulty equipment and to develop problem-solving skills. This type of long-term testing requires a large student population and the proper training environment. For these reasons, the Navy will conduct this long-term testing in its technical schools. The long term results of EPST will not be seen until after students are out in the field, which takes upwards from six months to a year. Based on the results of these tests, a reiteration of the design process should be used to redesign EPST.

6.3 Addition of an Advisor/Coach

The EPST student interface creates a reactive learning environment that allows students to formulate, test, and witness the consequences of their ideas without worrying about possible consequences. The current prototype, however, can not evaluate student actions or offer advice. Often students see things differently from instructors, and the program should be able to detect these different perspectives and then help alter them. A computer-based Advisor/Coach could do this. The Advisor/Coach should be able to determine when the student has made an error, and then judiciously decide when it is appropriate to interrupt. If the advisor interrupts too early, the student will not be able to learn from his own mistakes and learn to make corrections. If the Advisor/Coach interrupts too late, a valuable learning experience may be lost.

An Advisor/Coach could have multiple roles, offering timed advice, advice based on student performance, and student-initiated help/advice. For example, timed advice could have the Advisor/Coach track and periodically advise students whether they are on the right path or not and how they are doing in comparison to other students. The Advisor/Coach could interrupt students based on their performance and offer constructive advice when they are doing something wrong such as replacing an expensive device without first performing some cheaper tests to ensure it is the faulty device. Students could request help/advice at any time while troubleshooting. Examples of possible requests could be: what type of test should I perform now and why, is my hypothesis good, or is an action dangerous or not.

An Advisor/Coach could include an hypothesis evaluator that would inform the student when and why an hypothesis was inappropriate or appropriate. It could also offer possible hypotheses to students, based on where they are, what tests they have performed, and what their current hypothesis is. This type of help could be in the form of multiple choice questions for the student or as specific answers that might include a short explanation of why the hypothesis is appropriate.

In SOPHIE [5], an expert module used decision trees annotated with schema to produce explanations for troubleshooting a circuit. The annotations are associated with nodes in a decision tree, nodes which can be reached by only one path. Thus, the exact context is known ahead of time. Although EPST uses a more complex environment than SOPHIE, a similar idea may be used in EPST.

6.4 Natural Language Interface

The current EPST student interface offers on-line help, but this on-line help is limited by the number of commands offered to the student. This limitation could be removed by adding a Natural Language Interpreter that allows students to ask natural language questions of the database. In this way students could ask and receive any type of information available from the database. This also could decrease overall response time in help, since students would not always have to go through many levels of questions before actually receiving the information, but can ask for the specific information they desire. This procedure could be similar to the one Risa Stewart is designing [21], which provides a natural language interface that allows students to ask questions about information contained in semantic networks for a Computer-based Memorization System (CBMS).* Stewart's work could be modified to include Frames. A database browser could also be added to allow students to traverse the EPST database and examine its contents.

6.5 Efficiency

An important factor of an interactive system is its efficiency and response time. The overall effectiveness of a system is lost if a student must wait an unusually long time for responses from inputs. The EPST database was designed to increase efficiency, by decreasing search time and allowing quick access to knowledge contained in the database.

Currently each device in the parts tree contains all the information for that device. Inheritance is not used extensively in the ISA or IS_PART_OF tree. Developing a way to use inheritance with the ISA relationships would allow properties of devices to be associated with the most general object for which they are valid.** This would allow for a more concise statement of properties of the objects in the relations and would reduce storage space.

*CBMS is a component of CBESS

**The information could be attached to the node in the parts tree that contains the most general case.

REFERENCES

1 Anderson, J.R., *ACT Theory*, Harvard University Press, Boston, 1983.

2. Anderson, J.R., Boyle, C.F., and Reiser, B.J., "Intelligent Tutoring Systems," *Science*, Vol. 228, April 26, 1985, pp. 456 - 462.

3. Brandt, R. C. and Knapp, B. H., *Computer-Based Educational Software System Volume I Technical and Management Proposal*, Department of Computer Science, University of Utah, 1984

4. Brandt, R. C. and Knapp, B. H., *Sequence Editor Document*, Department of Computer Science, University of Utah, 1985

5. Brown, J. S., Burton, R. R., and DeKleer, J., "Pedagogical Natural Language and Knowledge Engineering Techniques in SOPHIE I, II, and III," in *Intelligent Tutoring Systems*, Brown, J. S. and Sleeman, D., ed., Academic Press, New York, 1982, pp. 227 - 282, ch. 11.

6. Brown, J.S., "Learning-by-Doing Revisited for Electronic Learning Environments," in *The Future of Electronic Learning*, White, M.A., ed., Lawrence Eribaum Assoc., Hillsdale, New Jersey, 1982.

7. Cubic Corporation, "Trainer Programming Report for Electronic Equipment Maintenance Trainer," Tech. report P-181/A005-1F, Cubic Corporation, Dec., 1982.

8. Davis, R., "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence*, Vol. 24, Dec., 1984, pp. 347 - 410.

9. Duncan, C., editor, *Thinking: Current Experimental Studies*, Lippincott, New York, 1967.

10. Farrell, R. G., Anderson, J. R., and Reiser, B. J., "An Interactive Computer-based Tutor for Lisp," *AAAI 84*, 1984, pp. 106 - 109.

11. Gould, J. D. and Lewis, C., "Designing for Usability - Key Principles and What Designers Think," *Human Factors in Computing Systems*, A. Janda, ed., ACM Special Interest Group on Computer & Human Interaction, The Association for Computing Machinery, Inc., New York, 1984, pp. 50 - 53.

12. Hoff, T., "NPRDC Contract NOO244-83-C-1759," Contract issued by NPRDC to University of Utah,.

13. Lahey, G. F. and Malec, V. M., Navy Personnel Research and Development Center, *Generalized Maintenance Trainer Simulator: User Manual*, San

Diego, California, 1982.

14. Lahey, G. F., "Generalized Maintenance Trainer Simulator: System Description," Technical Note 82-6, Navy Personnel Research and Development Center, Jan., 1982.

15. Landauer, T. K., Galotti, K. M., and Hartwell, S., "Natural Command Names and Initial Learning: A Study of Text-Editing Terms," *Communications of the ACM*, Vol. 26, No. 7, July, 1983, pp. 495 - 502.

16. Lemon, M. J., "A 'Less-Procedural' Methodology and Supporting Framework for Simulation Programming," Ph.D. dissertation, University of Utah, 1983.

17. Matty, D. G., "Constraint Driven Synthesis of Hardware Design," Ph.D. dissertation, University of Utah, 1983.

18. Morland, D. V., "Human Factors Guidelines for Terminal Interface Design," *Communications of the ACM*, Vol. 26, No. 7, July, 1983, pp. 484 - 494.

19. Paulsen, R. B., Collier, L. D., McKenney, D. G., Brandt, R. C., and Knapp, B. H., *Software and User Interface Definition Document - Equipment Problem Solving Techniques System*, University of Utah, 1985

20. Rich, E., *Artificial Intelligence*, McGraw-Hill, Inc., New York, McGraw-Hill Series in Artificial Intelligence, 1983.

21. Stewart, R., *A Natural Language Interpreter for the Computer-Based Memorization System*, Master of Science Thesis Proposal, University of Utah, 1984.

22. Williges, B. H. and R. C., "Dialogue Design Considerations for Interactive Computer Systems," *Human Factors Review: 1984*, F. A. Muckler, ed., The Human Factors Society, Santa Monica, California, 1984, pp. 167 - 208.