To the Graduate Council:

I am submitting herewith a thesis written by Eva R. Broadaway entitled "Development of a Transportable Knowledge-Based Assistant for Troubleshooting Industrial Equipment." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald W. Bouldin

Donald W. Bouldin, Major Professor

We have read this thesis
and recommend its acceptance:

Daniel B. Koch

Raymond K. Adams

Accepted for the Council:

Lew Winkel

Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from a reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Eva R. Broadaway

Date April 25, 1990

DEVELOPMENT OF A TRANSPORTABLE KNOWLEDGE-BASED ASSISTANT FOR TROUBLESHOOTING INDUSTRIAL EQUIPMENT

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Eva R. Broadaway

May 1990

ACKNOWLEDGMENTS

At Oak Ridge National Laboratory, a number of people in the Instrument and Controls Division contributed to the success of this project and my thesis. I wish to thank Bill Zabriskie for initiating the project and Dan McDonald for choosing me as lead engineer. Laird Williams, Abbie Roberts, Jim Mullens and Jamie Jefferis were instrumental in implementing the final system. Sam Gould was continually supportive and provided much assistance, both technical and of a project management nature. Becky Wilker and Tina Lyons of the Information Processing Center were extremely helpful in preparation of this manuscript. Brett Pate was very diligent in taking the quality photographs in this thesis. Jennifer Hall was especially helpful in preparing the drawings and in putting the copies together. Ed Anderson at the Georgia Institute of Technology's Research Institute was instrumental in helping me obtain the Knowledge FACTory II expert system software.

I also wish to thank my thesis advisor, Dr. Don Bouldin and committee members Ray Adams and Dr. Dan Koch for their comments and suggestions to improve the final version of this thesis.

Finally, I wish to acknowledge the support of the Instrumentation and Controls Division management, especially Bill Eads and Dan McDonald for providing an environment in which work such as this and the pursuit of additional education are supported and encouraged. This work was performed at Oak Ridge National Laboratory, operated by Martin Marietta Energy Systems, Inc., under Contract No. DE-AC05-84OR21400.

ABSTRACT

A transportable knowledge-based computer system was developed and implemented to assist maintenance technicians in diagnosis and repair of faults in industrial equipment. The computer system incorporates still-frame video images and expert system software to facilitate assistance to the technician. Background, design considerations, and a complete description of the system implementation are included. The implementation description consists of hardware, software, and knowledge acquisition for an initial application of the system.

TABLE OF CONTENTS

SECTION		PAGE
1.	INTRODUCTION	1
	1.1 Problem Statement	2
	1.2 Desired Solution	4
	1.3 Approach to Solution	6
	1.4 Organization of Thesis	8
2.	BACKGROUND	9
	2.1 Human Factors	9
	2.2 Computer Hardware	10
	2.3 Computer Software	11
	2.4 Knowledge Engineering	13
3.	DESIGN CONSIDERATIONS	15
	3.1 Human Factors	15
	3.2 Computer Hardware	18
	3.2.1 Computer Processing Unit and Bus	19
	3.2.2 Video Display and Storage	20
	3.3 Computer Software	22
	3.3.1 Operating System	22
	3.3.2 User Interface	23
	3.3.3 Expert System	25
	3.4 Knowledge Acquisition	26
	3.5 Design Summary	27
4.	SYSTEM IMPLEMENTATION	29
	4.1 Hardware	29
	4.1.1 Personal Computer	29
	4.1.2 Video Display	31
	4.1.3 User Input Devices	34
	4.2 Software	35
	4.2.1 System Software	36
	4.2.2 User Interface	38
	4.2.3 Authoring Interface	46
	4.2.4 Expert System	55
	4.2.5 Software Maintenance	58
	4.3 Knowledge Acquisition	59
	4.3.1 Synthetic Reasoning	60
	4.3.2 Algorithmic Reasoning	61

TABLE OF CONTENTS (continued)

SECTION	PAGE
5. SUMMARY AND CONCLUSIONS	64
LIST OF REFERENCES	66
APPENDICES	69
APPENDIX A. SYNTHETIC REASONING KNOWLEDGE BASE TEXT	70
APPENDIX B. ALGORITHMIC REASONING TOP-LEVEL PROCEDURE TEXT	80
APPENDIX C. ALGORITHMIC REASONING SUBPROCEDURE TEXT	90
VITA	93

LIST OF FIGURES

FIGURE		PAGE
1.	Typical Troubleshooting Work Flow Diagram	3
2.	Improved Troubleshooting Work Flow Diagram	5
3.	Transportable Troubleshooting Assistant Hardware	30
4.	Video Display Component and Signal Relationships	32
5.	Major System Software Component Module Relationships	36
6.	User Module Functional Relationships	38
7.	*User and Site Selector* User Module Interface Screen	39
8.	*Symptom Selector* User Module Interface Screen	41
9.	*Procedure* User Module Interface Screen	43
10.	*HyperManual* User Module Interface Screen	45
11.	Authoring System Module Relationships	46
12.	*HyperManual Browser* Authoring Module Interface Screen	48
13.	*SubSystem Directory Editor* of *Procedure Authoring System* Authoring Module Interface Screen	50
14.	*Procedure System Editor* of *Procedure Authoring System* Authoring Module Interface Screen	51
15.	*Procedure Editor* of *Procedure Authoring System* Authoring Module Interface Screen	53
16.	*Code Editor* of *Procedure Editor* Authoring Module Interface Screen	56

1. INTRODUCTION

The work described in this thesis deals with the development and implementation of a transportable, knowledge-based computer system with still-frame video pictures and expert system software which is used to provide troubleshooting assistance to a maintenance technician while working in the field. The scope of the work includes the specification of the hardware delivery system, design and implementation of the necessary software, and construction of the knowledge base. The ultimate purpose of the project is to improve productivity in the diagnosis and repair of faults in complex systems. This is to be accomplished by reducing the amount of time required to diagnosis a problem and, as an adjunct to this, to provide procedural help in repair of the problem. Improvements in productivity can only be determined with the use of the system over time. The determination of any productivity improvements is, therefore, outside the scope of this thesis and will be left to the project sponsor. The discussion focuses on the design and implementation of the hardware and software system and knowledge base, which are expected to help achieve that ultimate goal. This project was completed in 2 years with approximately $300,000 in funding.

Throughout the thesis, the term **user** refers to the technician who actually performs the troubleshooting and repair tasks. **Author** is the term used for the engineer who transfers knowledge to the system. **Developer** refers to the engineer who is doing actual design and development of the implementation platform for the system. Except for the discussion of pointing devices in section 3.1, the term **mouse** refers to any pointing device.

1.1 Problem Statement

As control systems for factory automation become more complex, troubleshooting them when problems occur also becomes more complex. This is especially true as relay control systems are replaced by programmable controllers where the control logic is found in software rather than hardware. A relatively simple failure may take hours to diagnose and repair because the information required for diagnosis and repair comes from multiple sources. Often most of the documentation available is in the form of manufacturer's manuals on particular system components which are difficult to transport to the site of the problem, and no source of information may exist which combines all component information into an integrated system. Some information necessary to troubleshoot the problem may not exist in any written form but, instead, may be rules-of-thumb that the design engineer learned while integrating the control system with the process. Additional diagnostic information may come from maintenance engineers and technicians, and operators who have previous experience with the equipment. Design documents, photographs or drawings may also be used to aid in troubleshooting. Figure 1 illustrates a typical sequence of events in diagnosis and repair of equipment faults prior to this project. All of these sources of information make diagnosis and repair of a complex control system problem difficult and time consuming.

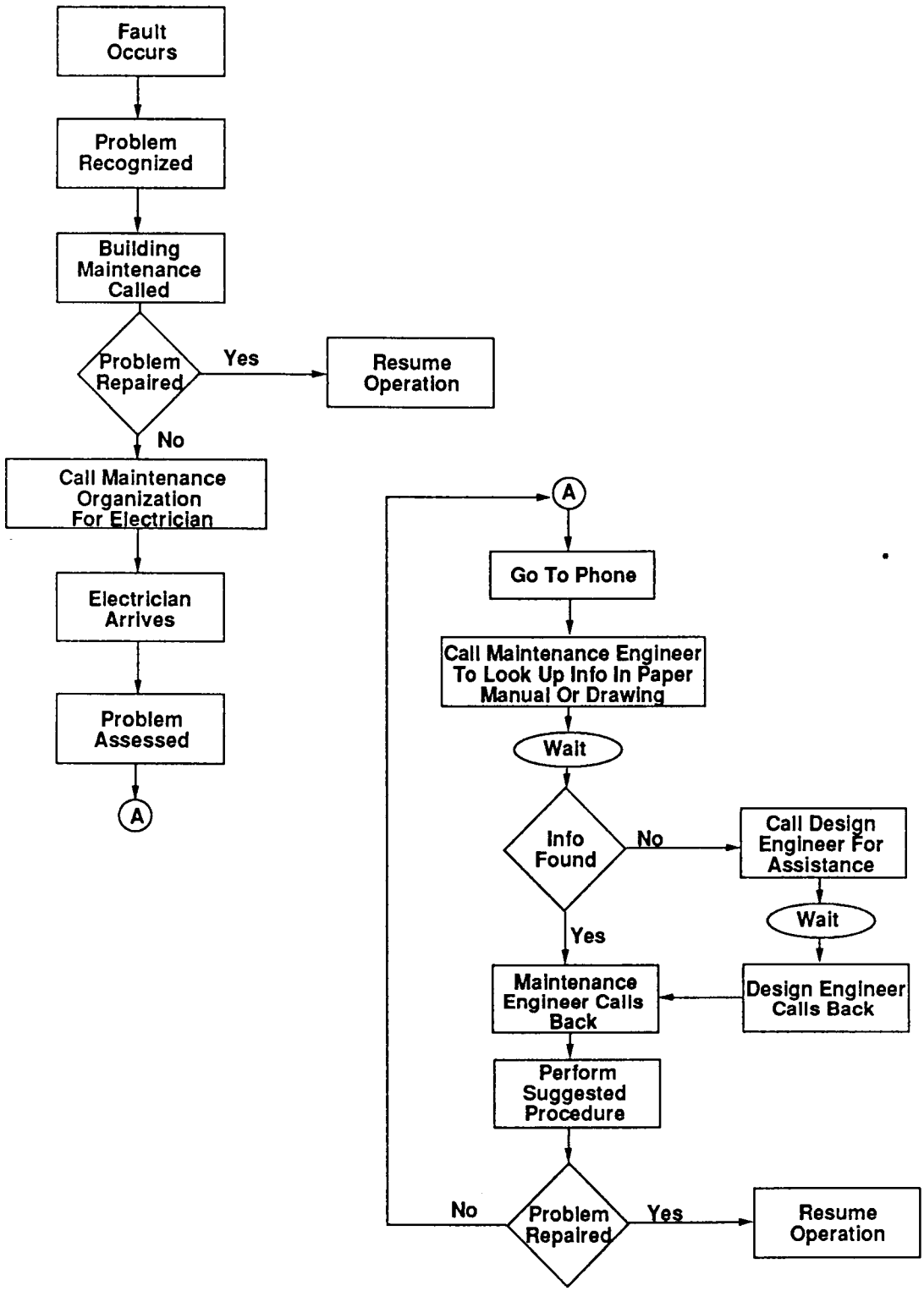


Figure 1 Typical Troubleshooting Work Flow Diagram

1.2 Desired Solution

Clearly, it is desired that out-of-service systems be diagnosed and repaired quickly. The downtime may increase costs due to idle production workers and can affect delivery dates and milestones; as the adage goes, "time is money."

The ideal solution to the problem would be a system in which all sources of information necessary to diagnose and repair the problem are integrated into a small package which the technician can easily transport to the work location. This system should provide manufacturers' manuals to the user as well as aiding in diagnosis of the problem. Repair procedures should be available and should be provided to the user when appropriate. Good quality drawings and/or photographs should be presented as needed. All information should be available to the user in a timely manner with little training necessary for effective use of the system. The information provided to the user should be appropriate to the skill level of the user: if the information is too rudimentary, the experienced technician will become bored; if the information is too sketchy, the novice will become frustrated.

In addition, since it may be desired to have similar troubleshooting assistant systems for other equipment systems, the system for authoring the assistance should be easy to use and should provide for as much re-use of modules as possible. This would eliminate the need for the author to be an expert in the use of the assistance system and allow him to focus on the real problem: diagnosis and repair of equipment failures. An improved work flow diagram using this system is shown in Figure 2. The subsequent chapters of this thesis describe the design and implementation of this transportable assistant.

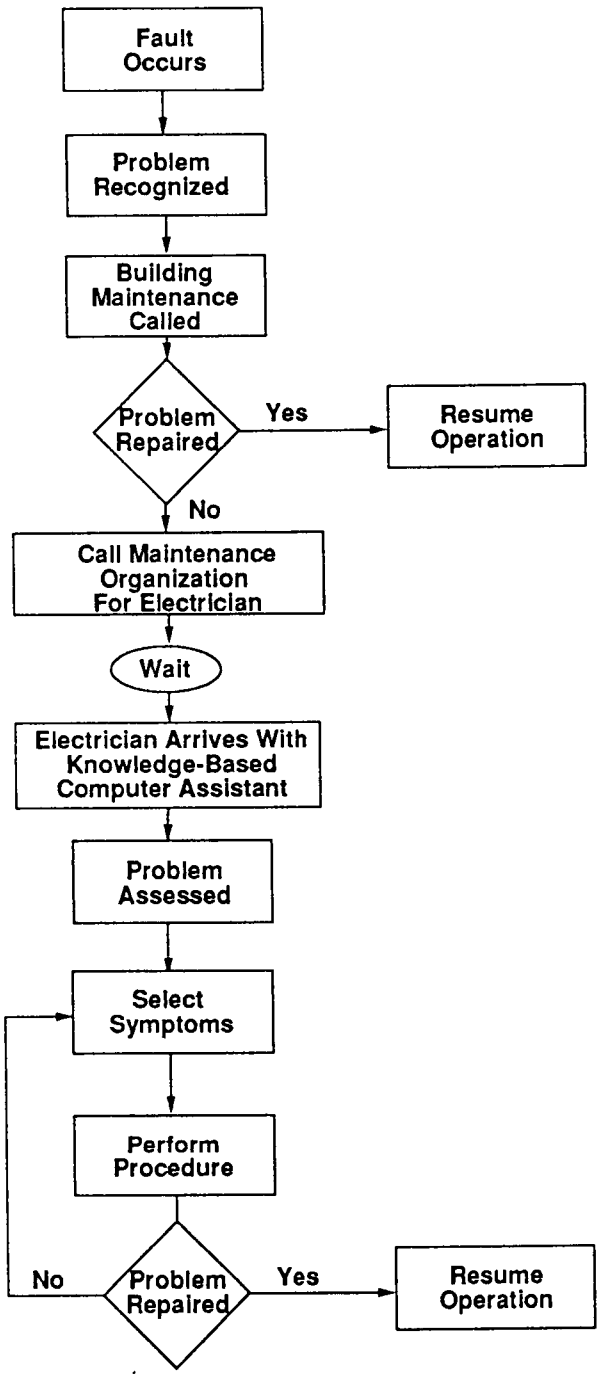


Figure 2 Improved Troubleshooting Work Flow Diagram

A transportable knowledge-based assistant was proposed as a means of implementing the desired solution.

1.3 Approach to Solution

Prior to my involvement in this project, a survey of similar systems was undertaken by Bill Zabriskie. Several were found to exist, particularly at Bendix/Allied Signal and at Texas Instruments. This indicated that a knowledge-based troubleshooting assistant was feasible. None of these systems were easily transportable, however.

A prototype troubleshooting assistant system which was not easily transportable was developed by Bill Zabriskie, Jim Mullens, and Laird Williams in 1988. This prototype used an IBM PS 2 Model 80 with two display monitors. One monitor was used for text and graphics. The second was a television monitor and displayed video images from an optical disk player/recorder. The system used PC-DOS (Personal Computer - Disk Operating System) and Smalltalk/V with an embedded frame-based expert system called ALEX. ALEX was found difficult to use and document, and was later abandoned in favor of a rule-based expert system described in section 4.2.4.

I became lead engineer for the project in October 1988. The prototype system with an initial application of a single subsystem of the industrial equipment was near completion. The hardware and software designs for the final system had not been done, and at that time it was still not clear that a single display, transportable system could be built.

With completion of the prototype system, Jim Mullens and Bill Zabriskie left the project. Shortly after this, Abbie Roberts and Jamie Jefferis joined Laird Williams and myself on the project team.

The primary features of the prototype which remain in the final system are that it is PC based and uses Smalltalk. The *HyperManual* module developed for the prototype was retained in the final system and the concept of a procedure module was retained although all code for it was rewritten.

Design of the final system was undertaken after survey of the current PC market via trade magazines and attendance at a trade show. Approximately 4 months was spent in learning Smalltalk and in determining the work breakdown structure for the project. Since this was my first experience as lead engineer on a project, the learning and use of common project management concepts and tools was also part of my task.

In the area of learning technologies, I attended a conference and exhibition sponsored by the Society for Applied Learning Technology to determine what, if any, commercially available products existed which could be used for this application. From this conference, I learned that no such integrated product existed although some research and development work in a similar area was being sponsored by the Electric Power Research Institute (Isle, Bloom and Butler 1989). However, this system required custom hardware and did not provide a single screen or color display.

1.4 Organization of Thesis

This thesis is divided into three primary areas. Background for the project is described in Chapter 2. This includes human factors and knowledge engineering for the system, as well as computer hardware and software. Chapter 3 describes the design considerations for the system as it relates to human factors, computer hardware and software, and knowledge acquisition. The actual implementation of the system hardware and software is described in Chapter 4. Included in this is a discussion of knowledge acquisition for both the synthetic and the algorithmic reasoning portions of the knowledge base. Chapter 5 is the summary and conclusions of the work to date with some discussion of system development decisions and conclusions, and future enhancements to the system which are planned.

2. BACKGROUND

A number of factors were considered in implementing a computer-based solution to the problem stated in section 1.1. In addition to the technical issues of computer hardware and software selection, the intended user and intended authors were considered. System response times (i.e., screen updates) must be sufficiently fast to prevent the user/author from waiting on the computer and consequently becoming bored or frustrated. Special consideration was given to the manner in which information is presented to the user and data are acquired from the user. Techniques for knowledge acquisition and incorporation into the system by the author were developed.

2.1 Human Factors

The intended users of the troubleshooting assistant system are hourly maintenance technicians. Although most will have some experience in the use of computers for equipment diagnostics, they will be inexperienced in the use of keyboards and other data entry devices. Historically, touch screens have often been used for data entry and screen control when the users are computer novices. This presents a problem, however, if large amounts of data must be input by the user. Touch screens also are prone to failure and, according to maintenance technicians, are difficult to repair. Another factor in the use of touch screens is the problem of screen coordinate calibration. Most touch screens are either resistive or infrared, with associated software to determine the location selected or "touched." This location determination is only as good as the initial "calibration" or coordinate designation for the touch screen. Coordinate designation can,

therefore, suffer from the parallax problem in which the user stands to one side causing the initial calibration to be distorted by the angle of touch.

In addition to the technical problems with the use of touch screens, users may be subconsciously biased against their use. This occurs because touching the screen requires touching glass and most people are taught as children not to leave fingerprints on glass. This initial reluctance can be overcome, but the use of touch screens in an industrial setting may also introduce technical problems with the accumulation of dirt and/or grease on the screen from the technician's fingers.

2.2 Computer Hardware

Prior to the development of the personal computer, sufficient computing power both in speed and storage capacity (memory and disk) necessary for the troubleshooting assistant was prohibitively expensive. The first personal computers reduced the expense greatly but lacked the computing power to perform the required tasks in a timely manner. Only with the development of Intel 80386-based personal computers with availability for 8 megabytes (Mbytes) of memory on the mother board and fast, high-capacity disk drives has the computer speed and storage capacity become available at a reasonable cost. Also necessary, but only recently available, was a full-color, high-resolution video display monitor. This type of monitor, combined with special video capture and display boards currently available, allows the presentation of screens containing both text and full-color photographic images.

2.3 Computer Software

Traditional, high-level programming languages such as Fortran, C and Basic are commonly used and are chosen for most applications due to the wide availability of personnel familiar with them. Within the area of artificial intelligence and expert systems, Lisp and Prolog are the most commonly used languages, with an increasing number of expert system shells (usually written in C) also available.

All of these languages mentioned are capable of providing some type of user interface, and an inference engine may be written in any of the languages. None of these languages, however, are particularly well suited for the primary task of delivering information to a user in an easily understood, consistent, and attractive manner. Object-oriented languages such as Smalltalk and C++ or Objective-C are well suited for user interface development (Rubin, Jones, et al., 1988).

Object-oriented languages, while markedly different in nature and semantics from traditional languages, in a somewhat abstract way model the manner in which humans behave. Object-oriented programming involves sending messages to an object to invoke a particular method known to that class of objects. Programming in this manner includes identifying the objects, classifying them according to similarities and differences, and designing the language of interaction among the objects (Goldberg 1984). These objects are simply related pieces of code and data which can communicate with other objects by sending and receiving messages. Each object belongs to a particular class of objects which may in turn be a subclass of another object. Individual objects are called instances of a class. Methods are used to define an object's behavior to the messages it receives and to send messages to other objects when appropriate (Digitalk 1988).

Methods are similar to subroutines in a traditional language. Objects are active rather than passive, however, and no longer exhibit the traditional separation of data and procedure.

All true object-oriented programming languages possess four characteristics: information hiding, data abstraction, dynamic binding, and inheritance (Pascoe 1986).

Information hiding means that the state of a module is visible only from within the module. Most modern languages, with the exception of standard Pascal, support some degree of information hiding.

Data abstraction is a way to use information hiding. An abstract data type is defined which consists of an internal representation plus a set of procedures to access and manipulate the data.

Dynamic binding means that each object is responsible for its own implementation. Each object can then be sent the same message. For example, an **Integer** will know how to print itself, as will a **String**, et cetera. This is known as polymorphism since the same message can elicit a different response depending on which class receives it.

Inheritance lets the developer specify differences in classes, not start completely over for each added module. Each object is a member (or instance) of a particular class of objects and has all the attributes of that class. That class may in turn be a child (or subclass) of another class of objects and will inherit all the attributes of that class as well. Many levels of inheritance are allowed in some object-oriented languages. For example, "Eva" is an instance of class "Electrical Engineer" which is a subclass of class "Engineer" which is itself a subclass of class "Technical Professionals," etc. "Eva" has all the attributes of each of the classes in the hierarchy of which she is a member, but her behavior is determined by the appropriate method at the lowest level (closest relation or

most specific) in the hierarchy. Some of the object-oriented languages also allow multiple inheritance so that an object may be a child of multiple, unrelated classes. Continuing with the example, "Eva" is also a member of class "Woman" (with the associated behaviors determined by "Woman" methods) which is a subclass of class "Human" (which has only two subclasses, "Woman" and "Man").

2.4 Knowledge Engineering

Knowledge engineering is the term applied to gathering all of the domain-specific information necessary to solve the problem of interest and allow the troubleshooting assistant to perform as required. Most expert system projects typically have the knowledge engineering shared among several people through a process called knowledge acquisition (Hart 1986). Typically, one or more domain experts exist who have the detailed knowledge necessary to solve the problem. These experts are a scarce resource, however, and often are unable to devote much time to the transfer of their knowledge to a computer system. Experts also seldom realize how much they really know about the subject and therefore tend to omit relevant information when describing how to solve the problem of interest (Kidd and Cooper 1985).

The knowledge engineer functions as both an interviewer of the domain expert and a programmer to transfer the knowledge to the computer system. The knowledge engineer must, in most cases, be capable of understanding the technical aspects of the problem so that appropriate questions can be asked of the domain experts. Additionally, the knowledge engineer must understand the computer software environment which is used to contain the knowledge. The manner in which the knowledge is represented by

the computer software must also be understood by the knowledge engineer since this can affect how questions are asked of the domain expert and, in some cases, even the knowledge which must be acquired (Waterman 1986).

3. DESIGN CONSIDERATIONS

The primary consideration for the design of the system was simplicity of use by the technician. Coupled with this was the need for ease of authoring and development for future applications and maintainability of both the hardware and software.

Human factors consideration was given for simplicity or ease of use and "comfortableness" to both the user and the developer. The computer hardware chosen had to be maintainable over the entire life of the system as well as being affordable. Computer software was chosen which was appropriate for the tasks to be performed. In addition to the basic software language and environment chosen, design of the individual software modules was also determined based on both user and author needs.

3.1 Human Factors

Design of the computer system was strongly influenced by a number of human factors considerations. Physical size of the system, ways of user and author interaction with the system including realism of the displayed information and levels of additional knowledge needed by both users and authors were all deemed to be important to the overall system design and, therefore, the ultimate success of the project.

The physical size of the system was considered since it was desired for the technician to transport the system to the factory floor for aid in troubleshooting at the actual work location. Union regulations require that any equipment weighing more than 45 pounds must be moved by riggers. Clearly then, each piece must weigh less than 45 pounds, or the system will no longer be transportable by the technician. It was

determined that the system should be as small and as light weight as possible while still being fully functional.

Since a number of different ways for interaction with both users and authors were available, each had to be considered. Information could be displayed to the user and/or author via text, drawings, and photographs. After discussions with maintenance engineers in the sponsoring organization, it was decided that the preferred method was the use of a limited amount of text with accompanying photographs. Some ability to annotate the photographs was also considered to be highly desirable. Photographs were chosen instead of drawings for two reasons. Although the initial cost for special video hardware was higher, a simple cost analysis determined that this would rapidly be offset by the much lower cost for generation of photographs as opposed to drawings. The cost of video hardware including camera was expected to be approximately $8500. Implementation costs were assumed to be approximately the same for either drawings or photographs, so this was not considered in the cost analysis. A relatively simple circuit board drawing (showing major components only) might take an experienced draftsman 8 hours to complete. A photograph of the same circuit board would take 15 minutes and require 2 people. This is a 16:1 reduction in cost. With a typical hourly rate of $50, this is a cost saving of $375 per photograph. As shown in the following equations, the additional cost of the video hardware would be recovered with the use of only 23 photographs!

Cost per drawing	8.0 * $50 = $400
Cost per photograph	0.5 * $50 = 25
Savings	$375

Breakeven point $8500 / $375 = 23

The cost is also reduced over the life of the system since re-shooting a photograph is typically less expensive than even minor changes to a drawing. Photographs are also much more realistic and show a much higher level of detail than is possible with drawings. With the ability to annotate the photographs, particular features can be highlighted while still preserving the overall integrity and realism of the photograph.

Information needed from the user is of two types: actual data which the system must act upon and control information for interaction of the computer with the user. Since technicians may be uncomfortable with the use of a keyboard, the consideration of ease of use required that the user do as little keyboard input as possible. An alternate user input device such as a mouse, trackball, light pen, or touch screen could be used for most user responses. From previous personal experience, I have found a trackball or mouse to be more durable than a light pen. The project sponsor had some experience with the use of a touch screen, and had found it difficult to maintain and repair. Both the light pen and touch screen were, therefore, eliminated from consideration based on past experience. A trackball or mouse can be physically located several feet from the display monitor; its use also eliminates the parallax problem sometimes associated with light pens or touch screens. The trackball was chosen since, for field use, it did not require a surface to "run over" as a mouse does.

For authoring, it was decided to use a keyboard for textual input and a mouse for control information and photograph annotation. We found that a mouse was preferable for authoring since it was more comfortable to use for the more finely detailed work associated with authoring. Although keyboard only input can be used, this is often awkward when using a software language which has been designed for use with a mouse

or other pointing device. Additionally, some type of video input device such as a video camera would be required for photograph generation or "shooting."

Discussions with the project sponsor indicated that multiple levels of user troubleshooting and repair ability should be supported. This would alleviate the boredom experienced by seasoned technicians when the information presented is too elementary, while still aiding the novice technician by providing more detailed information to the user when desired. Display of this additional information would be totally under the control of the user. In addition to descriptive text, we felt that some type of explanatory text should be available to the user when appropriate.

Authors, since they would ultimately be distinct from developers, should need only a superficial knowledge, if any, of the underlying language in which the software system was written. Additionally, as little recordkeeping and file management as possible should be required of the author.

3.2 Computer Hardware

The choice of computer hardware on which to base the system involved a number of considerations. The chosen platform was required to be transportable. It was also required to offer full-color displays with integrated video for photographic images. The chosen system must also have sufficient speed to prevent intolerable waiting by either the user or author. An intolerable wait can be defined as one in which the person wonders if the computer is really doing anything or if it would be faster to do the operation manually. Uebbing and Young stated that "If feedback to user-initiated events is produced in less than 300 milliseconds, it is perceived as occurring instantaneously"

(Uebbing and Young 1986). Since transitions from one step or display screen to another do not (and in some cases should not) appear instantaneous, a five second or less screen update time was considered acceptable. Sufficient disk storage capacity should be available for both current and future applications. The system should have current expansion capability for the addition of other features such as audio. This is desirable since the use of audio input would free the technicians' hands to perform repair work. Audio output would allow the technician to focus on the video image only and eliminate the need to read text on the screen. Modularity of hardware components was also considered important to allow upgrading as newer, better boards or components become available.

3.2.1 Computer Processing Unit and Bus

A high-speed processor was desired to prevent the user or author from "waiting" on the computer. Sufficient memory must also be available on the processor board to allow the programs to run at the full speed of the processor rather than at the often slower speed of add-on memory cards on the bus. In addition, industry standard, commercially available components were desirable to increase the ease of hardware maintenance over the life of the system. The bus on which the system operated was desired to be one for which a variety of add-on cards are available. This would reduce the risk which occurs when only a single source exists for a particular type of component or add-on board.

The transportability requirement for the system predicated the need that the processor and bus chosen be available in a transportable package. Expandability desires for the system were also a factor in choosing a bus architecture to be used since some

of the bus configurations considered had only one or two expansion slots available for user add-on cards.

Because video images are less commonly displayed with personal computer systems, it was also essential that the processor and bus be compatible with the video display boards necessary to display photographs.

3.2.2 Video Display and Storage

The video display needed to support the display of text, graphics, and photographic images in full color at a fairly high resolution. The use of higher resolution and full color makes the screen display more pleasing to the eye and, therefore, less tiresome to use. McCormick and Sanders have shown that higher resolution results in fewer misperceptions of information (McCormick and Sanders 1982). At the time of their writing however, the highest resolution available was less than that commonly available today.

High resolution and full color are both terms which must be defined since they often have different meanings in different contexts. The resolution of a video display is usually expressed in horizontal *x* vertical pixels. The greater the number of pixels, typically, the more pleasing the display. This is especially evident in the display of diagonal or curved lines. Full color for photographic images is typically represented by 24 bits per pixel, which provides 16,777,216 different colors. Video display adapters have differing resolutions and color capabilities. An IBM Color/Graphics Adapter (CGA) simultaneously displays 4 of 8 predefined colors with a resolution of 640 *x* 200 pixels. An Enhanced Graphics Adapter (EGA) for an IBM PC-AT can simultaneously display 16 colors from a palette of 64 colors with a resolution of 640 *x* 350 pixels. The

IBM Video Graphics Adapter (VGA) displays 256 colors simultaneously from a palette of 262,144 at 640 *x* 480 resolution.

Clearly then, the higher resolution and color capability of a VGA type display are desirable for text and graphics although EGA might be acceptable. Although the use of a VGA display is slightly more expensive, this additional cost is small in relation to overall project costs. If possible, however, the photographic images should be displayed with full 16,777,216 color capability. This could present a problem since most full color (16,777,216) capability is provided only by image capture and display boards which output an NTSC (National Television Standard Code) or RGB (red-green-blue) analog signal that can be displayed on a television type monitor with 512 *x* 512 resolution. The text and graphics generated by the computer and displayed through the video display adapter is typically output in a standard computer type signal for display on a TTL (transistor-transistor logic) signal level input type computer monitor or an analog type RGB noninterlaced computer monitor. Since it was desired to display both text and graphics, and photographic images on a single screen, a single display board or some method for combining the two signals was required. Additionally, the resulting signal must be capable of being displayed on a monitor small enough to be transported to the job site by the technician.

Storage of the photographic images must be done in such a way as to minimize the size and weight of any special storage devices needed. For this reason, an optical disk would not be a good choice since many optical disk players alone weigh almost 45 pounds.

An alternative to the use of an optical disk would be an image capture board which could store photographic images on a fixed disk or CD-ROM (Compact Disk -

Read Only Memory). If such a board were used, however, a large amount of disk storage space would be required for the images because a typical full-screen (512 *x* 512 pixels), full-color (24 bits/pixel) video photographic image requires 1.1-1.2 Mbytes of disk storage space. With this amount of space required, even a high-capacity (>300 Mbyte) disk drive would fill up very rapidly. Clearly, some method of reducing the amount of disk storage space consumed by each photographic image was highly desirable.

3.3 Computer Software

The first and foremost requirement for the computer software is that it be fully compatible with and appropriate for both the computer hardware used and the task which it is to perform. Consideration must also be given in design of the software to allow interaction with operating system commands and programs written in other languages.

Discussion of the computer software required can be divided into three primary functional areas: operating system, user interface, and expert system. These are presented in the following sections.

3.3.1 Operating System

The computer operating system is that part of the computer software which allows the developer and/or user to interact with the various hardware devices attached to the computer. Hardware devices include the display monitor, disk drive, and printer and may include remote devices attached via a local area network.

For the IBM PC-AT, PC-DOS is the most commonly used operating system, although others such as OS-2 (Operating System 2) and CP/M (Control Program/Monitor) are available. Most other software languages and hardware drivers are designed to run under DOS however. For this reason it was felt that DOS would be the operating system of choice for an IBM PC or compatible system.

Other personal computer systems such as the Apple Macintosh II have proprietary operating systems which are primarily transparent to the developer and user. Use of an operating system such as that of the Macintosh can present problems since development of applications must conform to the rigid specification of the operating system and is thought by some to be more difficult due to the relatively "closed" nature of the operating system.

3.3.2 User Interface

The design of the user interface software involved determination of general display screen layouts and the functions necessary to perform the desired tasks. Interaction with the user and author was desired to be as simple and require as little typing as possible. For this reason, an icon-based design was chosen for the display screens. With the use of icons and a pointing device (mouse, trackball, touch screen, etc.), most of the user and author interaction would be simplified.

Software for the user interface includes both the ability to display information as desired and the execution of the underlying functionality selected by the user or author. Another consideration is the ease of implementation for these functions by the developer. Although a particular display with its desired functions can be implemented in almost any language, certain types of languages are more appropriate for particular

applications. For example, although creating windows using FORTRAN is often cumbersome (unless the operating system has an inherent or layered windowing system), it may be appropriate for an application which involves large amounts of numerical computation.

With the use of an icon-based user interface, an object-oriented programming language was considered to be especially appropriate (Cox and Hunt 1986). In the object-oriented approach, each display screen is treated as an instance of a particular object and methods are used to implement the functionality of each icon. (See Section 2.3 for an explanation of this terminology.) It is helpful if the chosen object-oriented language includes support for development of windows for screen display. This is especially simplified if the chosen object-oriented language includes classes for development of icons and already supports the ability to respond to icon selection.

In determining general screen layout, both the user and author were considered. The primary authors are ultimately intended to be maintenance engineers who have some familiarity with computers for electronic mail, word processing, diagnostic assistance, et cetera, but are not programmers. To eliminate the need for these authors to learn to program in another (perhaps cryptic) language, it was felt that the authoring interface as well as the user interface should be icon-based. In keeping with this and the WYSIWYG (What You See Is What You Get) concept, the authoring interface should provide a display screen which closely resembles the user screen.

A very limited number of functions are required for the user, so screen size limitations did not preclude the use of large icons which can be used with a variety of pointing devices. (A touch screen often requires larger icons since fingers are fatter than an arrow directed by a mouse.) For the authors, however, a much greater number of

functions and their corresponding icons was needed. This would require the design of much smaller icons and the use of a much more precise pointing device such as a mouse, trackball, or light pen.

As described in Section 3.2.2, design of the user interface included the required ability to display text, graphics, and photographic images on a single color display. Additionally, it was desired to integrate the shooting of photographic images with the rest of the authoring process, if possible.

3.3.3 Expert System

The expert system refers to the computer program with an inference engine which facilitates synthetic reasoning thereby solving problems in the domain of interest. The expert system may be rule-based and act on typical "if-then" rules or may use frames or objects which contain multiple interactive pieces of information as the knowledge representation scheme. The use of frames or objects is often more powerful since each frame or object also includes multiple procedures describing the relationship between the parts. Rule-based systems may be less powerful and can be cumbersome for complex relationships but are appropriate for diagnosis and troubleshooting where the domain knowledge results from empirical associations developed through experience (Waterman 1986).

For this application, synthetic reasoning is only required to choose between general symptoms relating to subsystems of the industrial equipment (more about this in section 4.3). A rule-based expert system was considered appropriate in this case due to the relative simplicity of the synthetic reasoning required and the fact that rules are often easier for authors to understand and express textually. Rule-based expert systems

typically operate in either forward-chaining or backward-chaining mode. Forward-chaining is data driven and compares the data or facts to the rules to determine if goals are met. Backward-chaining is goal driven and seeks to establish a particular goal using stated or inferred facts. Backward-chaining works well when only a particular goal is sought, but forward-chaining is more appropriate when any goal will do or initial facts are known (Luger and Stubblefield 1989).

3.4 Knowledge Acquisition

Knowledge acquisition is often a time-consuming task. Manufacturers' manuals may need to be converted into a computer readable form although, with the advent of modern word processing, many vendors can now supply equipment manuals in a machine readable format on magnetic tape or floppy diskettes.

Many expert system projects are undertaken in order to capture the knowledge of human experts in a particular field. This was not the case for this project and in fact is contrary to popular opinion of requirements for a successful expert system project. The particular industrial equipment which would be used for the initial application was only recently installed, and no human experts existed for diagnosis and repair of equipment faults. The experts did not exist, therefore, not because it was not possible to become an expert in the task at hand but because no one had yet sufficiently considered the problem to become an expert in solving it. A project such as this can also be a good candidate for an expert system application (Porat and Rhyne 1990). Knowledge acquisition for this project, then, involved the elicitation of any knowledge about the industrial equipment from humans and incorporating this with general troubleshooting

techniques. The knowledge engineer, by the end of the project, clearly would be an expert in diagnosis and repair of the equipment and would have transferred this knowledge into the computer.

The acquired knowledge can be stored in a computer in a variety of forms. Text files may be the preferred method for storage and display of the information found in manufacturers' manuals. Procedure guide information may be converted to rules for an expert system or used directly as procedural algorithms. Knowledge gained from humans may also be used to develop procedures and/or expert system rules. Drawings and photographs are often presented as they are but may need to have areas emphasized and/or highlighted. Special methods, media, or add-on boards may be required for the capture, storage, and display of the drawings and photographs.

3.5 Design Summary

The final system hardware design was chosen as an IBM PC compatible transportable computer with special video hardware to capture and display photographic images. A keyboard and trackball or mouse would be used as input devices. A high-density fixed disk drive would be used as the storage device. For video display, an external VGA-type color monitor small enough to easily carry would be used.

Software for the chosen system included PC-DOS as the operating system and Smalltalk V/286 as the object-oriented programming language for software development. The expert system would be rule based and embedded within the Smalltalk environment. For hardware interaction, C would be used when needed.

Knowledge acquisition would be accomplished using photographs, manuals and interviews as appropriate.

A detailed description of the system implementation is given in chapter 4.

4. SYSTEM IMPLEMENTATION

Development and implementation of the system can be divided into two areas: the computer system and software required to achieve the desired functionality and the application- (or domain-) specific knowledge necessary to assist in the actual diagnosis and repair of industrial equipment malfunctions. The former of these is described in the following sections on hardware and software. The final section of this chapter describes the knowledge acquisition necessary for a specific application of the system.

4.1 Hardware

Hardware for the system includes the personal computer, video display, and user input devices. This hardware is shown in figure 3 and described in the following sections.

4.1.1 Personal Computer

Since it was required to use a relatively high-speed, transportable personal computer for which a wide variety of add-on boards are available, an IBM PC-AT compatible 80386 computer operating at 20MHz with 8 Mbytes of RAM (random access memory) on the system board was chosen as the hardware platform. This personal computer operates with an IBM AT bus at 8 MHz, so most add-on boards designed for the IBM PC-AT will function properly in it. The computer also includes both serial and parallel ports for attachment of a pointing device and/or printer. This computer is



Figure 3 Transportable Troubleshooting Assistant Hardware

packaged in a COMPAQ-style luggable case with the monochrome monitor removed to reduce the weight.

An ESDI (Enhanced Small Device Interface) 380-Mbyte, full-height internal hard disk drive with self-parking heads is used to meet the requirement for large amounts of disk storage. The disk drive was specified to have self-parking heads so shutdown of the computer can be accomplished simply by turning power off with no risk to the disk drive. This was especially important since the system is designed to be moved frequently.

To facilitate software interchange and backups, several peripheral device options are included: a 3.5-inch internal floppy disk drive, a 44-Mbyte external Bernoulli cartridge drive and controller, and a pocket Ethernet adapter which attaches to the parallel port rather than requiring an internal Ethernet adapter board. The Ethernet adapter is used to connect the computer to a local area network for backups and data exchange with other systems.

4.1.2 Video Display

Video display hardware for the system includes an add-on board for photographic image capture, compression, and display, as well as a video mixer board for display of combined computer and television-type video signals and the actual display monitor which the system uses. The signal relationships among these components are shown in figure 4.

The video camera used for the still-frame video photographs may be either a still video type or a full-motion video camera with integral tape transport. The ability to record images on video tape is neither needed nor utilized for this project. The camera

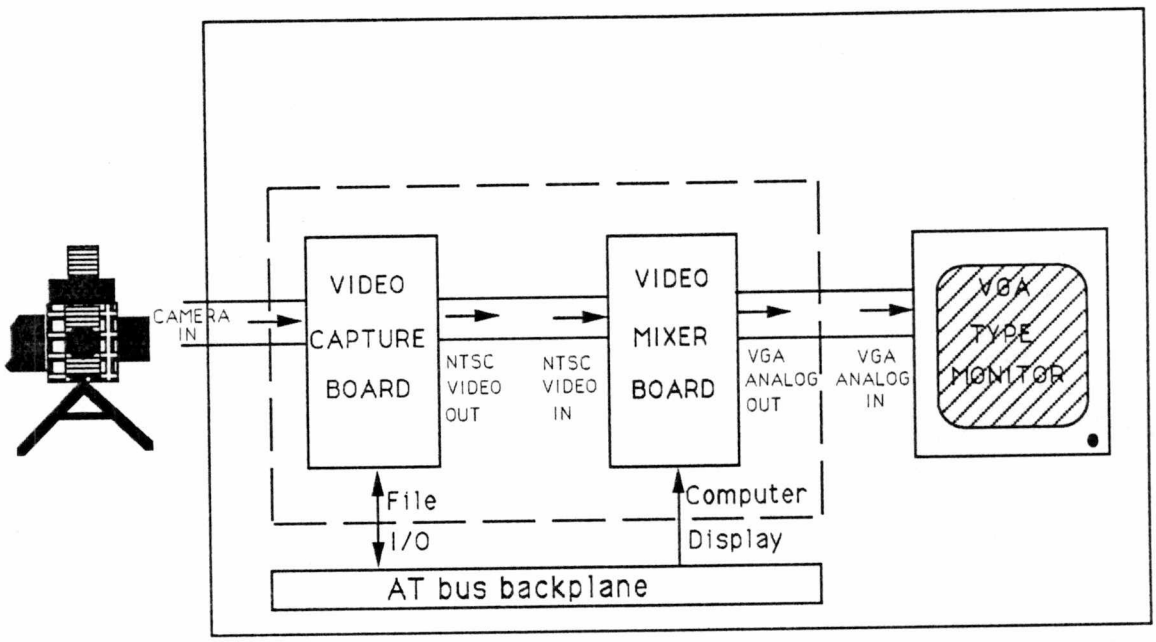


Figure 4 Video Display Component and Signal Relationships

used must have a video-out signal available. This is an NTSC composite video signal with sync included; the connection to the camera is typically via a single, RCA-type phono jack. A consumer grade camera is adequate for most of the photographs required, but lighting is critical, and the use of an external light source was found to greatly increase the color quality and image clarity. A tripod is also helpful since it usually results in fewer retakes and reduces photographer fatigue.

The video image capture, compression, and display board is a CVP 2800 from AudioDigital Imaging, Inc. (ADI) (formerly Discrete TimeSystems, Inc.). This board allows photographic images to be stored as files on a disk, eliminating the need for special video storage media such as a video disk recorder/player. The compression algorithms are proprietary to ADI, and compression is performed in the hardware and firmware on the board, rather than in software. This approach greatly reduces the load

on the computer's central processing unit and speeds the response time of the system when changing photographs. Compression ratios from 2:1 to 30:1 are available using a library of C language routines to interact with the board (Discrete TimeSystems 1988). Images may also be scaled to fit within a smaller window of the display screen. For this application, a few nearly full-screen images with 2:1 compression are used. Most of the photographic images used are scaled to approximately 1/2 the screen area and a 10:1 compression is used. Scaling and compression of the video images significantly reduces the storage space required for each image. For the majority of the photographic images (those which cover 1/2 the screen area), the disk file size is reduced from 1.1 to 1.2 Mbytes per image (if full-screen, full-color) to 60 to 90 kilobytes (Kbytes) per image. All of the photographic images are still full color with 16,777,216 colors available.

Output from the CVP 2800 video capture, compression, and display board is a standard NTSC composite video signal accessible via an RCA-type phono jack. As shown in figure 4, a single cable connects this video-out signal to a similar video input jack on a Matrox VGO-AT/NI video mixer board. According to the manufacturer of the CVP 2800 board, no other application exists where two boards such as these are used in the configuration shown in figure 4.

The VGO-AT/NI board functions as the display adapter for the system. It offers VGA resolution of 640 x 480 pixels but is capable of displaying only EGA color (see Section 3.2.2). One of the 16 EGA colors can be defined (by vendor supplied programs) as "clear." Once this is done, the video image from the video input jack of this board is displayed at any location where the "clear" color occurs on the display screen. All other colors are shown just as they would appear with a standard video adapter. This allows a window of the screen to be defined as "clear" for some functions

and contain computer-generated screen information for other functions. This arrangement of the two video boards means that the photographic video image is always being output from the CVP 2800 to the VGO-AT/NI, but the image may not be visible to the viewer if there is no window of the "clear" color defined. For this application, brown was chosen as the "clear" color. It was chosen because it is seldom used for color display screen features and/or icons. This is true because brown often lacks good contrast with other colors which might be used adjacent to it (as in a text and background situation).

Output from the VGO-AT/NI board is an analog RGB noninterlaced signal. The use of the noninterlaced signal reduces the flicker often seen on video displays. This output signal is supplied via a 15-pin D-type connector using standard VGA pin-outs. As shown in figure 4, this signal is connected via a VGA monitor cable to an analog RGB monitor. For this application, an external IBM 8513 12-inch video monitor is used. It was chosen due to the smaller size while still maintaining VGA resolution. No commercially available color VGA monitor small enough to fit in the COMPAQ-style transportable case existed at the time of hardware acquisition. The requirement for a color display, therefore, necessitated the use of an external monitor.

4.1.3 User Input Devices

A pointing device was desired to facilitate user input. For field use, a trackball called a RollerMouse from CH Products was chosen. This trackball fully emulates a Microsoft mouse (it will actually operate using the Microsoft mouse driver) which has become a de facto standard for IBM PCs. As explained in section 3.1, the trackball was chosen to eliminate the need for a surface to run over which a mouse requires. This

particular trackball has 4 buttons; the lower left and right buttons function exactly as the left and right Microsoft mouse buttons, respectively. The upper left and right buttons behave as toggles. Depressing either one acts as though that lower button is held down; depressing the same button again acts as a release on the lower button. This is especially handy for cut and paste or scrolling operations in a windowed environment which supports mouse input.

For authoring, either a Microsoft mouse or the RollerMouse may be used as the pointing device. They are completely interchangeable. The keyboard supplied with the transportable is used for textual input by both users and authors, although very little textual input is required from users.

A bar code reader, which consists of an add-on card for the PC and a wand, connects between the keyboard and the keyboard input jack of the PC. This bar code reader can be used in place of keyboard input for any information available in bar code form.

4.2 Software

Software for the system can be divided into two components: that necessary for formation and presentation of the troubleshooting and repair applications and the actual troubleshooting and repair applications knowledge base. The latter of these along with the hybrid expert system design approach taken is described in the succeeding section (4.3) on knowledge acquisition. The actual software developed for implementation of the application is described in this section.

4.2.1 System Software

For the primary software development environment, Smalltalk V/286 from Digitalk was chosen. Smalltalk is an object-oriented programming system which has rich support for user interfaces with windows and icons. Smalltalk runs on an IBM PC-AT or compatible (it requires an Intel 80286 or 80386 processor) with DOS version 3.3 or greater as the operating system. Additionally, C language programs are used for some of the video hardware interactions. Figure 5 shows the relationship among the operating system, Smalltalk, and C programs.

Smalltalk runs in protected mode on the PC and, consequently, is able to use extended memory. This is the reason for the large memory requirements for the system.

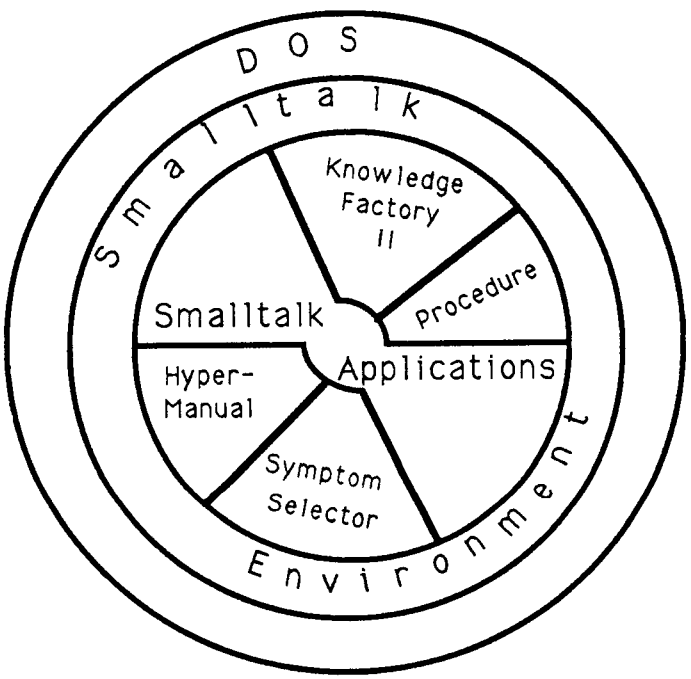


Figure 5 Major System Software Component Module Relationships

When Smalltalk is started, a portion of memory, beginning in the first 640 Kbytes and continuing into extended memory, is reserved for DOS commands. When a DOS command is initiated, Smalltalk temporarily switches to real mode to execute the command and then returns to protected mode for further Smalltalk implementation.

Smalltalk is a compiled language but differs from typical compiled languages since it is also the environment in which it operates. This means that within the Smalltalk environment, the developer has access to all of the code for all of the classes and methods which are implemented. This code resides in the **image** and must be saved at the end of each development session. In addition to the executable file which runs Smalltalk, three files are required for Smalltalk operation. These files are: **sources.sml**, which contains the source code for Smalltalk classes and methods; **change.log**, which is an ongoing record of all changes made to classes and methods since the **sources.sml** file was updated; and the **image** file, where pointers to the **sources.sml** and **change.log** files reside. The **change.log** may be periodically compressed to remove unused or duplicate code so only the last version of each class or method is included. In a similar manner, the **sources.sml** file may be compressed to incorporate all of the code from the **change.log** file into the sources.

Care must be taken when using Smalltalk that the **image** does not become corrupted. Recovery is possible if this happens, however, since classes and methods may be grouped together in 'projects' and then saved to an external file, or the individual classes (or only the changes to them) may be saved to external files. These external files can then be reinstated in the **image** by a "fileIn" process.

4.2.2 User Interface

The user interface for the system is actually not a single user interface but a collections of interfaces. This collection of interfaces is complete at this time (with the exception of future enhancements) and is used for presenting the knowledge base information to the technician. All of the user functions are shown in figure 6 and each is described below.

The *User and Site Selector* module displays the initial screen seen by the user. This screen is shown in figure 7. As shown, three functions are available from this screen. The **START** function initiates operation of the troubleshooting assistant system, in which case, the user is asked to enter his (her) badge number (via either keyboard or bar code reader from badge). The badge number is compared to a list of authorized users and authors. If the person is an approved author, he (she) may choose to author or run. The badge number is also used to record the identity of the user in a result log file for the troubleshooting and repair session. All subsequent information entered into

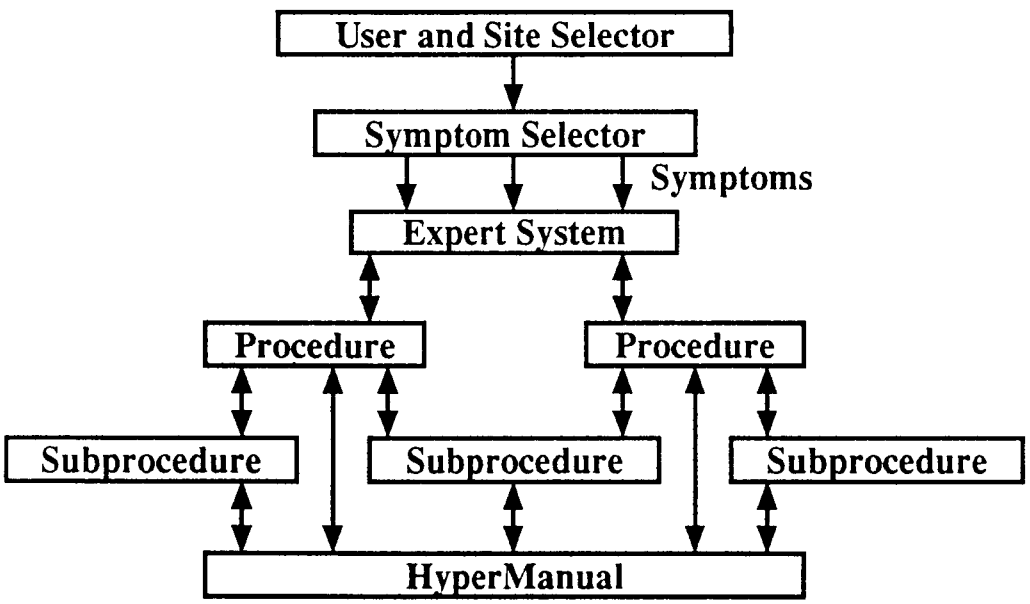


Figure 6 User Module Functional Relationships

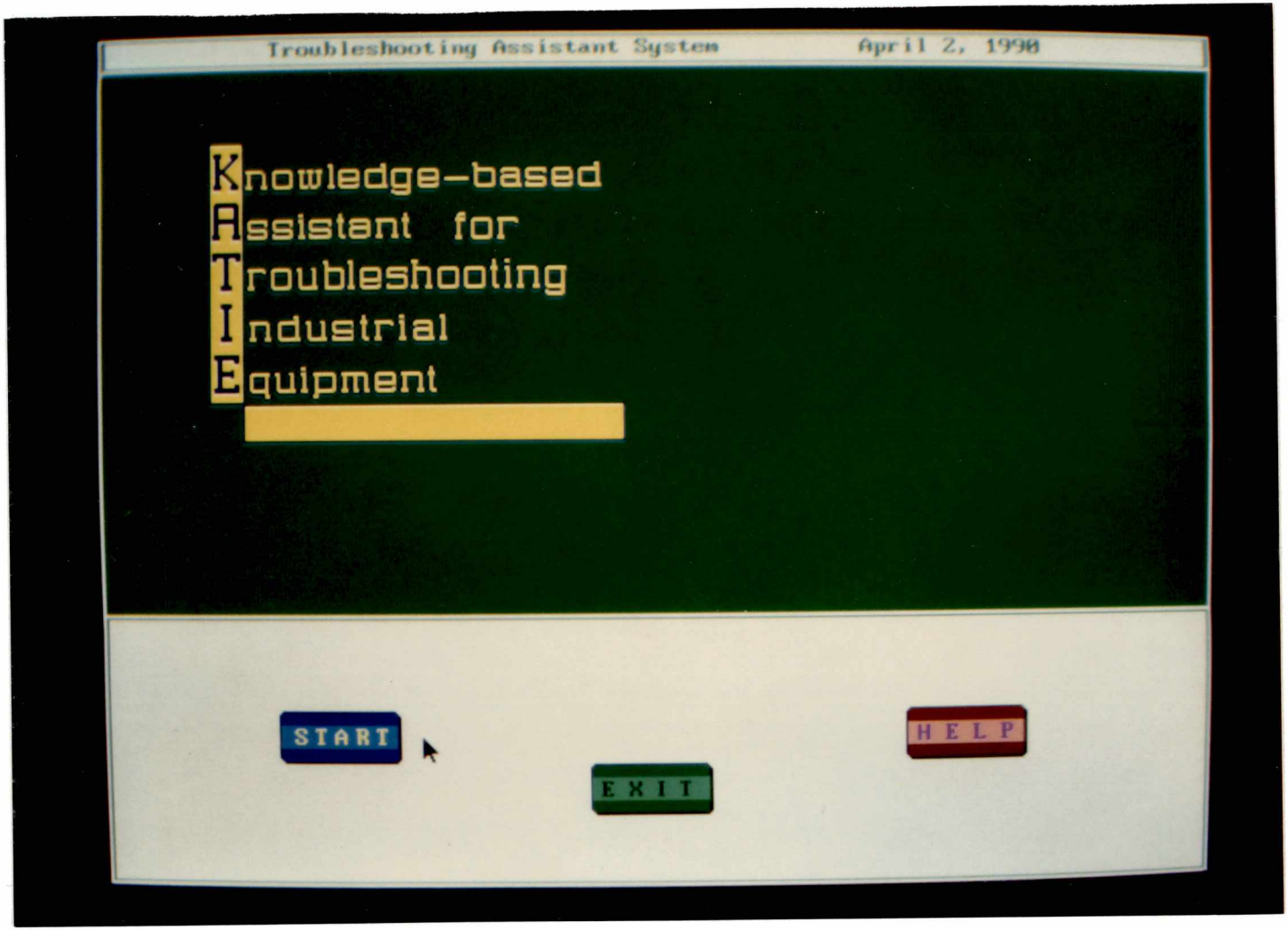


Figure 7 *User and Site Selector* User Module Interface Screen

the system by the user is also recorded in this file. The user is then asked to choose which of the industrial systems is of interest at this time. This response is recorded and control is transferred to the *Symptom Selector* module described below. The **EXIT** function exits from the troubleshooting assistant to DOS for users or to the Smalltalk environment for authors. The **HELP** function gives rudimentary information for getting started with the system.Initial information for troubleshooting is entered into the system via the *Symptom Selector* user interface shown in figure 8. This screen also has three functions selectable by icons. Additionally, the photographic portion of the screen has active areas which are used to select portions (or subsystems) of the industrial equipment which seem to be faulty. When a particular active area is selected, a symptom may be entered into a list of symptoms, or the user may be asked for additional information via menus before another symptom is added to the list. Each of these symptoms is displayed textually in the small window near the bottom of the screen. The active areas act as toggles for symptom selection. Initial selection enters the symptom into the list, and subsequent selection will remove it from the list. The **VIEW** icon allows the user to select other views of the industrial equipment (e.g., rear, doors open). The **GO** icon records the symptoms in the result log file and transfers the list of symptoms and control of the system to the *Expert System* module inference engine described in section 4.2.4. **EXIT** returns control to the *User and Site Selector* module. Operation of the *Expert System* module causes text to scroll in a temporary window on the screen. When the inference engine arrives at a goal which is capable of opening the *Procedure* module, this temporary window disappears. All rules confirmed by the *Expert System* module are also recorded in the result file. Control is then passed to the *Procedure* module for the application which has been selected by the expert system.

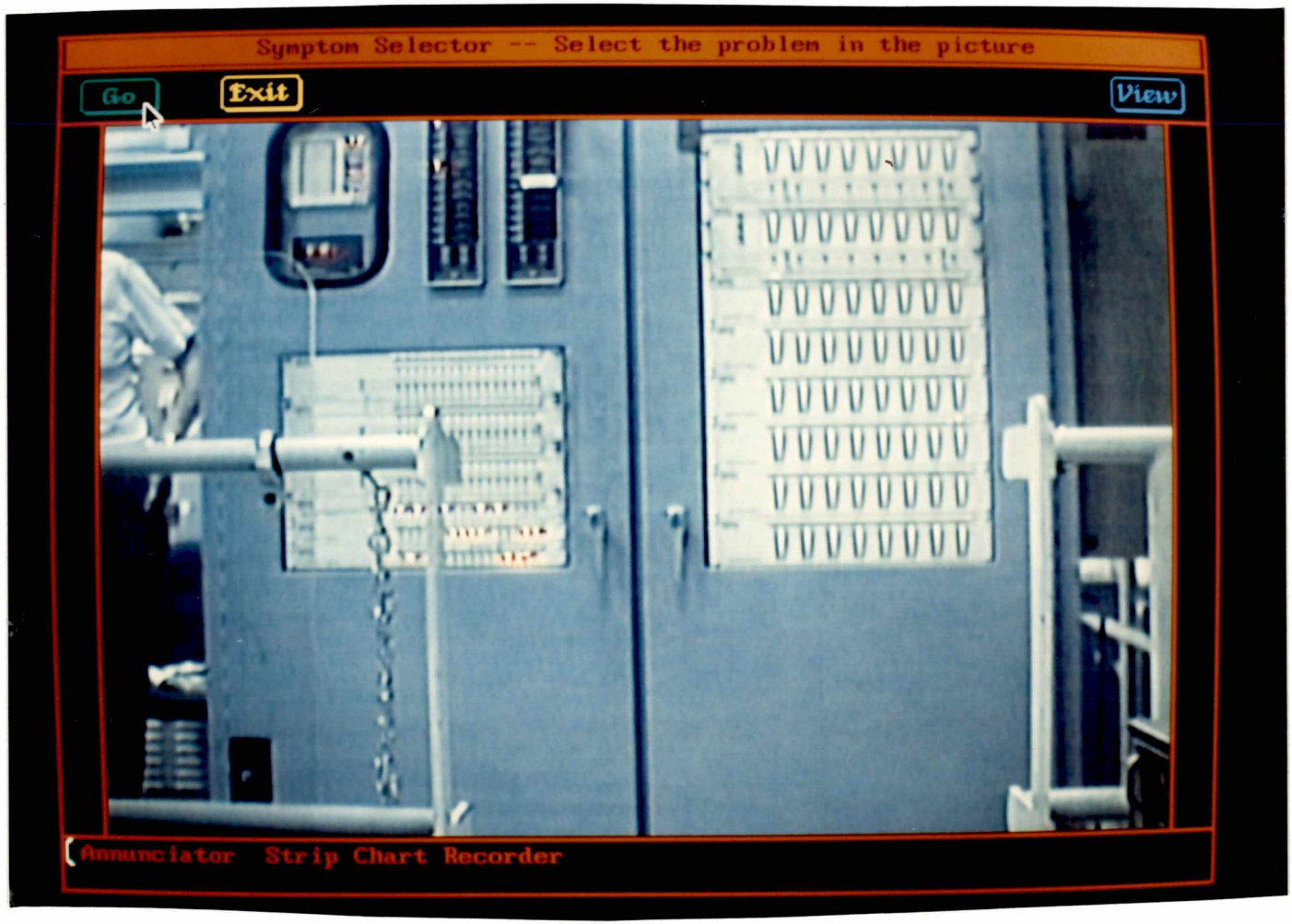


Figure 8 *Symptom Selector* User Module Interface Screen

The majority of the user's time is spent in interaction with *Procedure* modules. The application content for these modules is described in section 4.3.2. The function of the *Procedure* modules is to display to the user procedural information for diagnosis and repair of the industrial equipment subsystems. The user interface shown in figure 9 is identical in its functional interaction with the user for both procedures and sub-procedures as shown in figure 5 (p. 36). This interface window has three separate functional areas: a photographic image pane, a text display pane, and a control pane. The photographic image pane is used to display a video picture appropriate for the text displayed in the text display pane. The control pane allows the user to perform step control, detail, manual, and text functions. Step control includes **EXIT**, **RESTART**, < (go back a step), > (go to the next step), and **DETAIL**, which invokes an appropriate subprocedure. Use of the subprocedure is totally at the discretion of the technician. **TEXT** and **WHY** cause the display of descriptive or explanatory text, respectively, in the text display pane. **MANUAL** transfers control to the *HyperManual* module at an appropriate manual location. All icons displayed to the user are dynamic in the sense that they are displayed only if they truly have a function at that step (i.e. **RESTART** and < do not appear for step 1). The **EXIT** function is only available at the first and last steps, but the user may **RESTART** to the first step at any time. When the user chooses to advance to the next step (>) or **EXIT**, any Smalltalk code associated with that step is executed. This code is used to ask questions of the user, record user responses to the result log file, and branch to other steps. In this way, the user may not proceed consecutively through the steps. Instead, depending on how the questions are answered, a number of paths through a particular procedure may exist. Smalltalk code may also be used to directly access the *HyperManual* module, add or

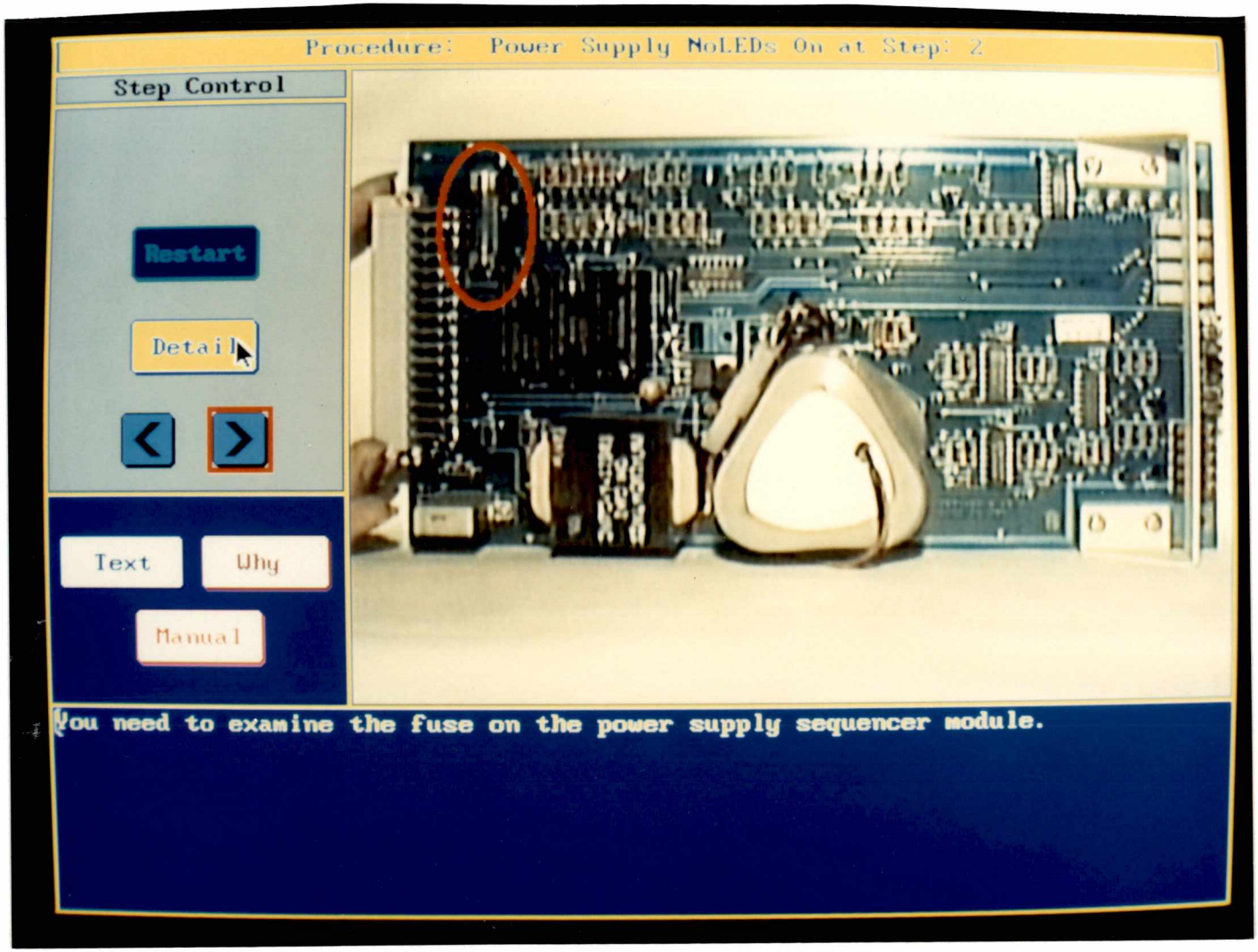


Figure 9 *Procedure* User Module Interface Screen

remove symptoms from the symptom list of the *Symptom Selector* module, or exit the *Procedure* module. When exiting a *Procedure* module, either via code or the **EXIT** icon, control is returned to the module which initiated that procedure. If the user is currently executing a subprocedure, then control is returned to the calling procedure at the step from which the subprocedure was launched. If the user is exiting from the initial procedure, control is returned to the *Symptom Selector* module.

The *HyperManual* module provides direct, on-line access to manufacturers' manuals and other strictly textual information. From any point in the manual, any other portion of the manual is accessible simply by selecting the appropriate reference with the mouse. The *HyperManual* module offers the user functions similar to those available when accessing a paper manual. Access to other sections of the manual can be achieved from the *HyperManual* screen (shown in figure 10) using the **MANUAL LOCATION**, **CROSS REFERENCES**, **|<<** (backup to beginning), **>>|** (go forward to end), **<** (backup to previous page), **>** (go to next page), and **GOTO MARK** selections. **SET MARK** and **REMOVE MARK** are used to set and remove bookmarks, respectively. **GOTO MARK** is then used to return directly to a manual page for which a book mark has been set. Each book mark is named, with the default name being the same as the manual section name. **MANUAL REFERENCE** gives an information only message of the location hierarchy for the particular manual section which is currently displayed. The **EXIT** selection returns control to the *Procedure* module and step from which the *HyperManual* module was accessed.

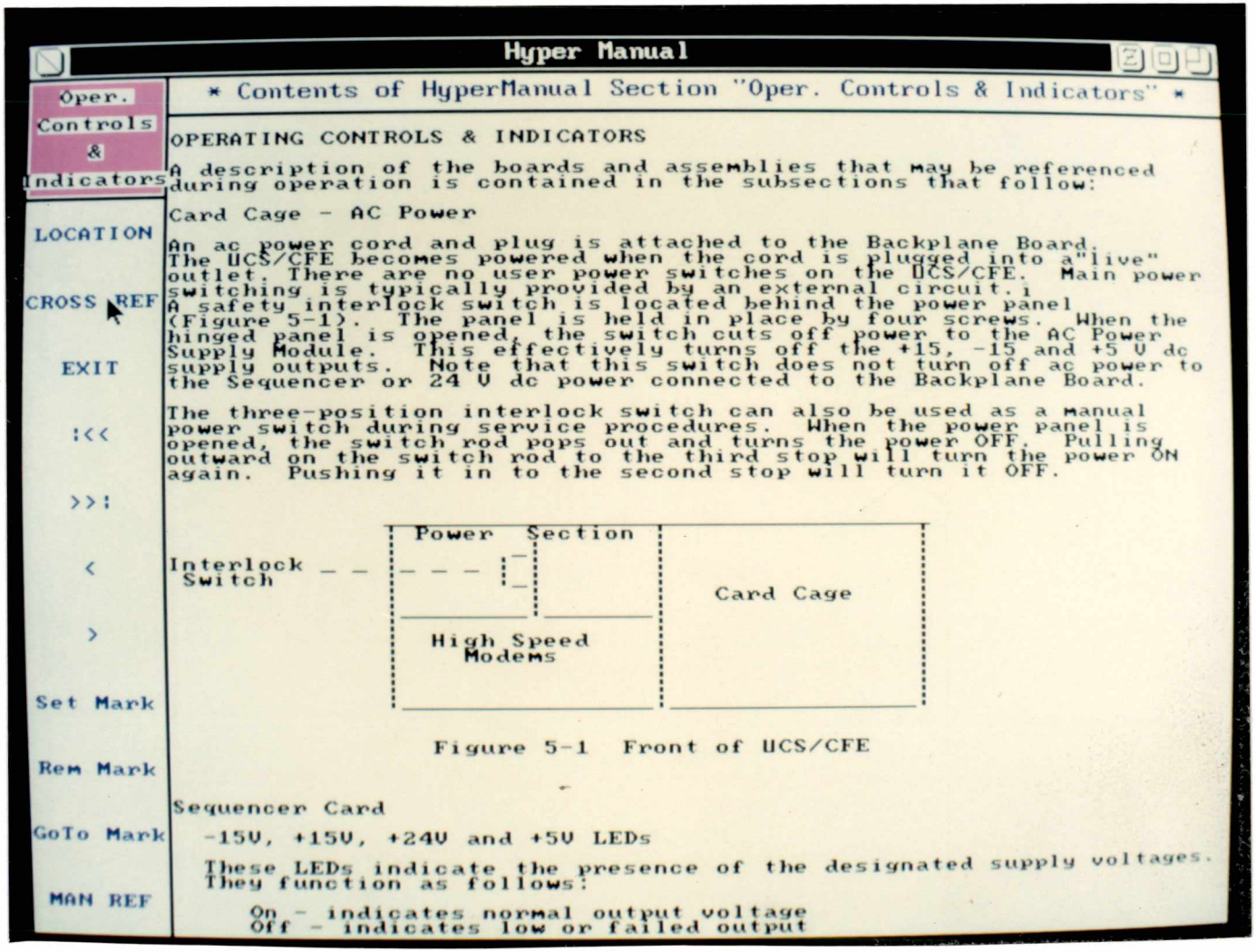


Figure 10 *HyperManual* User Module Interface Screen

4.2.3 Authoring Interface

The authoring system interfaces are used for transferring the knowledge base into the computer. Presently, two of the anticipated five authoring system modules have been completed. These are shown in figure 11. The *HyperManual Browser* and *Procedure Authoring System* have been completed. The *Industrial Equipment Browser, Symptom Selection Editor,* and *Rule Editor* have been designed and are presently in the process of having the Smalltalk code written to implement them. The interfaces to these modules will be icons as with all other interfaces in the system. Lack of completion for some of the authoring system modules has only minimally impacted the project because their functionality is needed for nonprogrammers to use the system for authoring (this is the ultimate intention) but was not required for the initial

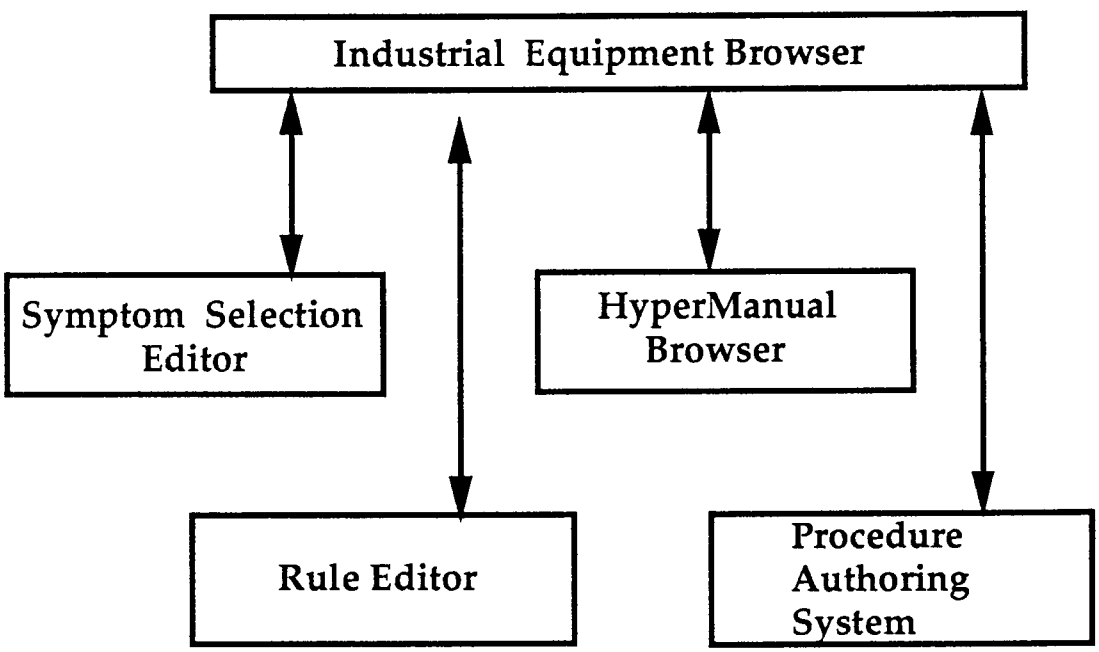


Figure 11 Authoring System Module Relationships

application. This was the case because the initial application was authored by the project team which developed the entire user and authoring system.

The term "Browser" is a commonly used Smalltalk term and indicates a window used to display and/or edit information which is functionally related. The *Industrial Equipment Browser* will be used at the top level to create and modify new applications for the entire system. The interface to this module will be used to access all of the other authoring modules as indicated in figure 11.

The *Symptom Selection Editor* has three functions: capture and display of full-screen photographs of the industrial equipment, designation of active areas on the screen for each view of the equipment used, and designation and association of the symptoms which may be selected by the user.

Creation, editing, and maintenance of the knowledge base used for synthetic reasoning is the function of the *Rule Editor* module. Prior to implementation of this module, this portion of the knowledge base for the initial application was manually created with a text editor and then added to the system **image** by highlighting the external file which contains this text and then choosing "do it" from the Smalltalk main menu. With implementation of the *Rule Editor* module, it is planned to continue maintaining this knowledge base in an external file, but updating the system with this information will be as simple as choosing an icon from a control pane.

The *HyperManual Browser* is used to define the hierarchy of the manual which is used for the application. The interface to this module is shown in figure 12. The hierarchy is a tree structure similar to a table of contents for a book. Each branch can be either a node or a leaf. A node points to another node or to a leaf. The leaves point to the actual text files which contain the manual sections. Cross-references can be

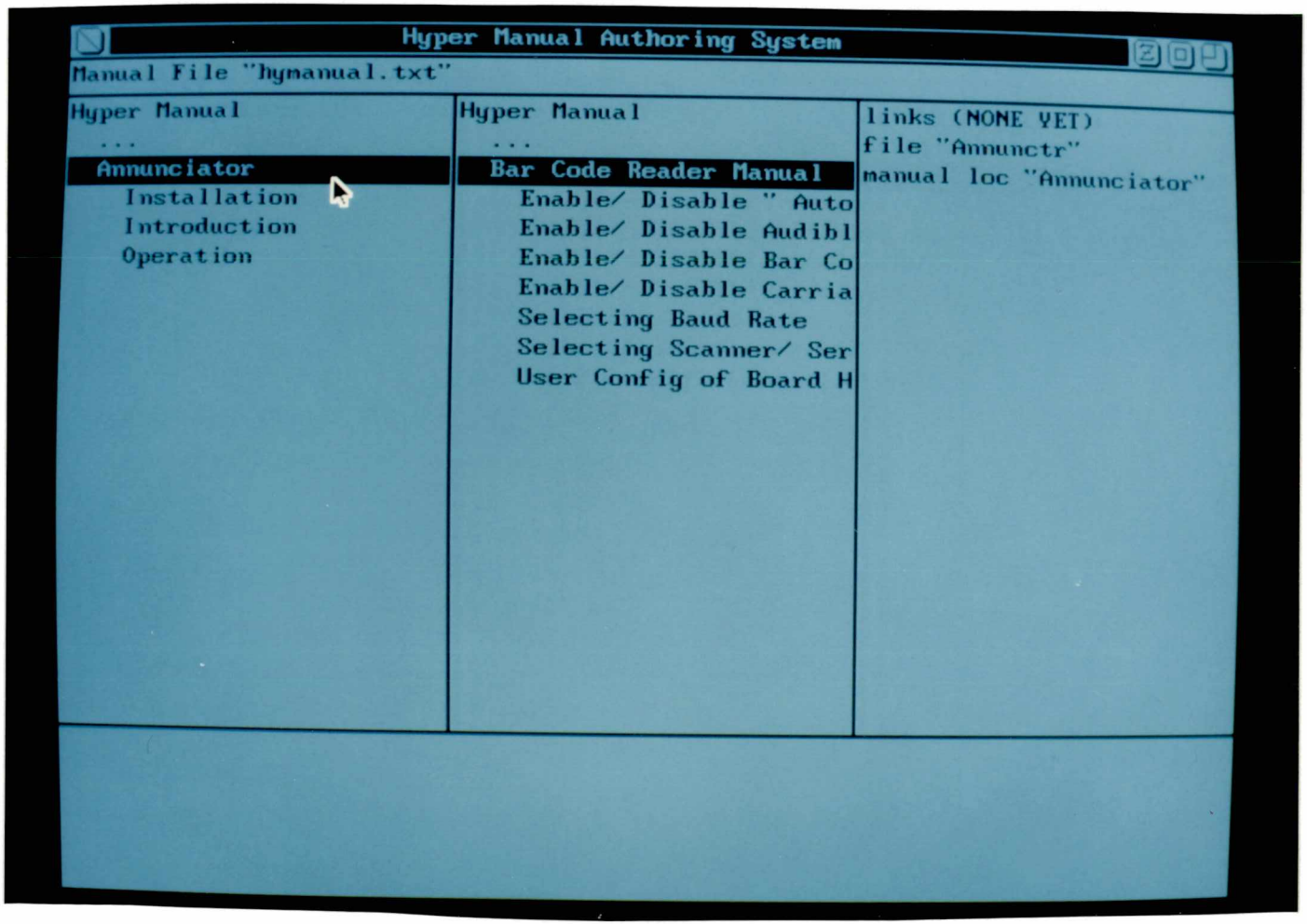


Figure 12 *HyperManual Browser* Authoring Module Interface Screen

defined for each node or leaf. These cross-references are manually entered by the author, and currently there is no plan to automate this (unfortunately).

The *Procedure Authoring System* is the most complex of the authoring interface modules and actually consists of four related screens. Use of the *Procedure Authoring System* is presently accessed from the *User and Site Selector* module of the user interface system. The initial screen is the *SubSystem Directory Editor* shown in figure 13. Each of these systems is maintained in a separate file. The term subsystem is used because typically these systems represent the functional subsystems of the industrial equipment. No index or list of these subsystems exists within the Smalltalk **image**. Instead, the system lists all of the subsystems contained on the disks in files which have the correct file extension ('.s02') and are of the correct internal format. This was done to simplify recordkeeping and file management. The system automatically determines a location for any new subsystem and its procedures when it is created. Most of the icons depicted perform various functions on these system files. The **LOAD** icon transfers control to the *Procedure System Editor* with the selected system as the one to be viewed. The **EXIT** icon returns control to the module which initiated the authoring.

The *Procedure System Editor* is used to create and maintain a group of functionally related procedures. The interface for this is shown in figure 14. All of the functions operate directly on the system file except **EXIT**, **RUN**, and **EDIT**. **EXIT** returns control to the *SubSystem Directory Editor* after querying the author concerning saving the system file. **RUN** transfers control to a *Procedure* module (described in section 4.2.2) operating on the selected procedure. Similarly, **EDIT** transfers control to a *Procedure Editor* module operating on the selected procedure.

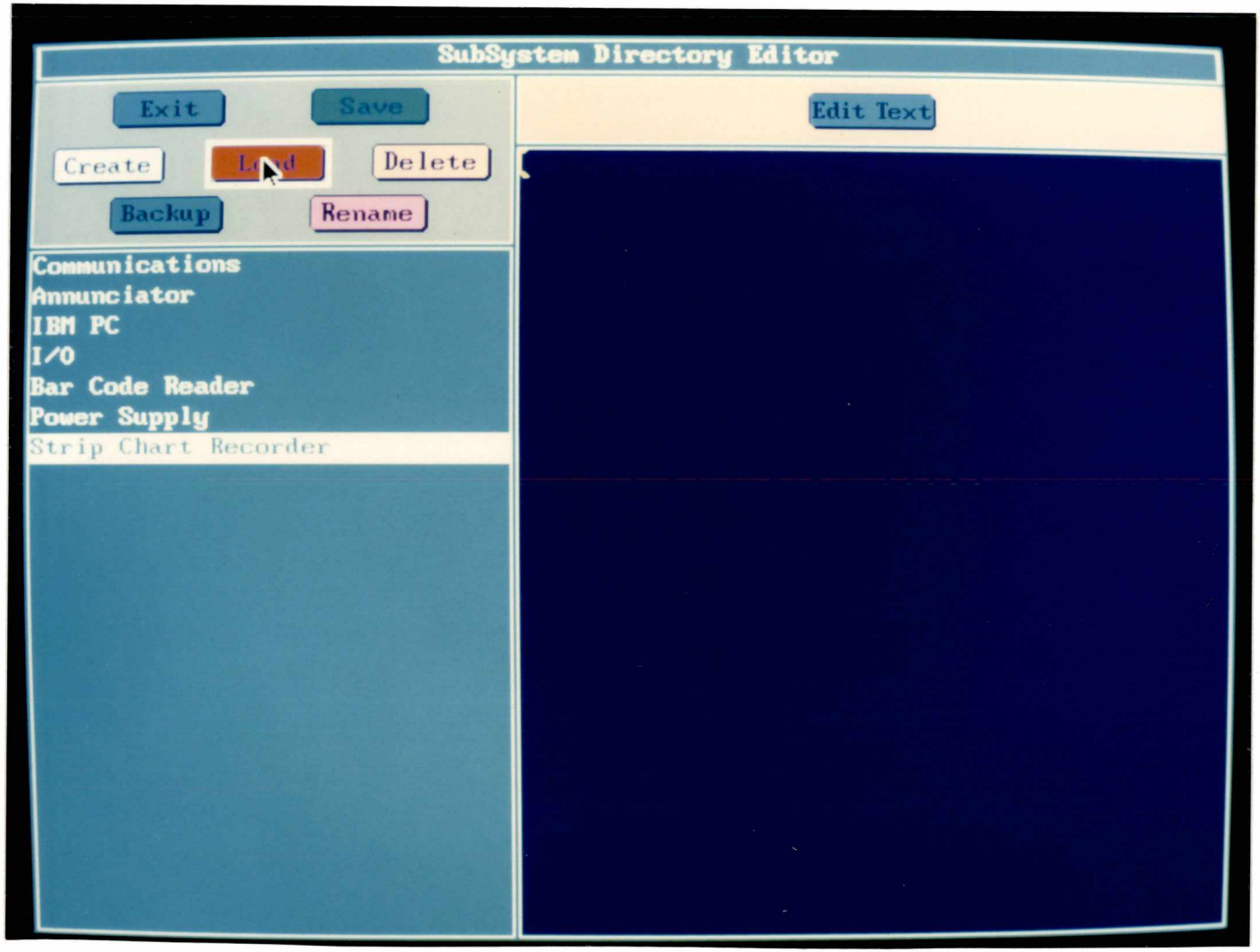


Figure 13 *SubSystem Directory Editor* of *Procedure Authoring System* Authoring Module Interface Screen

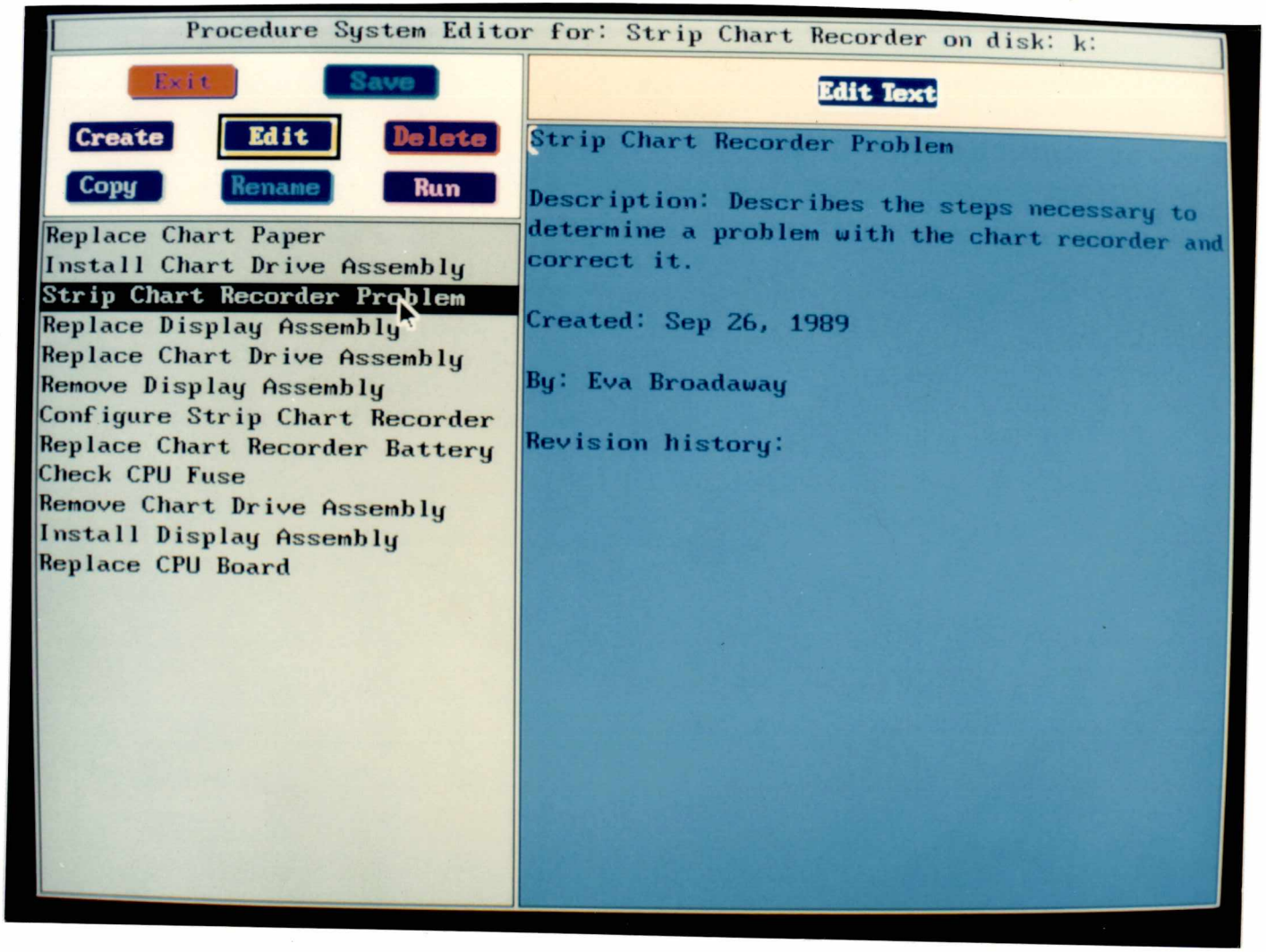


Figure 14 *Procedure System Editor* of *Procedure Authoring System* Authoring Module Interface Screen

Figure 15 depicts a typical *Procedure Editor* interface. Three primary panes exist within this window: the control pane on the left side, the photographic image pane on the upper right, and the text pane on the lower right. This screen layout is very similar to the screen layout for the *Procedure* module of the user interface. The only difference is in the size of the control and text panes which are smaller and larger, respectively; the photographic image pane is identical in size to that in the *Procedure* module. This was done to facilitate the WYSIWYG policy for the author.

In the *Procedure Editor* interface, the control pane has five functional areas. In the upper subpane, **EXIT** is used to return control to the *Procedure System Editor* module after first asking if the author wishes to save the procedure to a file. **SAVE** in this subpane performs the file save function directly and is often used for periodic saves during long editing sessions.

The Step Control subpane is used to move around within the procedure; add, delete, move, and copy steps in the procedure; print the text of the steps in the procedure to a printer; define the appropriate manual reference for a particular step in the procedure; and define a subprocedure which may be accessed from a particular step in the procedure. The **RUN** icon in this subpane transfers control to the *Procedure* module described in section 4.2.2; in this case though, the procedure begins at the current step rather than with step 1.

The Annotation subpane is used to overlay graphics on the photographic images. Color (foreground and background), fill pattern, line width, and line style may be defined. Using these definitions, text, straight lines, rectangles, ellipses, and free drawing may then be created. Objects within the drawing may also be filled. Objects, once created, may be selected and manipulated with the **COPY**, **CUT**, **DELETE**, and **PASTE**

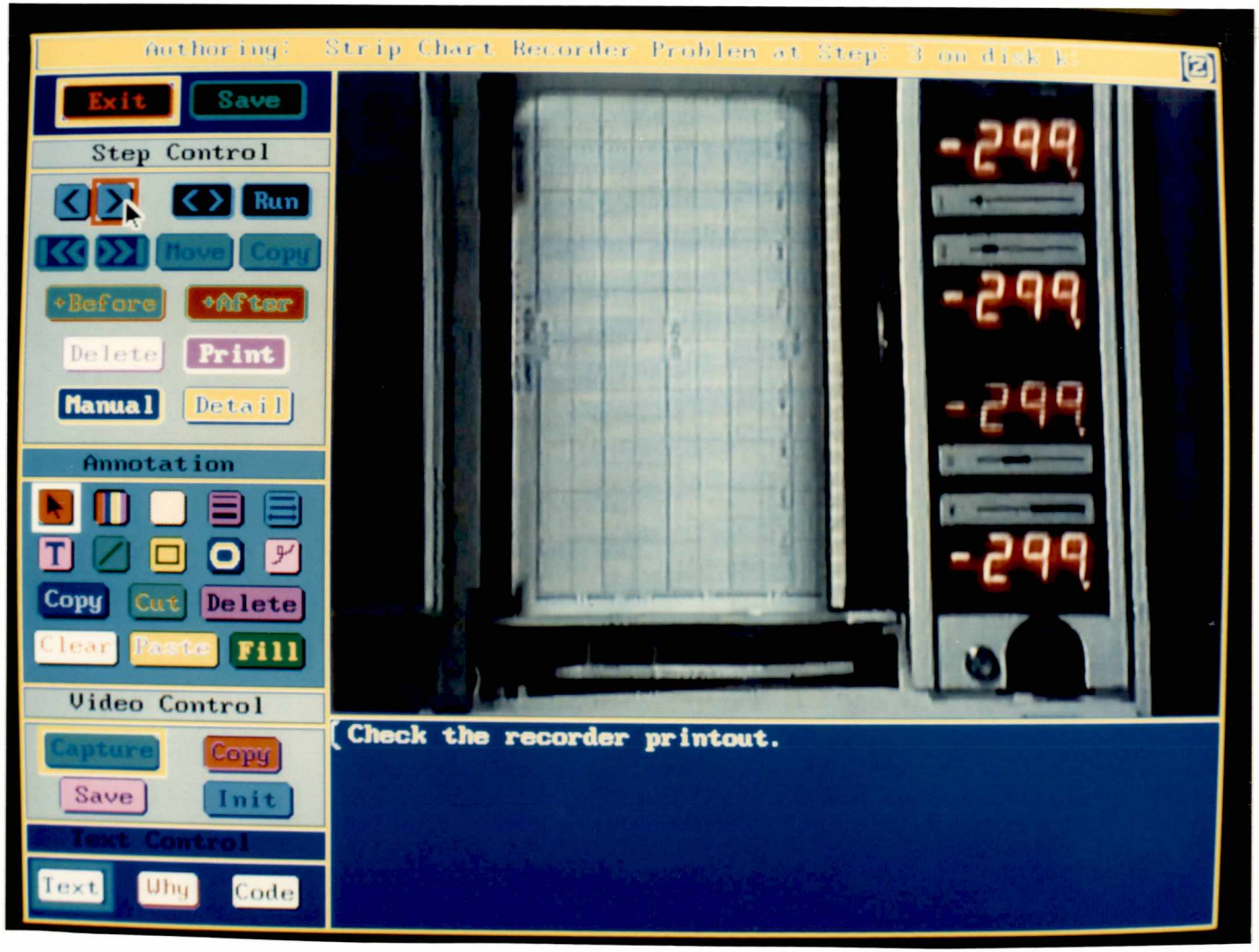


Figure 15 *Procedure Editor* of *Procedure Authoring System* Authoring Module Interface Screen

functions. **CLEAR** erases all annotation for that step. This is a truly object-oriented drawing package since the code to actually draw the annotation is stored as a metafile within each procedure step. A metafile is the term used to refer to a file within a file. Many drawing packages store bit-maps of the images. This can require large amounts of disk storage space and memory for display. The use of annotation in no way alters the photographic image which it overlays.

Photographic images are acquired using the hardware described in section 4.1.2. Control of the photography is performed using the Video Control subpane of the *Procedure Editor* window. The functional interaction with the video capture board is performed by running an external C program with the appropriate parameter. This is done transparently to the author. Selection of the **CAPTURE** icon (when a camera is attached to the video capture board) causes the screen to change to a full-screen, live photographic image. The author may then use the camera features to focus, zoom, etc. When the live image is as the author wishes, a single click of the left mouse button acts as the shutter for the camera. The full-screen image is immediately scaled to fit the photographic image pane and a 10:1 compression is performed. This scale and compression process is completed in less than 10 seconds. The screen then reverts to the typical *Procedure Editor* window form with the new picture in the photographic image pane. The author has the option of saving this image to a file or reshooting. Photographic images may be copied from other procedures or other steps within the current procedure. The **INIT** icon is used when adding or removing the camera from the system while in operation. This forces the video capture board to reinitialize itself and check for an external (camera) sync signal or utilize its own internal sync signal. To simplify the video photography, all photographic image files are automatically named by

the *Procedure Editor.* A video index is maintained internal to each procedure and the photograph files are named in the format, 'P00000nn.pic', where nn is the next sequential number. When a photographic image is copied, it is actually duplicated with a new file name (the next in the sequence). This procedure uses more disk space, but greatly simplifies the file management for the photographic image files. In this way the author can delete a step and its associated photographic image file with no fear of also removing the photographic image for another step.

The Text Control subpane is used to enter descriptive (**TEXT**) or explanatory (**WHY**) text. The **CODE** icon displays any Smalltalk code entered for this step. A second consecutive selection of the **CODE** icon transfers control to the *Code Editor* module. This module, with its interface shown in figure 16, is used to enter Smalltalk code which will be executed when the user advances to the next step. This code may ask questions of the user, record user responses to the result log file, and branch to other steps. Smalltalk code may also be used to directly access the *HyperManual* module, add or remove symptoms from the symptom list of the *Symptom Selector* module, or exit the *Procedure* module. Choosing the **EXIT** icon in the *Code Editor* window returns control to the *Procedure Editor* module at the current step.

Changes made to the procedure become permanent only when it is saved to a file using either the **EXIT** or **SAVE** functions implemented by the icons in the top left subpane.

4.2.4 Expert System

Knowledge FACTory II, a rule-based, forward- and backward-chaining expert system developed in Smalltalk at Georgia Institute of Technology's Research Institute,

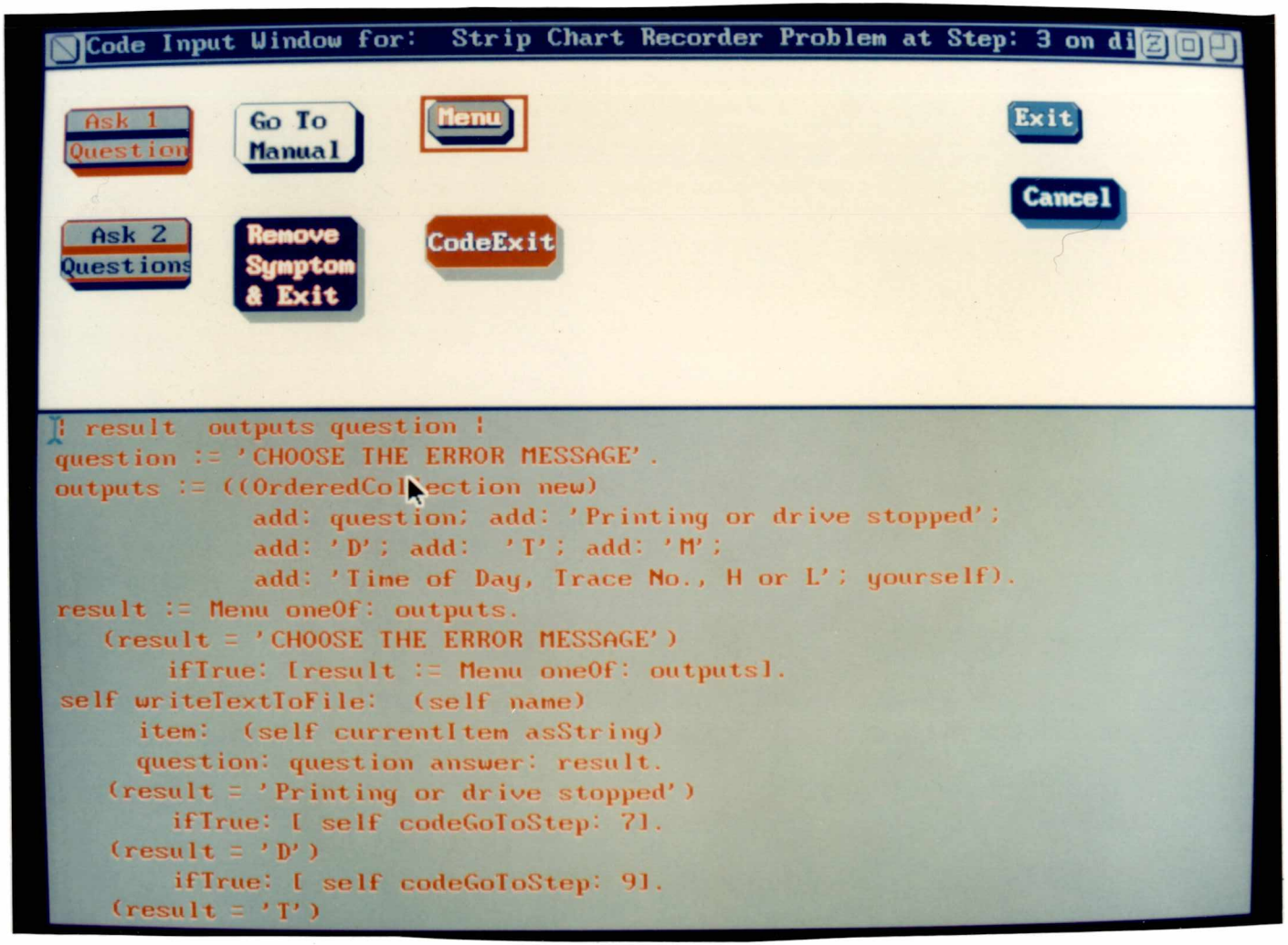


Figure 16 *Code Editor* of *Procedure Editor* Authoring Module Interface Screen

(Smith 1989) is used as the expert system shell for the project. Many commercially available expert system shells offer a wide variety of features but are not easily customized for a specific application. Often these expert system shells have very good authoring interfaces for rule development but very poor user interfaces for actual use. Commercial expert system shells also are often designed to run in a stand-alone mode with little ability to interact with other programs or be embedded within a larger application.

Knowledge FACTory II is a relatively simple expert system shell with string matching used to determine if goals have been met. The original implementation provided two types of rules, AND-IF and OR-IF. Using these rule structures, a list of facts could be checked to determine if all (AND-IF) or any (OR-IF) are true. This has been extended to two additional rule types, AND-NOT-IF and OR-NOT-IF, which are used to determine if all or any of the facts are not true, respectively. Additional rule structures are easily added, if desired. This is a major advantage of the use of this expert system since it can be customized as desired and embedded into the application so that use of it is essentially transparent to the user.

Four classes, Broker, Advisor, Rule, and Fact, comprise Knowledge FACTory II. The Broker coordinates the action of the Advisors, which act as experts within various fields. A typical knowledge base application will have one Broker with a number of Advisors as experts on the various subsystems of the industrial equipment. Advisors use the Rules contained in the knowledge base and any known or established Facts to determine if goals are met.

In a forward-chaining mode, which is used exclusively for this project, the satisfaction of a goal can result in either a new Fact being established or the execution

of Smalltalk code. The initial Facts known to the system are the symptoms selected by the user when operating the *Symptom Selector* module. Additional Facts may be established through the satisfaction of intermediate goals. In this application, all established Facts are immediately publicized so that other Advisors may make use of them. Smalltalk code is used to transfer control from the *Expert System* user module to the *Procedure* module described in section 4.2.2.

4.2.5 Software Maintenance

The issue of software maintenance can be broken down into two areas: source code maintenance for the authoring and user system and maintenance of the applications knowledge base(s). The latter is discussed in section 4.3.

Source code maintenance for the Smalltalk **image** is discussed briefly in section 4.2.1. Additional care was required during periods when multiple developers were generating new Smalltalk code. This was necessary because Smalltalk is not designed for concurrent use by multiple developers. All changes to the code are incorporated into the **image** when it is saved; utilizing another developers **image** will destroy any changes which were made only in the previous image. Initially, much of the concurrent code development was done with different people working on different portions of the code (e.g., one was developing the *SubSystem Directory Editor* while another was working on the *Procedure Editor*). While this was the case, at the end of each day the developers would save the classes or changes to a class which had been made that day. All of these classes or changes were then incorporated into a single **image** and this was distributed to the developers for use the following day. Archive copies of the classes; changes to classes; and **image**, **change.log**, and **sources.sml** files

were also maintained. These are being kept at least until all code development is complete since the cost of diskettes is low compared to the cost of engineering development services.

Later development has required the initiation of an alternative Smalltalk code development and sharing strategy. One of the development stations is designated as the primary system. All code developed on this system is automatically stored when the **image** is saved. For code development on other stations, each developer has a unique project application, and all code development is performed within this project. The project can then be saved to disk, transferred to the primary development station, and incorporated within the **image** on that station. This **image** (with **change.log** and **sources.sml**) is then transferred to the other development stations and is also archived for backup purposes. This code sharing has been facilitated by the use of an Ethernet local area network (LAN) and the Bernoulli cartridge drive for both transfer and archiving of data.

4.3 Knowledge Acquisition

Both synthetic and algorithmic reasoning are used within the knowledge base of this project. This was done in a hybrid expert system design approach which separated information in the knowledge base into two categories: that which required some deduction based on the known or inferred facts and that which was primarily or purely algorithmic in nature (Luger and Stubblefield 1989). The separation of the knowledge base into two parts was chosen because most expert systems perform very poorly on information which is purely algorithmic in nature (Waterman 1986). Analysis of the

nature of the information necessary for effective troubleshooting and repair revealed that once a determination had been made as to the industrial equipment subsystem which needed to investigated, the troubleshooting and repair of the subsystem was then relatively straightforward and typically did not require any synthesis of information. Included in the following section are discussion of these two facets of the knowledge base along with maintenance strategies for the knowledge base.

4.3.1 Synthetic Reasoning

The *Expert System* module described in section 4.2.4 is used to operate on the knowledge base used for synthetic reasoning. The facts on which the *Expert System* module operates are generated by the technician (user) with the *Symptom Selector* module. The information (photographic images and symptom lists) found in the *Symptom Selector* module, while not requiring reasoning, is still essential to the performance of the *Expert System* module.

The *Expert System* knowledge base was constructed using general troubleshooting techniques, such as checking for power to the equipment first, and conducting interviews with engineers familiar with the equipment. This has been an iterative process, with the first implementation able to handle single symptoms only. Subsequent analysis revealed that the symptoms could generally be divided into two groups: those with low priority which may be the result of the failure of another subsystem and those with high priority in resulting in the failure of other subsystems. A bar code reader failure may fall in the first category while a power supply failure would fall into the second category. Multiple symptoms were addressed in this way, with general rules written to cover the cases of lower priority symptoms and then specific rules for each of the higher priority symptoms

in conjunction with other symptoms. The rules are by no means exhaustive, and it is expected that additional rules will be added as experience in use of the system continues.

Maintenance of this knowledge base is currently performed manually by storing all rules and relationships in external files. This procedure is described in section 4.2.3. Appendix A lists the text of the knowledge base for the initial industrial equipment application. Since the knowledge base is maintained external to the Smalltalk **image**, problems can arise in determining if the text of the external file agrees with the actual expert system operation. No changes should be made to the external file unless they are to be immediately incorporated into the **image**. This can be accomplished by editing the knowledge base file on a computer system other than the one on which the current primary **image** resides and then transferring the completed updates to the primary development station.

4.3.2 Algorithmic Reasoning

The algorithmic reasoning portion of the knowledge base uses the *Procedure System Editor* for development and maintenance. This information is presented to the user with the *Procedure* module of the user interface. A procedure is actually nothing more than a collection of steps. Information contained for each step includes descriptive and explanatory text, Smalltalk code, a video picture file name, annotation information, the hierarchy of the appropriate manual location, and the name of any subprocedure. Procedural information is actually divided into two types. The top-level procedures tell the technician what to do, i.e., "Replace CPU board." These procedures are highly interactive and often ask questions of the user. All questions and user responses are

recorded in the result log file. Subprocedures are used to describe the sequence of steps necessary to actually replace the faulty component. The text of a typical top-level procedure is given in Appendix B. Text for an accompanying subprocedure is presented in Appendix C.

The knowledge base contained in a procedure consists of three elements. The first component is the descriptive and explanatory text and associated Smalltalk code. This was developed either by using established procedures or by writing new procedures (this was more often the case). A video picture file name and the annotation associated with it are the second component of the algorithmic knowledge base. The third component is the manual reference hierarchy and any subprocedure reference.

Manual text for the system is introduced via the *HyperManual Browser* module of the authoring interface. Much of this information needed for the initial industrial equipment application was available in machine readable format on either magnetic tape or floppy diskette. Maintenance of these files is fairly easy since updates to the text can be made simply by editing or replacing the appropriate text file. No direct interaction with Smalltalk is required for this. Maintenance of the hierarchy and cross-reference file for the system is similar to that for the synthetic knowledge base file. The updated *HyperManual* information can be directly incorporated into the Smalltalk **image** via a "fileIn" process however.

Maintenance of the actual procedure files is also fairly straightforward. All of the algorithmic knowledge base procedures are maintained in files external to Smalltalk. None of the information actually resides in the **image**. Whenever a procedure is accessed in Smalltalk, it must be "readIn" from the external file. Additionally, each of

the photographic images resides in a separate file on the disk. Maintenance of these files is discussed in section 4.2.3 on the authoring interface.

The disk partitioning and directory structure of DOS was used to further simplify file maintenance. The 380-Mbyte hard disk was divided into 19 partitions of 20 Mbytes each. Partitions C: and D: are used for DOS and Smalltalk sources, respectively. The E: partition is used for the synthetic reasoning knowledge base and includes the *Symptom Selector* module photographic images. Partitions F: through U: are utilized for procedure files. The organization is such that a particular subsystem resides entirely on a single disk partition. The system definition file for related procedures described in section 4.2.3 resides in the root directory of the disk partition. The procedure files associated with this subsystem are each contained in a separate subdirectory. Each subdirectory also contains all the photographic images associated with the procedure. In this way, an entire subsystem can be archived or restored simply by copying from or to the entire partition.

5. SUMMARY AND CONCLUSIONS

The Smalltalk V/286 object-oriented software environment has proven to be a good choice for system development and implementation. It provided a good mechanism for user interface development and display for both the user and authoring systems. It facilitated incremental development of both the system and the application and simplified the addition of new features.

The hybrid software design approach to knowledge base development for the project has also proven effective. The primary focus of the development on the interfaces (both user and authoring) and creation of the knowledge base with use of the expert system secondary has worked well.

Although the current hardware configuration is bulky, the size of the PC itself can be reduced in future versions. A source has been located for a VGA-type color display which can be mounted within the PC cabinet. A source also now exists for a transportable 80386 computer with integral color VGA, CRT-type display.

The modularity of the system design in both hardware and software will prove useful as newer video and display boards become available and new features are desired. The system can be upgraded with new hardware every few years if desired.

Although the system has been completed for a single application on a particular type of industrial equipment, it is planned to extend it to other similar industrial equipment and eventually to dissimilar types of industrial equipment.

Enhancements to the system are anticipated. Audio output for instruction text will remove the need for the technician to read much of the information. This will free his eyes to observe the video portion of the screen and the equipment under repair.

A voice recognition module with a limited vocabulary will eliminate the trackball and free the technician's hands for work on the equipment.

This initial implementation meets the goal of incorporating all relevant information for troubleshooting and repair into a single transportable source. It remains to be seen, however, if use of this system will result in increased productivity due to reduced time for diagnosis and repair of faults. This can only be determined with continued experience in use of the system by the project sponsor.

LIST OF REFERENCES

LIST OF REFERENCES

Cleal, D. M. and N. O. Heaton, 1988, *Knowledge-Based Systems: Implications for Human-Computer Interfaces*, Ellis Horwood Limited, Chichister, West Sussex, England.

Cox, Brad and Bill Hunt, 1986, "Objects, Icons, and Software-ICs," *BYTE*, vol. 11, no. 8, pp. 161-176 (August).

Digitalk, Inc., 1988, *Smalltalk/V 286 User's Manual*, Digitalk, Inc., Los Angeles.

Discrete TimeSystems, Inc., 1988, *CVP-2000 Color Video Processors Programmer's Manual*, Discrete TimeSystems, Inc., Arlington Heights, Ill.

Goldberg, Adele, 1984, *Smalltalk-80—The Interactive Programming Environment*, Addison-Wesley, Reading, Mass.

Hart, Anna, 1986, *Knowledge Acquisition for Expert Systems*, McGraw-Hill, New York.

Isle, Brian A., Charles P. Bloom and Arch Butler, 1989, "The Delivery of Expert Systems for Special Environments", *Proceedings of Seventh Conference on Interactive Instruction Delivery*, pp. 138-142, Society for Applied Learning Technology, Warrenton, Vir.

Kantowitz, Barry H. and Robert D. Sorkin, 1983, *Human Factors: Understanding People-System Relationships*, John Wiley and Sons, New York.

Kidd, Alison L. and Martin B. Cooper, 1985, "Man-machine interface issues in the construction and use of an expert system," *Int. J. Man-Machine Studies*, vol. 22, no. 1, pp. 91-102 (January).

Luger, George F. and William A. Stubblefield, 1989, *Artificial Intelligence and the Design of Expert Systems*, Benjamin/Cummings, Redwood City, Calif.

McCormick, Ernest J. and Mark S. Sanders, 1982, *Human Factors in Engineering and Design*, McGraw-Hill, New York.

Pascoe, Geoffrey A., 1986, "Elements of Object-Oriented Programming," *BYTE*, vol. 11, no. 8, pp. 139-144 (August).

Porat, Shai and David Rhyne, 1990, "Computerizing Maintenance Expertise," *Chilton's I&CS*, vol. 63, no. 2, pp. 63-66 (March).

Rubin, Kenneth S., et al., "A Smalltalk Implementation of an Intelligent Operator's Associate," *OOPSLA '88 Conference Proceedings*, vol. 22, no. 12, pp. 234-247, ACM Press, New York.

Smith, John F., 1989, *Knowledge FACTory II Usage*, Georgia Institute of Technology, Atlanta.

Uebbing, John and Charles Young, 1986, "User Interface Performance Issues," *BYTE*, vol. 11, no. 8, pp. 175 (August).

Waterman, Donald A., 1986, *A Guide to Expert Systems*, Addison-Wesley, Reading, Mass.

APPENDICES

APPENDIX A

SYNTHETIC REASONING KNOWLEDGE BASE TEXT

```
" Set up broker for FE01 "
EISFE01 := Broker new.

" Set up an advisor and rules for each major piece "
EISPowerSupply := Advisor new.
EISAnnunciator := Advisor new.
EISInputOutput := Advisor new.
EISIBMpc := Advisor new.
EISCommunications := Advisor new.
EISBarcodeReader := Advisor new.
EISStripChartRecorder := Advisor new.
EISMiscellaneous := Advisor new.
EISFE01 addToTeam: EISPowerSupply; addToTeam: EISAnnunciator; addToTeam:
EISInputOutput;
                addToTeam: EISIBMpc; addToTeam: EISCommunications; addToTeam:
EISBarcodeReader;
                addToTeam: EISStripChartRecorder; addToTeam: EISMiscellaneous.


EISPowerSupply

    ruleName: 'nothing except field power supply'
    andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
                ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' IBM PC Fixed Disk ' ' Host Communications '
                ' Net/Mon Communications ' ' Barcode Reader ' ' Strip Chart Recorder
')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                (File pathName: 'g:\chfps\chckflps.t03'))];

    ruleName: 'nothing except PS noLeds'
    andNotIfs: #(' Annunciator ' ' EIS Power Supply SomeLEDs '
                ' Input ' ' Output ' ' IBM PC Display '
                ' IBM PC Fixed Disk ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' Host Communications ' ' Net/Mon Communications '
                ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                (File pathName: 'g:\psnle\pwsnledo.t03'))];

    ruleName: 'nothing except PS someLeds'
    andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' Input ' ' Output ' ' IBM PC Display '
```

```
                ' IBM PC Fixed Disk ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' Host Communications ' ' Net/Mon Communications '
                ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                  (File pathName: 'g:\pssle\pwssledo.t03'))];

   ruleName: 'other than field power supply'
    orIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
                ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' IBM PC Fixed Disk ' ' Host Communications '
                ' Net/Mon Communications ' ' Barcode Reader ' ' Strip Chart Recorder
')
    then: #('other than Field Power Supply');

    ruleName: 'field power supply'
    andIfs: #(' Field Power Supply ' 'other than Field Power Supply')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                  (File pathName: 'g:\chfps\chckflps.t03'))];

   ruleName: 'other than EIS Power Supply NoLEDs'
    orIfs: #(' Annunciator ' ' Field Power Supply '
                ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
                ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' IBM PC Fixed Disk ' ' Host Communications '
                ' Net/Mon Communications ' ' Barcode Reader ' ' Strip Chart Recorder
')
    then: #('other than EIS Power Supply NoLEDs');

    ruleName: 'PS noLeds'
    andIfs: #(' EIS Power Supply NoLEDs '
                'other than EIS Power Supply NoLEDs')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                  (File pathName: 'g:\psnle\pwsnledo.t03'))];

   ruleName: 'other than EIS Power Supply SomeLEDs'
    orIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' Field Power Supply ' ' Input ' ' Output '
                ' IBM PC Display ' ' IBM PC Keyboard '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem ' ' IBM PC
Diskette Drive '
```

```
            ' IBM PC Fixed Disk ' ' Host Communications '
            ' Net/Mon Communications ' ' Barcode Reader ' ' Strip Chart Recorder
')
    then: #('other than EIS Power Supply SomeLEDs');

    ruleName: 'PS someLeds'
    andIfs: #(' EIS Power Supply SomeLEDs '
            'other than EIS Power Supply SomeLEDs')
    then: [(TutorialGuide new)
          openOn: (Tutorial readFrom:
              (File pathName: 'g:\pssle\pwssledo.t03'))];

   ruleName: 'not power supply'
    orNotIfs: #( ' EIS Power Supply NoLEDs '
          ' Field Power Supply '
          ' EIS Power Supply SomeLEDs ')
    then: #('Not Power Supply').

EISPowerSupply  addToAgenda: 'nothing except PS noLEDs';
            addToAgenda: 'nothing except PS someLEDs';
            addToAgenda: 'nothing except field power supply';
            addToAgenda: 'other than field power supply';
            addToAgenda: 'field power supply';
            addToAgenda: 'other than EIS Power Supply SomeLEDs';
            addToAgenda: 'PS someLeds';
            addToAgenda: 'other than EIS Power Supply NoLEDs';
            addToAgenda: 'PS noLeds';
            addToAgenda: 'not power supply'.

EISAnnunciator

    ruleName: 'nothing except annunciator'
    andNotIfs: #(' EIS Power Supply NoLEDs ' ' EIS Power Supply SomeLEDs '
            ' Input ' ' Output ' ' IBM PC Display '
            ' IBM PC Fixed Disk ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
            ' IBM PC System Board ' ' IBM PC Undetermined Problem '
            ' Host Communications ' ' Net/Mon Communications '
            ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
              (File pathName: 'm:\annnp\annnctrp.t03'))];

    ruleName: 'annunciator'
    andIfs: #(' Annunciator ' 'Not Power Supply')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
              (File pathName: 'm:\annnp\annnctrp.t03'))].
```

```
EISAnnunciator  addToAgenda: 'nothing except annunciator';
                addToAgenda: 'annunciator'.

EISInputOutput

    ruleName: 'nothing except input'
    andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' EIS Power Supply SomeLEDs ' ' Output ' ' IBM PC Display '
                ' IBM PC Fixed Disk ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' Host Communications ' ' Net/Mon Communications '
                ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                    (File pathName: 'f:\inptp\inptprbl.t03'))];

    ruleName: 'nothing except output'
    andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' EIS Power Supply SomeLEDs ' ' Input ' ' IBM PC Display '
                ' IBM PC Fixed Disk ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' Host Communications ' ' Net/Mon Communications '
                ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                    (File pathName: 'f:\otptp\otptprbl.t03'))];

    ruleName: 'input/output problem'
    andIfs: #(' Input ' ' Output ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                    (File pathName: 'f:\otptp\otptprbl.t03'))].

EISInputOutput  addToAgenda: 'nothing except input';
                addToAgenda: 'nothing except output';
                addToAgenda: 'input/output problem'.

EISIBMpc

    ruleName: 'nothing except IBM display'
    andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
                ' IBM PC Fixed Disk ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' Host Communications ' ' Net/Mon Communications '
                ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
```

```
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\ibmcd\ibmclrds.t03'))];

ruleName: 'IBM display and lower priority symptoms'
andIfs: #(' IBM PC Display ' 'Lower Priority Symptoms')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\ibmcd\ibmclrds.t03'))];

ruleName: 'nothing except IBM fixed disk'
andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
            ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
            ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
            ' IBM PC System Board ' ' IBM PC Undetermined Problem '
            ' Host Communications ' ' Net/Mon Communications '
            ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\fxddd\fxddskdr.t03'))];

ruleName: 'IBM fixed disk and lower priority symptoms'
andIfs: #(' IBM PC Fixed Disk ' 'Lower Priority Symptoms')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\fxddd\fxddskdr.t03'))];

ruleName: 'nothing except IBM diskette drive'
andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
            ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
            ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Fixed Disk '
            ' IBM PC System Board ' ' IBM PC Undetermined Problem '
            ' Host Communications ' ' Net/Mon Communications '
            ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\dskdp\dskttdrp.t03'))];

ruleName: 'IBM diskette drive and lower priority symptoms'
andIfs: #(' IBM PC Diskette Drive ' 'Lower Priority Symptoms')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\dskdp\dskttdrp.t03'))];

ruleName: 'nothing except IBM keyboard'
andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
            ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
```

```
            ' IBM PC Display ' ' IBM PC Fixed Disk ' ' IBM PC Diskette Drive '
            ' IBM PC System Board ' ' IBM PC Undetermined Problem '
            ' Host Communications ' ' Net/Mon Communications '
            ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\kybrp\kybrdprb.t03'))];

ruleName: 'IBM keyboard and lower priority symptoms'
andIfs: #(' IBM PC Keyboard ' 'Lower Priority Symptoms')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\kybrp\kybrdprb.t03'))];

ruleName: 'nothing except IBM system board'
andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
            ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
            ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
            ' IBM PC Fixed Disk ' ' IBM PC Undetermined Problem '
            ' Host Communications ' ' Net/Mon Communications '
            ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\sysbp\systmbrp.t03'))];

ruleName: 'IBM system board and lower priority symptoms'
andIfs: #(' IBM PC System Board ' 'Lower Priority Symptoms')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\sysbp\systmbrp.t03'))];

ruleName: 'nothing except IBM undetermined problem'
andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
            ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
            ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
            ' IBM PC System Board ' ' IBM PC Fixed Disk '
            ' Host Communications ' ' Net/Mon Communications '
            ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\undtp\undtrmnp.t03'))];

ruleName: 'IBM undetermined problem and lower priority symptoms'
andIfs: #(' IBM PC Undetermined Problem ' 'Lower Priority Symptoms')
then: [(TutorialGuide new)
        openOn: (Tutorial readFrom:
              (File pathName: 'l:\undtp\undtrmnp.t03'))];
```

```
    ruleName: 'any IBM PC part'
    orIfs: #(' IBM PC Display ' ' IBM PC Keyboard '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' IBM PC Fixed Disk ' ' IBM PC Diskette Drive ')
    then: #('any IBM PC part').


EISIBMpc    addToAgenda: 'nothing except IBM display';
            addToAgenda: 'nothing except IBM fixed disk';
            addToAgenda: 'nothing except IBM diskette drive';
            addToAgenda: 'nothing except IBM keyboard';
            addToAgenda: 'nothing except IBM system board';
            addToAgenda: 'nothing except IBM undetermined problem';
            addToAgenda: 'any IBM PC part';
            addToAgenda: 'IBM display and lower priority symptoms';
            addToAgenda: 'IBM fixed disk and lower priority symptoms';
            addToAgenda: 'IBM diskette drive and lower priority symptoms';
            addToAgenda: 'IBM keyboard and lower priority symptoms';
            addToAgenda: 'IBM system board and lower priority symptoms';
            addToAgenda: 'IBM undetermined problem and lower priority symptoms'.

EISCommunications

    ruleName: 'nothing except host communications'
    andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
                ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' IBM PC Fixed Disk ' ' Net/Mon Communications '
                ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                    (File pathName: 'h:\hstcp\hstcmmnp.t03'))];

    ruleName: 'nothing except net/mon communications'
    andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
                ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' IBM PC Fixed Disk ' ' Host Communications '
                ' Barcode Reader ' ' Field Power Supply ' ' Strip Chart Recorder ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                    (File pathName: 'h:\ntmcp\ntmncmmp.t03'))];

    ruleName: 'host communications and any PC part'
```

```
    andIfs: #(' Host Communications '
                'any IBM PC part')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                (File pathName: 'h:\hstcp\hstcmmnp.t03'))];

    ruleName: 'host communications and lower priority symptom'
    andIfs: #(' Host Communications '
                'Lower Priority Symptoms')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                (File pathName: 'h:\hstcp\hstcmmnp.t03'))];

    ruleName: 'net/mon communications and I/O'
    andIfs: #(' Net/Mon Communications ' 'SCR or BCR or IO')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                (File pathName: 'h:\ntmcp\ntmncmmp.t03'))].

EISCommunications   addToAgenda: 'nothing except host communications';
                    addToAgenda: 'nothing except net/mon communications';
                    addToAgenda: 'host communications and any PC part';
                    addToAgenda: 'host communications and lower priority symptom';
                    addToAgenda: 'net/mon communications and I/O'.

EISBarcodeReader

    ruleName: 'nothing except barcode reader'
    andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
                ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
                ' IBM PC System Board ' ' IBM PC Undetermined Problem '
                ' IBM PC Fixed Disk ' ' Host Communications '
                ' Net/Mon Communications ' ' Field Power Supply ' ' Strip Chart
Recorder ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                (File pathName: 'j:\brcrp\brcdrdrp.t03'))].

EISBarcodeReader  addToAgenda: 'nothing except barcode reader'.

EISStripChartRecorder

    ruleName: 'nothing except strip chart recorder'
    andNotIfs: #(' Annunciator ' ' EIS Power Supply NoLEDs '
                ' EIS Power Supply SomeLEDs ' ' Input ' ' Output '
                ' IBM PC Display ' ' IBM PC Keyboard ' ' IBM PC Diskette Drive '
```

```
            ' IBM PC System Board ' ' IBM PC Undetermined Problem '
            ' IBM PC Fixed Disk ' ' Host Communications '
            ' Net/Mon Communications ' ' Barcode Reader ' ' Field Power Supply ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                (File pathName: 'k:\stcrp\strpchrp.t03'))];

    ruleName: 'strip chart recorder and output'
    andIfs: #(' Strip Chart Recorder ' ' Output ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                (File pathName: 'k:\stcrp\strpchrp.t03'))];

    ruleName: 'strip chart recorder and input'
    andIfs: #(' Strip Chart Recorder ' ' Input ')
    then: [(TutorialGuide new)
            openOn: (Tutorial readFrom:
                (File pathName: 'k:\stcrp\strpchrp.t03'))].

EISStripChartRecorder   addToAgenda: 'nothing except strip chart recorder';
                        addToAgenda: 'strip chart recorder and output';
                        addToAgenda: 'strip chart recorder and input'.

EISMiscellaneous
    ruleName: 'miscellaneous 1'
    orIfs: #(' Strip Chart Recorder ' ' Barcode Reader '
            ' Input ' ' Output ')
    then: #( 'SCR or BCR or IO');

    ruleName: 'miscellaneous 2'
    orIfs: #('SCR or BCR or IO' ' Net/Mon Communications ')
    then: #( 'Lower Priority Symptoms').

EISMiscellaneous    addToAgenda: 'miscellaneous 1';
                    addToAgenda: 'miscellaneous 2'.

EISFE01 allOutput: (ColorTextEditor windowLabeled: 'FE01 Expert System'
                            frame: (0@0 extent: 640@480)
                            foreColor: 12
                            backColor: 1).
```

APPENDIX B

ALGORITHMIC REASONING TOP-LEVEL PROCEDURE TEXT

```
Strip Chart Recorder Problem
18
nil
~
P0000000.pic
blank
nil
Strip Chart Recorder
nil

You have indicated that there is a problem with the chart recorder.
 ((self flag) = false)
    ifTrue: [self codeRemoveSymptom: ' Strip Chart Recorder '.
             SymptomPane bottom: #nil]



~
P0000001.pic
blank
nil
Operation
Strip Chart Recorder
nil

 Check the status LEDs on the front of the recorder.
|  var1 var2 question |
  question :='Is a red LED flashing?'.
  var1 := (Menu confirm: question).

   (var1)
    ifTrue: [self writeTextToFile: (self name)
        item: (self currentItem asString) question: question answer: ' yes']
    ifFalse: [self writeTextToFile: (self name)
        item: (self currentItem asString) question: question answer: ' no'].

  (var1)
    ifTrue: [self codeGoToStep: 3]
    ifFalse: [ question := 'Is the green power LED on?'.
               var2 := (Menu confirm: question).

               (var2)
                  ifTrue: [self writeTextToFile: (self name)
                       item: (self currentItem asString) question: question answer: ' yes']
                  ifFalse: [self writeTextToFile: (self name)
                       item: (self currentItem asString) question: question answer: ' no'].
```

```
            (var2)
               ifTrue: [self codeGoToStep: 13]
               ifFalse: [self codeGoToStep: 4]].
```

~

P0000002.pic
blank
nil
Operation
Strip Chart Recorder
nil

```
 Check the recorder printout.
| result  outputs question |
 question := 'CHOOSE THE ERROR MESSAGE'.
 outputs := ((OrderedCollection new)
             add: question; add: 'Printing or drive stopped';
             add: 'D'; add:  'T'; add: 'M';
             add: 'Time of Day, Trace No., H or L'; yourself).
 result := Menu oneOf: outputs.
    (result = 'CHOOSE THE ERROR MESSAGE')
        ifTrue: [result := Menu oneOf: outputs].
 self writeTextToFile:  (self name)
      item:  (self currentItem asString)
      question: question answer: result.
    (result = 'Printing or drive stopped')
        ifTrue: [ self codeGoToStep: 7].
    (result = 'D')
        ifTrue: [ self codeGoToStep: 9].
    (result = 'T')
        ifTrue: [ self codeGoToStep: 10].
    (result = 'M')
         ifTrue: [ self codeGoToStep: 11].
     (result = 'Time of Day, Trace No., H or L')
         ifTrue: [ self codeGoToStep: 12].
```

~

P0000003.pic
blank
nil
Strip Chart Recorder
nil

```
 Check the power to the recorder.
```

```
| var1 question |
question := 'Is there power to the recorder?'.
var1 := (Menu confirm: question).
(var1)
        ifTrue: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer: ' yes']
        ifFalse: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer:' no'].
 (var1)
      ifTrue: [self codeGoToStep: 6]
      ifFalse: [self codeGoToStep: 5].


~
P0000004.pic
blank
nil
BristolManual
nil

 You may have a power supply problem.
 The symptom list will be updated with this information.

 Please select GO from the Symptom Selector when control is returned to it.
self codeAddSymptom: ' EIS SomeLEDs On '.
self codeRemoveSymptom: ' Strip Chart Recorder '.
SymptomPane bottom: #nil.
self codeExit.
~
P0000005.pic
blank
'Check CPU Fuse'
Service
Strip Chart Recorder
nil

 Check the CPU fuse and replace it if it is bad.
| var1 question |
question := 'Does the Strip Chart Recorder now function properly?'.
var1 := (Menu confirm: question).
(var1)
        ifTrue: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer: ' yes']
        ifFalse: [self writeTextToFile: (self name)
```

```
                                item: (self currentItem asString) question: question
answer:' no'].
 (var1)
      ifTrue: [self codeRemoveSymptom: ' Strip Chart Recorder '.
               SymptomPane bottom: #nil.
               self codeExit]
      ifFalse: [self codeGoToStep: 13].
~
P0000006.pic
blank
nil
Operation
Strip Chart Recorder
nil

 Make sure the chart drive is turned on.
~
P0000007.pic
blank
'Replace Chart Paper'
Service
Strip Chart Recorder
nil

 If the recorder is out of paper, install a new roll.
| question  var1 |
question := 'Does the Strip Chart Recorder now function properly?'.
var1 := (Menu confirm: question).
(var1)
       ifTrue: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer: ' yes']
       ifFalse: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer:' no'].
 (var1)
      ifTrue: [self codeRemoveSymptom: ' Strip Chart Recorder '.
               SymptomPane bottom: #nil.
               self codeExit]
      ifFalse: [self codeGoToStep: 13].

~
P0000008.pic
blank
nil
Calibration
Strip Chart Recorder
```

```
nil

 Recalibrate the DAC.
| question  var1 |
question := 'Does the Strip Chart Recorder now function properly?'.
var1 := (Menu confirm: question).
(var1)
        ifTrue: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer: ' yes']
        ifFalse: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer:' no'].
 (var1)
      ifTrue: [self codeRemoveSymptom: ' Strip Chart Recorder '.
               SymptomPane bottom: #nil.
               self codeExit]
      ifFalse: [self codeGoToStep: 13].



~
P0000009.pic
blank
'Configure Strip Chart Recorder'
Operation
Strip Chart Recorder
nil

 Reset the recorder time.
| var1 question |
question := 'Does the Strip Chart Recorder now function properly?'.
var1 := (Menu confirm: question).
(var1)
        ifTrue: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer: ' yes']
        ifFalse: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer:' no'].
 (var1)
      ifTrue: [self codeRemoveSymptom: ' Strip Chart Recorder '.
               SymptomPane bottom: #nil.
               self codeExit]
      ifFalse: [self codeGoToStep: 13].

~
```

```
P0000010.pic
blank
'Configure Strip Chart Recorder'
Calibration
Strip Chart Recorder
nil

 The operating parameters have been lost.  They need to be reprogrammed
using the configurator.
| var1 question |
question := 'Does the Strip Chart Recorder now function properly?'.
var1 := (Menu confirm: question).
(var1)
        ifTrue: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer: ' yes']
        ifFalse: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer:' no'].
 (var1)
      ifTrue: [self codeRemoveSymptom: ' Strip Chart Recorder '.
              SymptomPane bottom: #nil.
              self codeExit]
      ifFalse: [self codeGoToStep: 13].

~
P0000011.pic
blank
nil
Service
Strip Chart Recorder
nil

 One of the input signals is in alarm.  Press Print/Ack to acknowledge
the signal.
| var1 question |
question := 'Does the Strip Chart Recorder now function properly?'.
var1 := (Menu confirm: question).
(var1)
        ifTrue: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer: ' yes']
        ifFalse: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer:' no'].
 (var1)
      ifTrue: [self codeRemoveSymptom: ' Strip Chart Recorder '.
```

```
            SymptomPane bottom: #nil.
            self codeExit]
      ifFalse: [self codeGoToStep: 13].

~
P0000012.pic
blank
nil
Service
Strip Chart Recorder
nil

 Check the Display Panel and Chart Drive.
| var1 var2 question resultfile |
question := 'Does the display work properly?'.
var1 := Menu confirm: question.
    (var1)
      ifTrue: [self writeTextToFile: (self name)
        item: (self currentItem asString) question: question answer: ' yes']
      ifFalse: [self writeTextToFile: (self name)
        item: (self currentItem asString) question: question answer: ' no'].


question := 'Does the chart drive work properly?'.
var2 := Menu confirm: question.
   (var2)
      ifTrue: [self writeTextToFile: (self name)
        item: (self currentItem asString) question: question answer: ' yes']
      ifFalse: [self writeTextToFile: (self name)
        item: (self currentItem asString) question: question answer: ' no'].



(var1 and: [var2])
      ifTrue: [self codeGoToStep: 17]
      ifFalse: [(var1)
                  ifTrue: [self codeGoToStep: 16].
                (var2)
                  ifTrue: [self codeGoToStep: 15].
                (var1 or: [var2])
                  ifFalse: [self codeGoToStep: 14]].

~
P0000013.pic
blank
'Replace CPU Board'
Service
```

```
Strip Chart Recorder
nil

 Replace the CPU board.
| var1 question |
question := 'Does the Strip Chart Recorder now function properly?'.
var1 := (Menu confirm: question).
(var1)
        ifTrue: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer: ' yes']
        ifFalse: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer:' no'].
 (var1)
      ifTrue: [self codeRemoveSymptom: ' Strip Chart Recorder '.
              SymptomPane bottom: #nil.
              self codeExit]
      ifFalse: [self codeGoToStep: 15].


~
P0000014.pic
blank
nil
'Replace Display Assembly'
Service
Strip Chart Recorder
nil

 Replace the Display Assembly.
| var1 question |
question := 'Does the Strip Chart Recorder now function properly?'.
var1 := (Menu confirm: question).
(var1)
        ifTrue: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer: ' yes']
        ifFalse: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer:' no'].
 (var1)
      ifTrue: [self codeRemoveSymptom: ' Strip Chart Recorder '.
              SymptomPane bottom: #nil.
              self codeExit]
      ifFalse: [self codeGoToStep: 16].
```

```
~
P0000015.pic
blank
'Replace Chart Drive Assembly'
Service
Strip Chart Recorder
nil

 Replace the Chart Drive Assembly.
| var1 question |
question := 'Does the Strip Chart Recorder now function properly?'.
var1 := (Menu confirm: question).
(var1)
        ifTrue: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer: ' yes']
        ifFalse: [self writeTextToFile: (self name)
                                item: (self currentItem asString) question: question
answer:' no'].
 (var1)
      ifTrue: [self codeRemoveSymptom: ' Strip Chart Recorder '.
               SymptomPane bottom: #nil.
               self codeExit]
      ifFalse: [self codeGoToStep: 17].


~
P0000016.pic
blank
nil
Strip Chart Recorder
nil

 Problem not found.
 Please select additional symptoms and then GO from the Symptom Selector
 or consult the system engineer.
self codeRemoveSymptom: ' Strip Chart Recorder '.
SymptomPane bottom: #nil.
self codeExit


%
```

APPENDIX C

ALGORITHMIC REASONING SUBPROCEDURE TEXT

Check CPU Fuse
7
nil
~
P0000000.pic
blank
nil
Service
Strip Chart Recorder
nil

Remove power to the strip chart recorder.
~
P0000006.pic
blank
nil
Service
Strip Chart Recorder
nil

Flip down the service identifier panel and open the viewing door.
~
P0000005.pic
blank
nil
Service
Strip Chart Recorder
nil

Unscrew the Display Assembly retaining screw.
~
P0000004.pic
blank
nil
Service
Strip Chart Recorder
nil

Remove the Display Assembly.
~
P0000003.pic
blank
nil
Service
Strip Chart Recorder

nil
Remove the ribbon cable connector from the CPU Board and place the Display Assembly aside.
~
P0000002.pic
blank
nil
Service
Strip Chart Recorder
nil

Grasp the handle of the CPU Board and partially pull it forward. Unplug the three ribbon connectors and withdraw the CPU Board.
~
P0000001.pic
blank
nil
Service
Strip Chart Recorder
nil

Check the fuse on the CPU Board and replace it if it is bad.
%

VITA

Eva R. Broadaway was born on May 25, 1956 in Jonesboro, Arkansas. She graduated from Westside High School in 1974. She received a Bachelor of Science (cum laude) with a double major in Microbiology and Physical Sciences in 1977 from Mississippi University for Women in Columbus, Mississippi. In 1978, she received a diploma in Medical Technology from Baptist Memorial Hospital School of Medical Technology in Memphis, Tennessee. She holds a registry in Medical Technology from the American Society of Clinical Pathologists and is licensed in the State of Tennessee. Immediately after completing her medical technology training, she began working in the Hematology laboratory of Baptist Memorial Hospital. She continued working there and in 1979 began study toward a Bachelor of Science in Electrical Engineering at Memphis State University which she completed magna cum laude in 1983. She enrolled in the Graduate School of The University of Tennessee in 1983.

She began her engineering career in the Instrumentation and Controls division of the Oak Ridge National Laboratory in 1983, where she is currently employed. Her areas of specialization include real-time computer systems and expert systems development.

She is a member of Tau Beta Pi, the Institute of Electrical and Electronics Engineers (IEEE), and the IEEE Computer Society.