USACERL Interim Report FE-93/07
August 1993
Coal-Fired Central Heating Plant Operating System

US Army Corps
of Engineers
Construction Engineering
Research Laboratories

(2)

Development and Use of the Coal-Fired Central Energy Plant Operations Expert System (CEPES)

by
Ralph Moshage
Tony Magliero
Bob Lorand
Madhavi Kantamneni
Todd Blindt

Rising operation and maintenance (O&M) costs of central heating plants have forced the Army to seek alternative methods of running these facilities. Computer technology offers a great potential to automate and assist in many O&M tasks by helping to diagnose equipment malfunctions and failures. An automated diagnostic tool for coal-fired central heating plant equipment could reduce the demand for human labor, freeing personnel for higher priority work; reduce downtime for repair; promote thermal efficiency; and improve online reliability.

In this phase of the study, researchers began development of the Coal-Fired Central Energy Plant Operations Expert System (CEPES), which analyzes and recommends solutions to coal-fired boiler operational problems. This phase included selection of the hardware and software platforms, and development and coding of the expert system. Later phases will expand and beta test the present system, and develop a comprehensive technology transfer plan.

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products. The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

DESTROY THIS REPORT WHEN IT IS NO LONGER NEEDED

DO NOT RETURN IT TO THE ORIGINATOR

USER EVALUATION OF REPORT

REFERENCE: USACERL Interim Report (IR) 93/07, *Development and Use of the Coal-Fired Central Energy Plant Operations Expert System (CEPES)*

Please take a few minutes to answer the questions below, tear out this sheet, and return it to USACERL. As user of this report, your customer comments will provide USACERL with information essential for improving future reports.

1. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which report will be used.)

_____

_____

_____

2. How, specifically, is the report being used? (Information source, design data or procedure, management procedure, source of ideas, etc.)

_____

_____

3. Has the information in this report led to any quantitative savings as far as manhours/contract dollars saved, operating costs avoided, efficiencies achieved, etc.? If so, please elaborate.

_____

_____

4. What is your evaluation of this report in the following areas?

   a. Presentation:_____

   b. Completeness:_____

   c. Easy to Understand:_____

   d. Easy to Implement:_____

   e. Adequate Reference Material:_____

   f. Relates to Area of Interest:_____

   g. Did the report meet your expectations?_____

   h. Does the report raise unanswered questions?_____

i. General Comments. (Indicate what you think should be changed to make this report and future reports of this type more responsive to your needs, more usable, improve readability, etc.)

_____

_____

_____

_____

_____

5. If you would like to be contacted by the personnel who prepared this report to raise specific questions or discuss the topic, please fill in the following information.

Name: _____

Telephone Number: _____

Organization Address: _____

_____

_____

6. Please mail the completed form to:

Department of the Army
CONSTRUCTION ENGINEERING RESEARCH LABORATORIES
ATTN: CECER-IMT
P.O. Box 9005
Champaign, IL 61826-9005

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1993	Final

4. TITLE AND SUBTITLE
Development and Use of the Coal-Fired Central Energy Plant Operations Expert System (CEPES)

5. FUNDING NUMBERS
Reimbursable Order No. 89-AC-01

6. AUTHOR(S)
Ralph Moshage, Tony Magliero, Robert Lorand, Madhavi Kantamnemi, and Todd Blindt

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Construction Engineering Research Laboratories (USACERL)
PO Box 9005
Champaign, IL 61826-9005

8. PERFORMING ORGANIZATION REPORT NUMBER
IR-93/07

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Center for Public Works (USACPW)
ATTN: CECPW-FU-M
Bldg 358
Fort Belvoir, VA 22060-5516

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Copies are available from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Rising operation and maintenance (O&M) costs of central heating plants have forced the Army to seek alternative methods of running these facilities. Computer technology offers a great potential to automate and assist in many O&M tasks by helping to diagnose equipment malfunctions and failures. An automated diagnostic tool for coal-fired central heating plant equipment could reduce the demand for human labor, freeing personnel for higher priority work; reduce downtime for repair; promote thermal efficiency; and improve online reliability.

In this phase of the study, researchers began development of the Coal-Fired Central Energy Plant Operations Expert System (CEPES), which analyzes and recommends solutions to coal-fired boiler operational problems. This phase included selection of the hardware and software platforms, and development and coding of the expert system. Later phases will expand and beta test the present system, and develop a comprehensive technology transfer plan.

14. SUBJECT TERMS
central heating plants
artificial intelligence
coal-fired energy plant operations expert system (CEPES)
boilers

15. NUMBER OF PAGES
66

16. PRICE CODE

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

FOREWORD

This study was performed for the U.S. Army Center for Public Works (USACPW) under TACOM reimbursable order No. 89-AC-01, "Coal-Fired Central Heating Plant Operating Expert System." The technical monitor was James Donnelly, CECPW-FU-M.

This research was conducted by the Energy and Utility Systems Division (FE), Infrastructure Laboratory (FL), U.S. Army Construction Engineering Research Laboratories (USACERL). The USACERL principal investigator was Ralph Moshage. Dr. David Joncich is Division Chief, CECER-FE, and Dr. Michael J. O'Connor is Laboratory Chief, CECER-FL. Special thanks is owed to Kristi Nies, CECER-FE, for her assistance in preparing this report. Appreciation is also expressed to Charles Schmidt, of Schmidt and Associates, and Thomas Parsons, of Science Applications International Corporation (SAIC), for providing expert knowledge engineering information. Robert Lorand and Tony Magliero are also associated with SAIC. The USACERL technical editor was William J. Wolfe, Information Management Office.

LTC David J. Rehbein is Commander of USACERL, and Dr. L.R. Shaffer is Director.

CONTENTS

CONTENTS (Cont'd)

FIGURES

TABLES

DEVELOPMENT AND USE OF THE COAL-FIRED CENTRAL ENERGY PLANT OPERATIONS EXPERT SYSTEM (CEPES)

1 INTRODUCTION

Background

Army installations operate and maintain a large number of fossil fuel powered central energy plants, some of which use coal as a primary fuel. A general problem with coal-fired central energy plants is that the mechanisms involved in the coal combustion are so complex that plant personnel sometimes find it hard to quantify the impact of operational parameters on the boiler plant. This is complicated by the fact that coal is a nonhomogeneous fuel, which means that critical fuel specification parameters may fluctuate even within a single shipment of coal. As yet, there is no automated tool to analyze and solve coal-fired boiler operational problems. This technology gap can result in reduced boiler plant efficiency, shorter equipment life, reduced plant reliability, and the perception that coal-fired boilers are difficult to operate.

There is a need for a system to bridge this gap, to help heating plant personnel analyze and solve coal-fired boiler operational problems. Such a system should also give users appropriate technical background information on coal combustion processes.

Objective

The overall objective of this project is to develop an expert system to help personnel at coal-fired central heating plants analyze and solve boiler operational problems. This system should provide technical support for evaluating and solving fuel and equipment-based problems, and problems resulting from incorrect operating procedures. This system should also give plant personnel appropriate technical information on coal combustion processes.

The first phase of work, covered in this report, involved: (1) selection of the hardware and software platform; (2) development of the expert system structure; (3) investigation of knowledge engineering; and (4) coding of the expert system.

The second phase of this project will involve: (1) continued development of the expert system through the development of a CEPES System Editor; (2) expansion of the knowledge base to include additional coal combustion technologies; (3) detailed beta testing of the completed program; and (4) development of a comprehensive technology transfer plan.

Approach

A personal computer (PC) hardware platform was chosen for the CEPES software based on the relatively low cost, good performance, and common availability of these systems. A recommended minimum hardware configuration was specified.

Since CEPES is a relatively small project, it was considered preferable to use an "expert system tool" rather than to develop source code "from scratch." A number of commercially available expert

system tools were evaluated and one system was selected based on its "smartness," performance capabilities, user interfaces, and ability to communicate with other software.

Of the two principal diagnostic techniques commonly used in artificial intelligence ("tracing" and "branching" logic), tracing logic was chosen as the strategy to apply in creating CEPES. CEPES software was developed incrementally; that is to say, that modules developed and tested separately were later coordinated into a single larger working unit.

2 CEPES FUNCTIONAL CONCEPTS

An expert system is a computer program that uses knowledge and inference to solve problems that would normally require human expertise. (Appendix A contains supplemental information on expert systems technology.)

The Central Energy Plant Operations Expert System (CEPES) is an expert system developed with three main objectives:

1. CEPES should be easy to use; i.e., individuals with little or no computer experience should be able to access and use CEPES with no formal training.

The system will be essentially self-explanatory, requiring only minimal instruction on the use of the mouse pointing device. Illustrations and ASCII text files will be used to describe the boiler plant equipment. Schematics of the plant will be used as much as possible to communicate with the operators. Data entry should be flexible; the expert system will not require data to be entered in any specific order.

2. Plant engineers (not plant operators) will be assigned the task of maintaining and updating CEPES.

The ability of engineers to revise CEPES will be enhanced by the use of external ASCII and bitmap files that CEPES will load at runtime. Thus, engineers can change CEPES messages and advice by editing the ASCII text files or by updating bitmap illustrations.

3. CEPES should address a "generic" coal combustion technology type of boiler plant so that users can customize the program for their own particular plant configuration.

In CEPES the spreader stoker plant is broken down into five systems (Figure 1):

1. Boiler
2. Coal Handling
3. Pollution Control
4. Solid Waste
5. Water Treatment.

Each of these systems is further broken down into subsystems such as:

System	Subsystem
Coal Handling	Coal Bunker
Solid Waste	Fly Ash Handling
Pollution Control	Multicyclone

The introductory screen for CEPES (Figure 2) is a modified schematic that displays a complete set of spreader stoker-fired boiler system and subsystems. For each subsystem, CEPES lists a number of problems associated with that subsystem. All systems, subsystems, and problems are represented by mouse-sensitive rectangular images. By positioning the mouse pointer over a rectangle and clicking the right mouse button, users can access the appropriate information.

Figure 3 shows the screen for the **CoalHandling** system. Figure 4 shows the screen for the bucket conveyor **Subsystem**. The right side blocks identify the **Subsystem** problems. The left side blocks show specific troubleshooting tests that can be made regarding each **Subsystem**. It is assumed that the operators will first notice the problem and then need a checklist of the diagnostic tests to be made. CEPES does

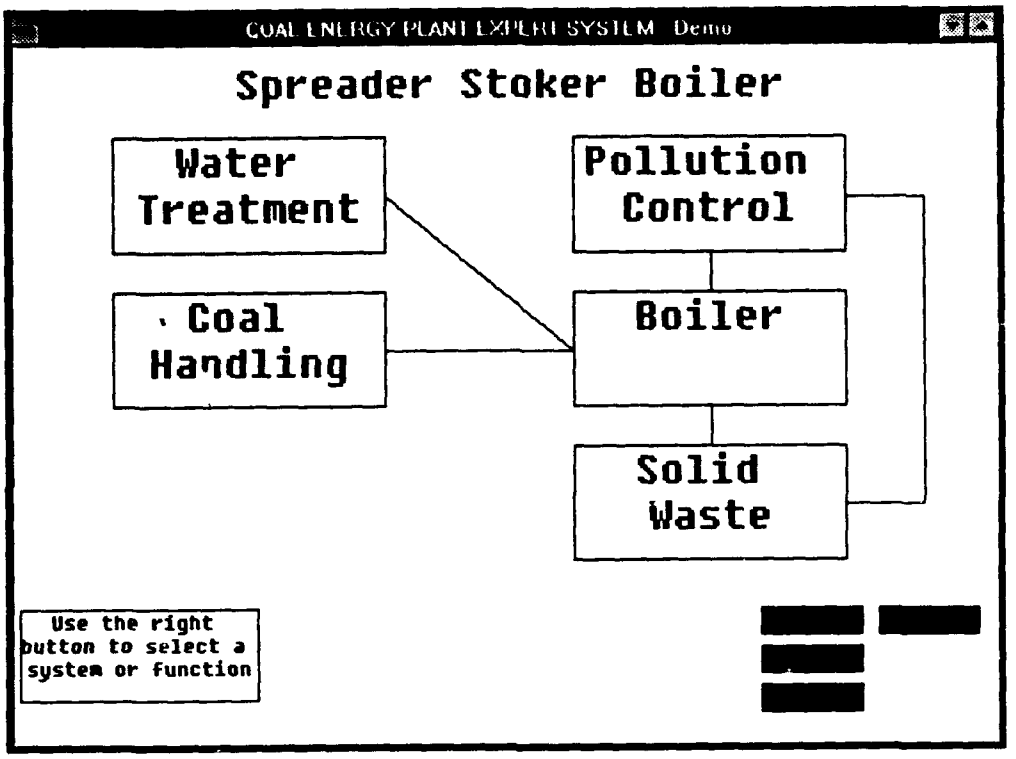

Figure 1. System Menu for Spreader Stoker Boiler.

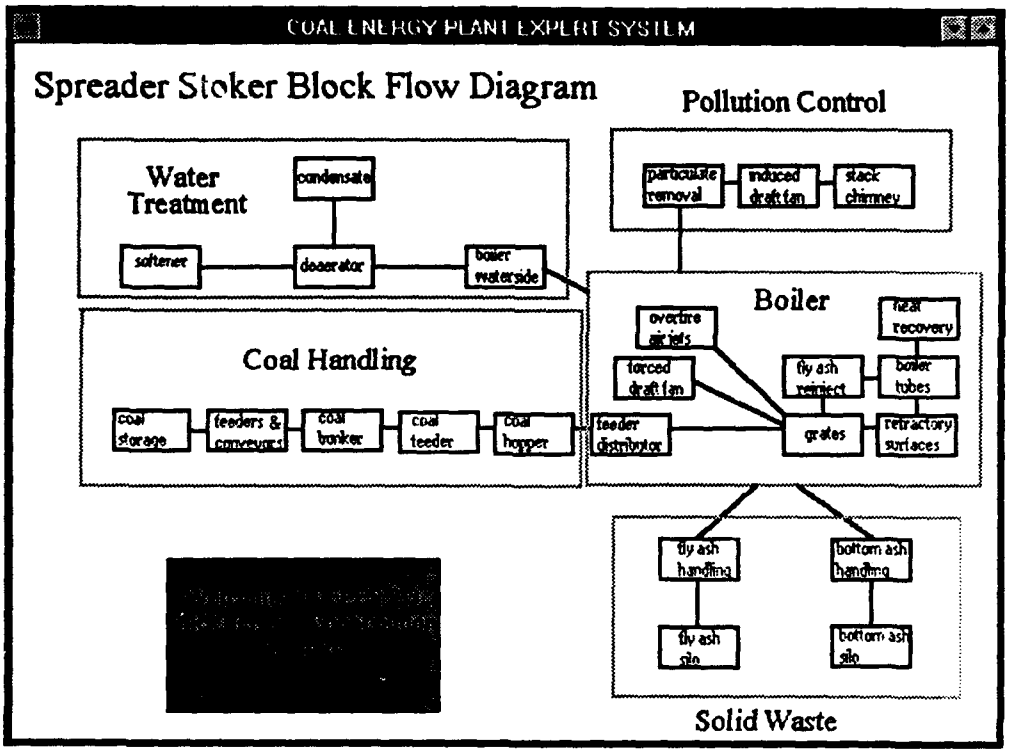

Figure 2. Introductory Screen.

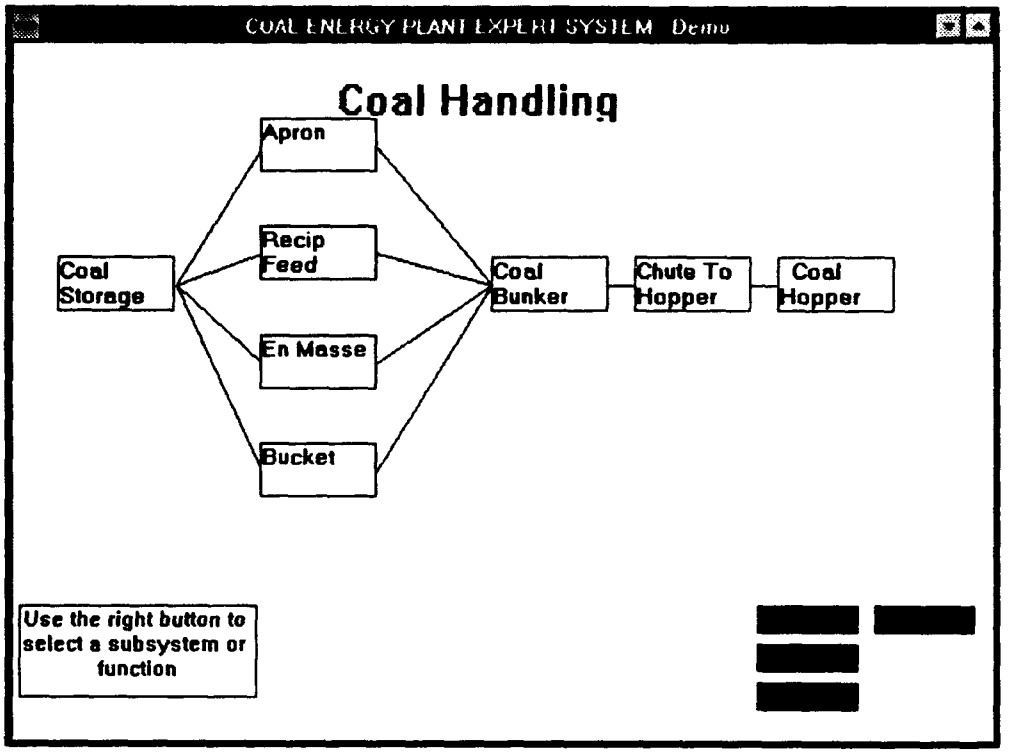

Figure 3. Coal-Handling System.

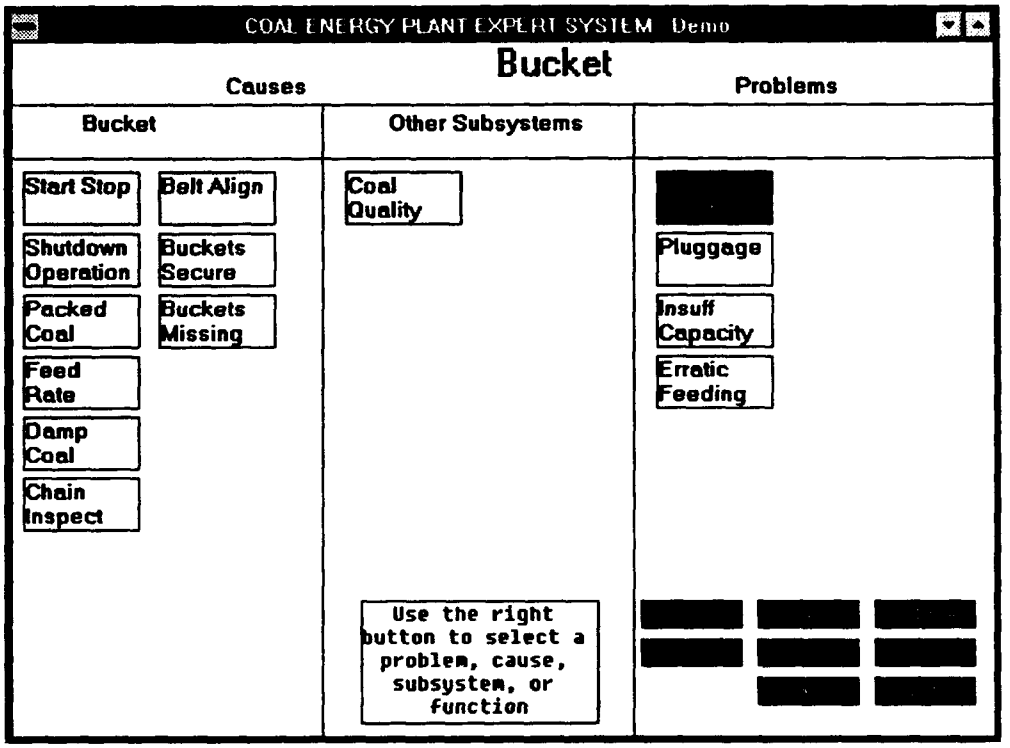

Figure 4. Bucket Conveyor Subsystem.

not make this assumption. Users may enter data in any order they wish. For example, operators may first notice insufficient capacity in the bucket conveyor or they may first notice the presence of damp coal. If insufficient capacity is entered first, CEPES will direct the user to check for the presence of damp coal (among other things). If damp coal is entered first, the user is warned that damp coal may lead to insufficient capacity.

In many situations, finding a problem will require the operator to inspect more than one **Subsystem**. Therefore, CEPES will allow the operators to easily move from one **Subsystem** to another. This capability is provided by the center column in Figure 4, which displays other **Subsystems** that may cause the problems shown on the rightmost column. By selecting these images with the mouse, the user can quickly access these **Subsystems**.

The three-column format shown in Figure 4 is used for all **Subsystems**. When a specific problem is selected from the **Bucket Conveyor** screen shown in Figure 4, a screen showing information about that particular problem is displayed. In addition to **Subsystems** and problems, Figure 4 also shows function buttons in the lower right hand corner. These buttons are mouse-sensitive and provide access to other utilities and functions.

All subsystems and problems in CEPES use color codes to indicate their status. Initially, everything is white. When the user indicates a problem, the associated rectangle becomes red and other subsystems/problems that should be investigated become yellow. These yellow rectangles represent subsystems and other problems that should be investigated. By selecting the yellow rectangles, the user can track down all of the possible causes of a problem. When a problem is resolved, the rectangle becomes green.

Figure 5 shows the problem description and causes screen for **BucketInsuffCap** (Bucket Insufficient Capacity). Information about the problem is displayed in the two main text screens, and function buttons **TOOLS** and **DRAWING** are available to assist in program diagnosis.

The function buttons act as the menu system for the CEPES program. For example, when you select the **TOOLS** function button in the boiler operation screen, an efficiency calculator pops up (Figure 6) to calculate the boiler efficiency based on user-entered input values. Enter the input values by using the "sliders," which keep user input within the appropriate range.

When the user selects the **PRIORITY** function button, it presents a menu of the problems associated with the activated **Subsystem** on the current screen (Figure 7). When the user selects the **DRAWING** button, a screen with the appropriate schematic is displayed (Figure 8).

When the user selects the **HISTORY** button, a pop-up window appears that lists all the previously accessed **Systems** and **Subsystems**. You can access any of the **System** or **Subsystem** screens at any time either from the **history** pop-up window (Figure 9) or from the System Menu Screen (Figure 1).

The user can get on-line help by clicking on the **HELP** button (Figure 10). The user can also start a new session or quit CEPES at any time by choosing either the **CLEAR** or **QUIT** buttons (Figures 11 and 12). Chapter 8 describes these features in more detail.

CEPES provides a final report summary of each session, which is stored in an ASCII file for future analysis. The report includes a listing of all of the observations entered by the user, all repair actions reported by the user, and all conclusions reached by CEPES.

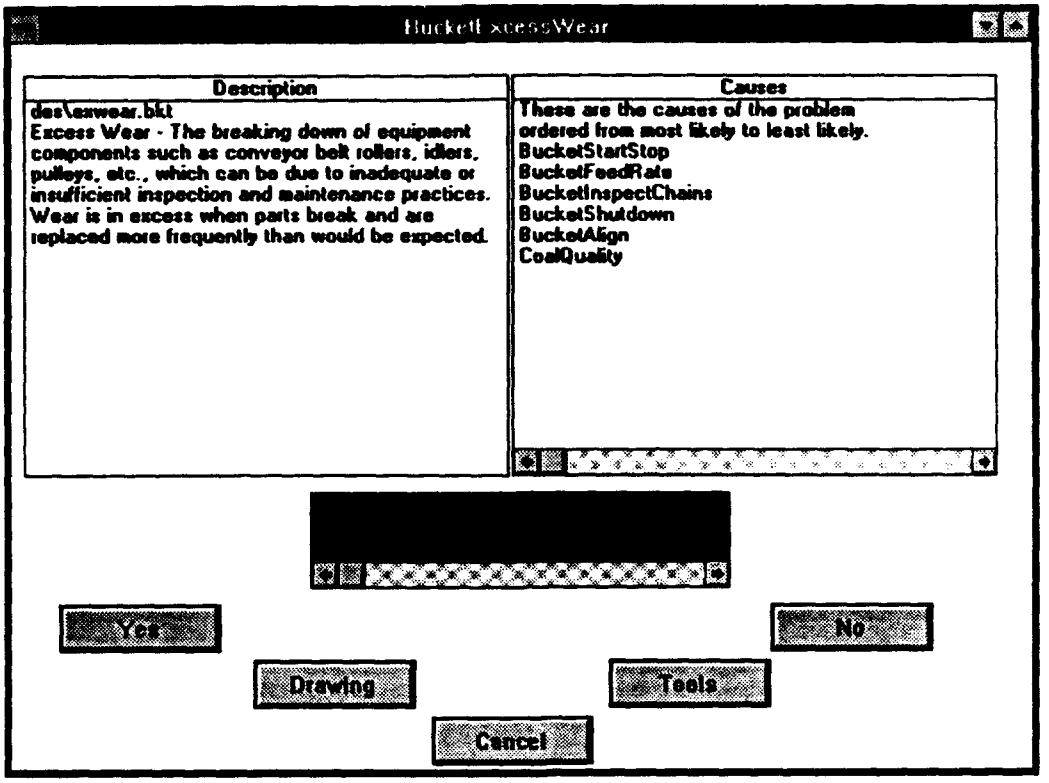

Figure 5. Insufficient Capacity: Bucket Conveyor Subsystem.

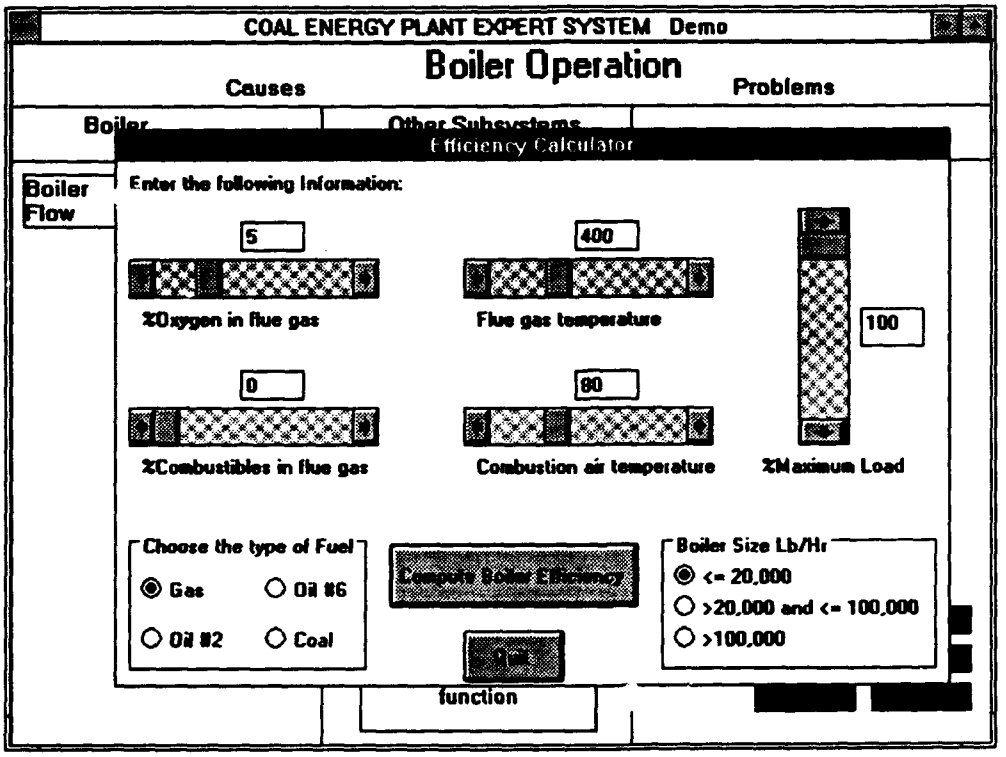

Figure 6. TOOLS Function Example.

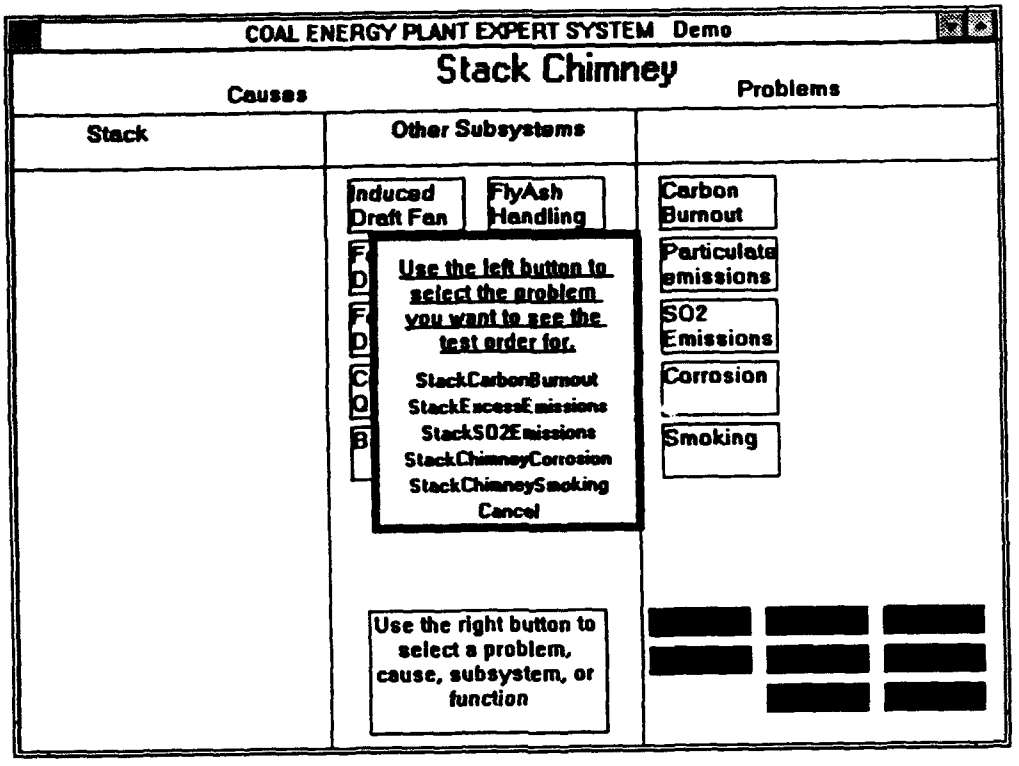

Figure 7. PRIORITY Function Example.

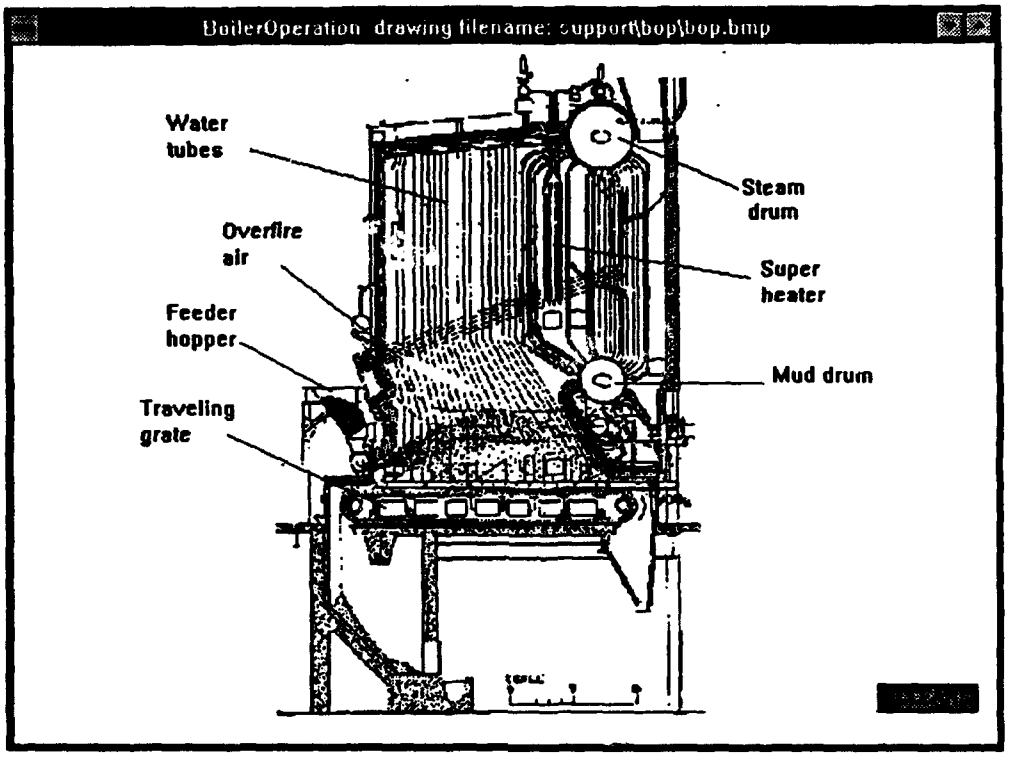

Figure 8. DRAWING Function Example.

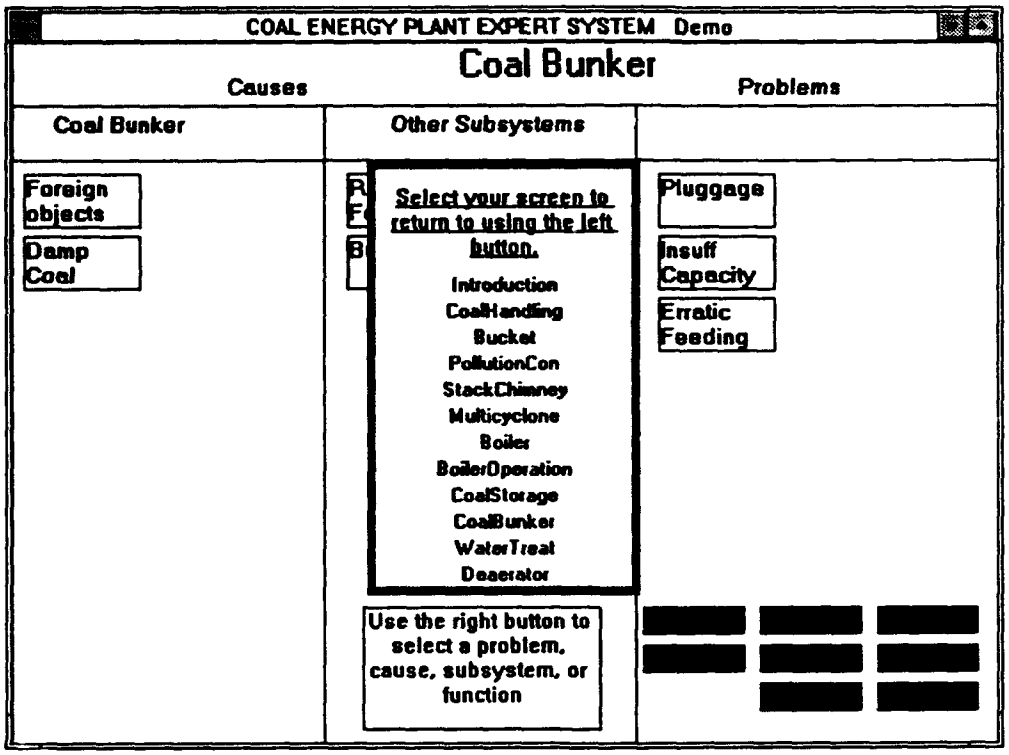

Figure 9. HISTORY Function Example.

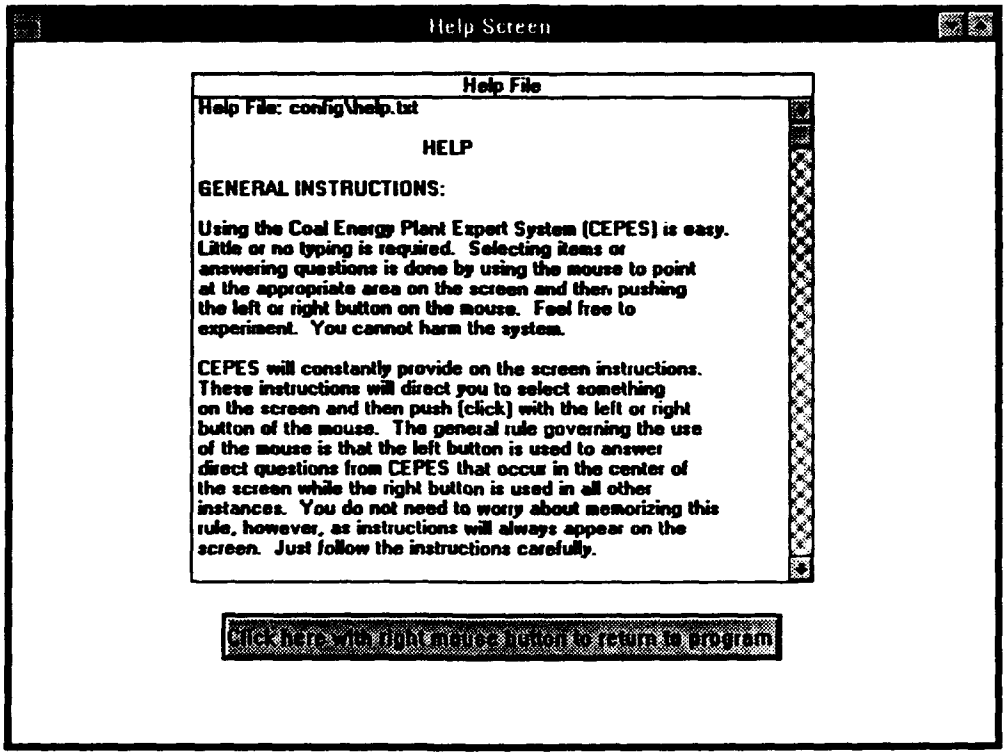

Figure 10. HELP Function Example.

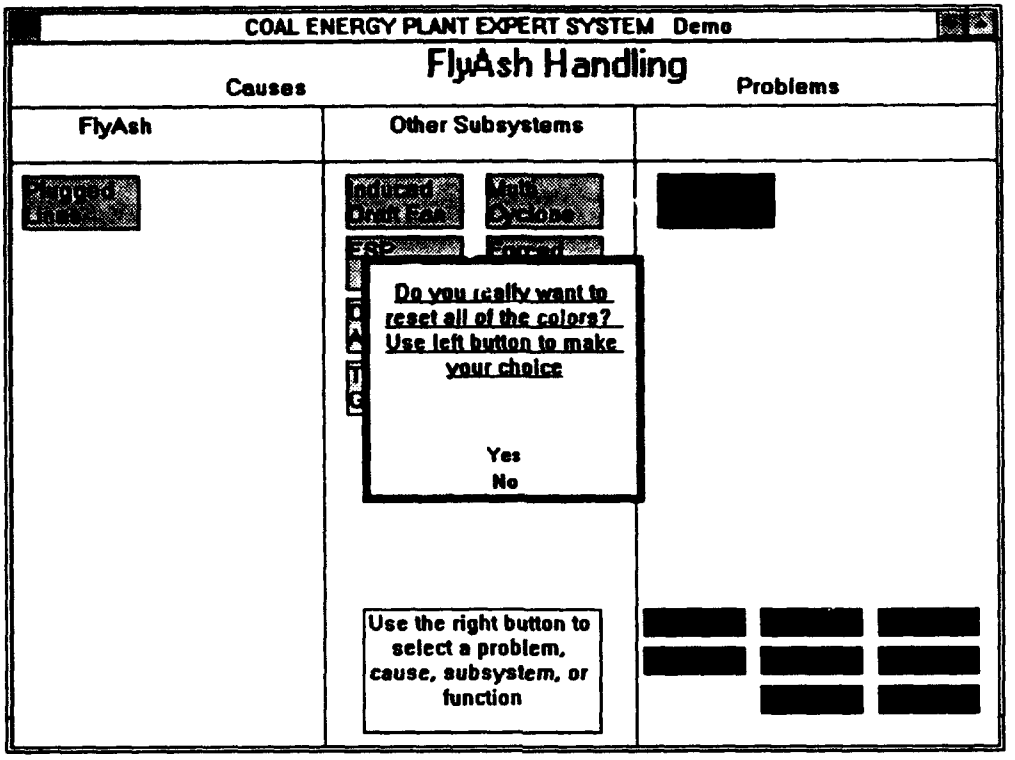

Figure 11. CLEAR Function Example.

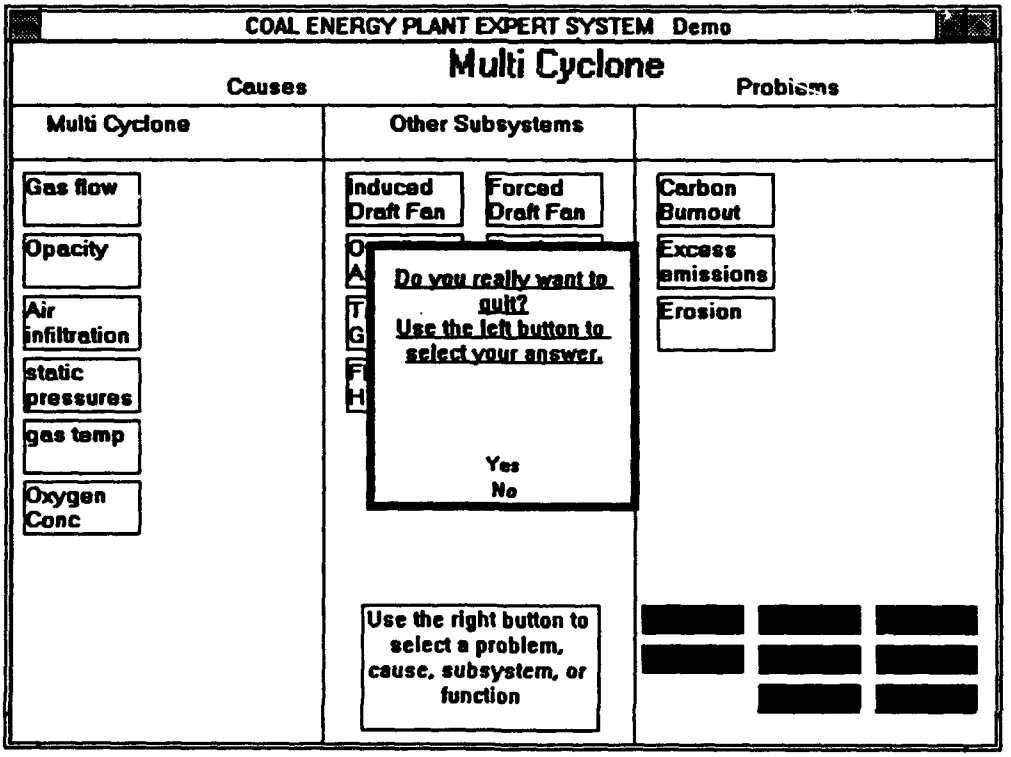

Figure 12. QUIT Function Example.

3 CEPES ENVIRONMENT

Hardware

The capabilities of PCs have grown dramatically in recent years. Machines based on the Intel 80486 chip have a capability equivalent of the workstations of 5 years ago that cost 10 times their current $2000 price tag. The use of Microsoft® Windows™ also allows these machines to access the large amounts of random access memory (RAM) necessary to run graphic interfaces and multi-tasked programs.

For these reasons, the PC environment was selected to host CEPES. At a minimum, a 80286 PC with 4 Mb of RAM and 20 Mb of free space on the hard disk is required. The PC will also need to have VGA color graphics capabilities and a two-button mouse. With the 80286 chip, the performance of CEPES will be quite slow and frustrating. With 4 Mb of RAM, it is also possible that memory will be exhausted if a large number of **Subsystems** are examined (**Subsystem** information is loaded only as it is needed).

A recommended configuration for CEPES would consist of a 80386 PC platform with 8 Mb of RAM and 30 Mb of free hard disk space. The minimum requirement for VGA color and a two button mouse remains unchanged.

Software

Along with the dramatic increase in the capabilities of PC hardware, there has been a large increase in the number and capability of software packages that could support this project. Because CEPES was a relatively small project, it was not practical to develop a large amount of source code in a "native" programming language such as "C," Pascal, or Ada. These efforts are always labor intensive.

A more efficient strategy was to use a currently available expert system building tool. These tools are available for PC applications at prices ranging from $100 to $5000. Since a good tool can increase programmer productivity dramatically, even the most expensive tool can save the project considerable resources. It was therefore decided to purchase a commercial expert system building tool to construct CEPES. Table 1 lists the candidate systems considered.

Each of these expert system tools was examined based on four broad criteria:

1. Ability to be "smart." An expert system tool should provide some standard reasoning capabilities:

- Data driven and goal driven processing. This provides maximum flexibility to the developers and makes the system easy to change.
- Object-oriented processing. This would be very useful because it would allow building systems that can be adapted to different plant configurations easily.
- An agenda mechanism with a procedural language. Again, this provides maximum flexibility and makes the system easy to change.

Table 1

Candidate Systems

Tool Name	Vendor
Knowledge Engineering System	Template Software
Nexpert Object	Neuron Data
Level 5 Object	Information Builders
Aion Development System	Aion
Kappa	Intellicorp

2. Performance. CEPES contains about 400 programming objects (when loaded with every subsystem). Each object has an average of 15 slots (attributes) attached to it. A tool should be able to support a program of this magnitude and still perform with little or no detectable delay.

3. User interface. There are wide differences among the products. The criteria used to judge the products were:

- Not acceptable: Only vendor-defined screens and formats are provided. (The user must learn and memorize how to use predefined screens.)
- Barely acceptable: The developer can format text-based screens and design the users' interactions with them.
- Much, much better: The developer can format text and graphics-based screens and design the users' interactions with them. The screens can allow for mouse or keyboard entry of data.
- Absolutely great: The developer can format both text and graphics-based screens, and screens based on scanned images. These screens can be used for mouse or keyboard entry of data.

4. Communications with other software. Some of the tools come with ready-made hooks to programs written in other languages, such as "C." This can be extremely useful because the tools seldom provide everything you will need.

Based on these factors, the Kappa tool of Intellicorp (GSA price: $2500) was selected. The remaining tools were all less developed in one or more areas. Level5 Object was found to be relatively slow even while operating on small systems. Nexpert Object was found to support fewer different problem solving approaches than competing programs. The Knowledge Engineering System and the Aion Development System both provided fixed runtime screens (but neither could be customized). Finally, three of the tools (Knowledge Engineering System, Aion Development System, Nexpert Object runtime system) were not Windows™ compatible. Since they are therefore limited to addressing only 640 Kb, they may not be able to support a large expert system.

Kappa scored well on all of the four points. It provided a number of problem approaches (data-driven reasoning, goal-driven reasoning, object-oriented processing) as well as common procedural code programming constructs (if–then–else, for loops, function calls). Kappa runs under Windows™, which allows it to access large amounts of RAM, and to provide flexible user interfaces with customized, mouse-sensitive images. Also, Kappa is written in "C" and performs quite well. It can also invoke and communicate with any other application programs running under Windows™.

4 CEPES DEVELOPMENT APPROACH

Any expert system development project involves two phases: (1) knowledge engineering (interviewing experts and reading subject matter documentation), and (2) software engineering (converting what is learned into an executable code).

The two activities occur concurrently—knowledge that is gained is quickly converted into software. Thus, there is constant switching back and forth between the two activities, often on a daily basis. The CEPES project used this "build as you go" philosophy to quickly uncover errors or inadequacies. The following sections describe this approach to each of these two phases.

Knowledge Engineering

The driving force behind the design and development of the software was the domain knowledge gained by the project programmers. The program depended heavily on written documentation, primarily tables and figures, to minimize the demands on utility engineers. These sources were used to convey the propagation of failure modes through the plant (by causal chains) and/or the structural/functional layout of the elements (subsystems) of the plant. For each plant element and associated failure modes, the program should list some corrective action.

Two diagnostic techniques used in artificial intelligence—tracing and branching logic—were applied. (In the AI literature, these techniques are more commonly known as model-based and rule-based reasoning). With tracing logic, the expert system begins with the problem (an observation of an incorrect parameter or plant operating condition) and then traces the possible causes of the fault backwards or forward through the plant subsystems. With a branching logic, a series of "if–then" rules are applied to a particular problem or inappropriate condition. These rules may create a simple checklist, or they may encapsulate a more complicated multi-step branching logic.

In CEPES, a model-based tracing logic was used to track the causes and effects of incorrect parameters (problems) throughout the plant. This tracing is done by examining the connections that define a network of subsystems and possible problems.

CEPES can ask for and accept many observations and tests about the condition of the utility. The procedures and steps that each test requires are specified in external ASCII text files and bitmap files that can be updated by plant engineers. Thus, CEPES provides a structure that can be updated with additional technical knowledge without modifying the CEPES software itself.

The first phase of development emphasized a breadth of analysis rather than depth to ensure that all major subsystems were covered by the CEPES, but within resource constraints. Additional quantitative data—primarily appropriate ranges for parameters such as temperatures, pressures, oxygen concentrations, etc.—can be added as needed for particular subsystems. CEPES also contains provisions for future development of economic or other executable subroutines for each **Subsystem** and problem.

Software Development

The approach used to develop CEPES software is known as "incremental development." This involves several modern software engineering techniques, such as object-oriented development and information hiding.

As the name implies, "incremental development" involves the creation of software modules that work together in a single, larger unit. Code development proceeded with frequent testing and evaluations. A working program was almost always available to test and gather feedback. This way, problems in newly introduced material could be identified and corrected immediately.

Object-oriented development and information hiding are software development practices and standards closely aligned with incremental development. Technical detail behind these concepts will not be provided here, but it is sufficient to say that in these methodologies, bodies of software are broken into modules that are designed to meet two criteria:

1. The independence among the modules is maximized so changes in one module do not affect changes in another. (This greatly facilitates bug fixing.)

2. The modules are understandable. Each module represents a physical or conceptual entity in the application, so the meaning and importance of each module is easy to grasp.

These practices allow incremental development by adding and fitting new modules together. Encountering difficult modules is not fatal to the project because they can be surgically removed without crippling the software.

CEPES currently does not use true object-oriented message passing for communication among the **Subsystems** and problems. To implement object-oriented message handling, all **Subsystems** and problems that represent a particular utility configuration need to be resident in memory at the same time. This would result in almost 500 objects being loaded into memory.

Pilot testing with an 80386 computer (with 8 Mb RAM and loaded with the Kappa development system) shows that approximately 25 subsystems can be loaded before memory is exhausted and the system crashes. To maximize performance, the **Subsystem** objects are currently loaded only when the user needs to access a particular **Subsystem**. With only the Kappa runtime system loaded, it is unlikely that an engineer would need to access enough **Subsystems** to exhaust the computer's memory.

5 CEPES DIRECTORY STRUCTURE

The CEPES diskettes contain several directories of program and data files. The CEPES root directory contains two files: **cepes.bin** and **cepes.kal.** The **cepes.bin** file contains the binary code executed by the CEPES application. The **cepes.kal** contains the source code from which the **cepes.bin** file was generated. If the source code in the **cepes.kal** file is changed, the **cepes.bin** file must be regenerated.

Beneath the main CEPES directory are the **config, instance, cor, des, coal, support,** and **history** subdirectories, each of which is described below.

Config Subdirectory

The **config** subdirectory contains four files (systems.kal, name.kal, intro.txt, and help.txt) and will also contain one file for each utility configuration generated by CEPES users. The **systems.kal** file contains the Kappa definition of all of the possible **Subsystems** within a coal energy plant. This file is loaded whenever a new configuration is to be built. A new utility configuration is constructed by selecting the appropriate **Subsystem** from the **systems.kal** file. The configuration is now stored in this directory as ***.kal**, where * is a user-defined name.

Each subsystem defined in **systems.kal** contains a unique **ThreeLetterCode**. The **ThreeLetter-Codes** are listed in Appendix B. They are used repeatedly to generate the names of ASCII files, bitmap files, and executable programs that are integrated into CEPES.

An index to the utility configurations is provided in the **names.kal** file. Whenever a new configuration is made, the new configuration name is added to this file. When an old configuration is to be used, the **names.kal** file is used to generate the menu choices.

Also in the **config directory** are two ASCII text files that are displayed to the user. The **intro.txt** file contains the text instructions presented to users early in the session. The **help.txt** file contains the help message that can be displayed by users at any time. These files can be edited with any ASCII text editor.

Instance Directory

The instance directory contains a ***.kal** file for each defined subsystem in CEPES. The names of these files are listed in Appendix B. These files contain five major categories of information:

1. A Kappa definition of the problems for each subsystem

2. The Kappa specification of the causes of each problem

3. The Kappa definition of the function to be executed whenever the problems of this subsystem are to be displayed

4. The Kappa definition of the function to be executed whenever the problems of the subsystem are to be hidden

5. The Kappa definition of the drawings that could be associated with the subsystem and with the problems belonging to that subsystem.

Each problem defined in the ***.kal** files has an associated **FileCode**–a descriptive name of eight or fewer letters. Unlike the **Subsystem** three letter codes, the **FileCodes** are not unique. For example, "pluggage" in all of the **Subsystems** has the **FileCode** "pluggage." Like the **Subsystem** three letter codes, however, the **FileCode** is used to generate names for the ASCII, bitmap, and executable files integrated into CEPES. By combining the **FileCode** and the **Subsystem ThreeLetterCodes**, unique names can be generated. (See Appendix C for an example of an instance file.)

Cor Subdirectory

This directory contains **correction files**, which are ASCII text files that define remedial actions. Each problem defined in the many **Subsystem** files of the instance directory contains a slot called a **CorrectionFile**. This value of this slot is set to the name of a text file that describes the remedial actions associated with the problem.

All **correction files** are in this directory and all are named ***.#**, where * is the **FileCode name** (of eight or fewer characters) for the problem, and # is a three letter code assigned to the **Subsystem** to which the problem belongs. Each problem has a unique correction file. All can be displayed by the end user. (Appendix B lists these codes.)

The correction files for coal quality related problems are not named as described above, nor are they stored in the **cor** directory. The procedures for coal quality corrections are described later in this chapter.

Des Subdirectory

This directory contains **description files**, which are ASCII text files that provide descriptions of problems. Each problem defined in the many subsystem files of the instance directory contains a slot called a **DescriptionFile**. The value of this slot is set to the name of a text file that provides a description of the problem.

All description files are in this directory and all are named ***.#**, where * is the problem **FileCode** and # is a three letter code assigned to the subsystem to which the problems belongs. Each problem has a unique description file, and all can be displayed by the end user. (Appendix B lists these codes.)

The description files for coal quality related problems are not named as described above, nor are they stored in the **des** directory. The procedures for coal quality descriptions are described later in this chapter.

Support Subdirectory

Each **Subsystem** and problem in CEPES can have a bitmap drawing and an executable function (**TOOLS**) associated with it. For **Subsystems**, bitmap file names are **support\#\#.bmp** and executable function file names are **support\#\#.exe**, where # is the **Subsystem ThreeLetterCode**. For problems, bitmap file names are **support\#*.bmp** and executable function file names are **support\#*.exe**, where

represents the **ThreeLetterCode** for the **Subsystem** to which the problem belongs and * is the **FileCode** for the problem.

Again, the executables and bitmaps for coal quality related problems are not stored in the **support** subdirectory, but are described below.

Coal Subdirectory

The **CoalQuality Subsystem** is a special case in CEPES, not handled like the other **Subsystems**. Information about **CoalQuality** problems needs to be customized for each of the other **Subsystems**. Therefore, the filenames for **CoalQuality** information use the name of the operational **Subsystem** from which **CoalQuality** was derived. The file names are:

description file: coal\#*.des
correction file: coal\#*.des
bitmap file: coal\#*.bmp
executable file: coal\#*.exe

where # is the **ThreeLetterCode** of the coal quality initiating subsystem and * is the **FileCode** for the **CoalQuality** problem. For example, if the user selects **CoalQuality** from the **CoalBunker** screen (**ThreeLetterCode for CoalBunker is cbk**) and then selects **CoalQualityAsh (FileCode = ash)**, the description file is **coal\cbk\ash.des** and the correction file is **coal\ckb\ash.cor**.

History Subdirectory

Each time CEPES is invoked, a file named ***.his** (where * is a user-supplied name or a default name provided by CEPES) is generated and stored in this directory. This file keeps a history of all problems reported to be present or absent, and all corrective actions reported by the users. These are ASCII text files that may be viewed or printed with any text editor.

6 CEPES CODE SUMMARY

The CEPES source code can be broken into two main components: functions and classes.

Functions

CEPES functions are similar to functions in any program language, i.e., they invoke each other and pass arguments (Appendix D). A most useful feature in the Kappa programming language is its ability to attach a function to an image on the screen. This function is then executed whenever the image is selected with the right mouse button. This feature provides much of the CEPES capability to move among subsystems and problems.

Two important functions in CEPES are called **SubSystemRightAction** and **ProblemRightAction**. The user invokes these functions by selecting a subsystem image or a problem image, respectively. **SubsystemRightAction** essentially performs only two actions. It will:

1. Check to see if the **Subsystem** information (from the instance directory) has already been loaded. If not, it will then read it in.

2. Execute the ***Init** function (where * is the **Subsystem** name). This ***Init** function is stored in the **Subsystem** file from the **instance** subdirectory.

After the **SubSystemRightAction** function is executed, the user sees the three column format with images of subsystem problems showing on the left and right columns. Selecting any of these problems will execute the **ProblemAction** function.

The **ProblemAction** function is much more complicated than the **SubSystemRightAction** function. It is important to note here that all problems are classified as causes or effects by this function, depending on what is stored in the instance directory. The following steps describe the **ProblemAction** function.

ProblemAction will:

1. Activate a new window and place it over the previous window. This will serve as a background for the upcoming images.

2. Show the description text associated with this problem in the description view window on the left side of the screen. If no description file is found, then "Description file not found" is displayed.

3. Check to see if the problem to be displayed is an effect or a cause. This is done by checking the length of the **Causes** slot and the length of the **IsCausedBy** slot. For an effect, the former is zero and the latter is greater than zero. For a cause, the opposite is true. Further processing by **ProblemAction** depends upon the outcome of this test.

If the outcome is an **effect, ProblemAction** will:

4. Show the description view window on the right side of the screen and display the description associated with this effect. If no description file is present, then "Description file not found" is displayed.

5. Show the image with instructions. ("Do you see the problem described here? Click on the right answer with your right button.")

6. Show the user-activated buttons for responding to the instructions:

A. "Problem is present" button.
B. "Problem is absent" button.
C. "Show drawing of problem" button.
D. Cancel out of problem screen and return to subsystems.

If the outcome is a cause, **ProblemAction** will:

4. Show the correction view window on the right side of the screen and display the corrective procedure associated with this cause. If no correction file is present, then "Correction file not found" is displayed.

5. Show the image with the instructions: "What is the status of your (problem name)? Select your answer with the right button."

6. Show the user-activated buttons for responding to the instructions:

A. Problem is already with specification
B. Problem has just been put within specification by corrective action
C. Problem is not within specification
D. Show drawing of the Problem
E. Problem evaluation utility with **Tools**
F. Cancel out of problem screen and return to Subsystems.

There is one other function in CEPES worth special mention. The **RunApplication** function is automatically executed whenever CEPES is loaded. **RunApplication** serves as the function that always initializes CEPES. This simple function displays the introductory window that includes the CEPES high level block flow diagram and the "Click here to continue button" in the lower left corner.

Classes and Instances

The Kappa programming language provides extensive capabilities in object-oriented programming. These capabilities allow Kappa developers to define hierarchies of classes and instances. Figure 13 shows a portion of the CEPES class hierarchy. Items low in the hierarchy are specializations of items high in the hierarchy.

The top node in Figure 13 is the text node. The text class is a Kappa-provided class. Text objects are objects that can be made to appear on the screen with a title, and that can have a mouse-activated function associated with them. There are two specialization's of the text class: **Pieces and Problems.** The **Pieces** specialization is further subdivided into **Systems and SubSystems.** Beneath **Systems and SubSystems** are examples of these two subclasses. There are five example **Systems** in CEPES: **CoalHandling, WaterTreat, PollutionCon, Boiler, and SolidWaste.** For **SubSystems,** the exact number and kind of examples depend on how the utility configuration is defined. Some possible examples include **CoalStorage, FeederDistributor, Condensate, and Multicyclone.**

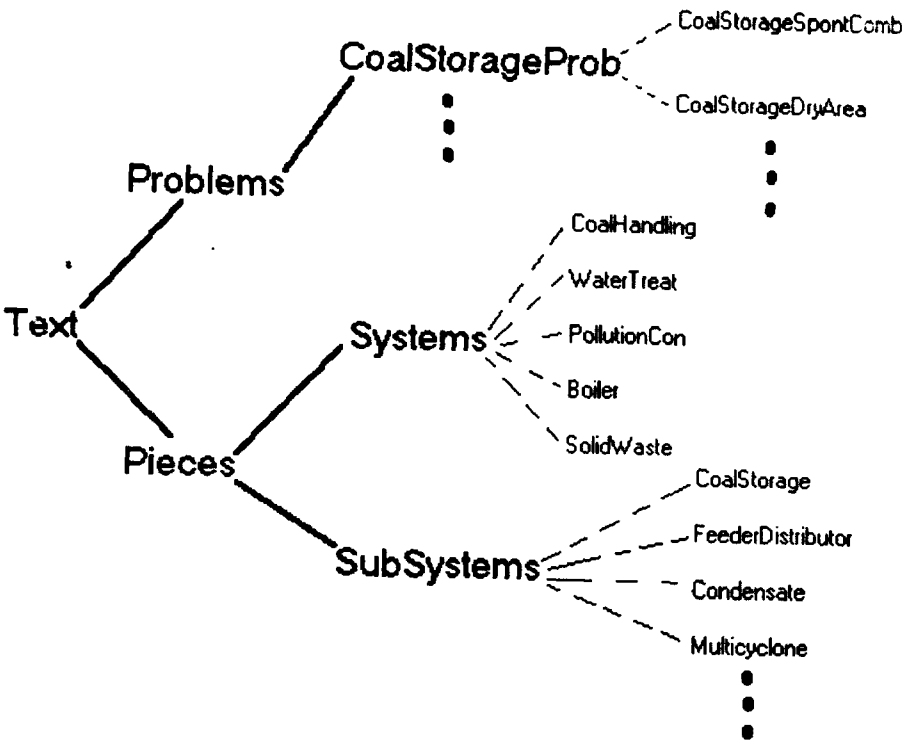

Figure 13. Class and Instance Hierarchy.

The **SubSystems** that belong in a utility configuration are defined in a file generated whenever a configuration is made. This file is a ***.kal** file in the config directory where * is a name given at configuration time by the end user. When a defined configuration is loaded, users can analyze **SubSystems** by mouse-activating them, which executes the **SubSystemRightAction** function. This function loads in the specific information associated with that **SubSystem** (stored in ***.kal in** the **instance** subdirectory, where * is a shorthand name for the **SubSystem**).

The information loaded from the instance directory is then placed under the **Problems node** in the CEPES hierarchy. This is done by creating a specialization of Problems titled ***Prob**, where * is the name of a **SubSystem** and placing the examples of problems for * under that node. Figure 13 shows an example with a node entitled **CoalStorageProb** and example problems known as **CoalStorageSpontComb** and **CoalStorageDryArea**.

Figure 13 shows only a small portion of the total CEPES hierarchy. Most of the other nodes in the hierarchy specify headings, titles, and instructional messages. They are not relevant for understanding the fundamental logic of CEPES.

One of the advantages of this hierarchical approach is that functions can be made to apply to any level in the hierarchy. CEPES contains many functions that apply to **SubSystems** in general, to **Problems** in general, and to specific **SubSystems**. This capability to write general or specific functions reduces the amount of code to be written.

7 UPDATING CEPES

Adding functionality to CEPES can be tedious. Most types of changes should not be attempted by anyone without considerable experience with the Kappa programming language, called KAL. Making minor changes to the logic within the current functionality is feasible, however, if carefully done.

CEPES loads many files describing the subsystems and potential problems. Each file refers to **Subsystem** names and problem names in other files. The Kappa tool with which CEPES is built is case sensitive, and a spelling or case mismatch will lead to an error message. Thus, after an edit such as changing the name of the problem in one **Subsystem**, all of the other **Subsystem** files (in the instance directory) must be checked for references to that **Subsystem** problem.

One type of change is fairly easy to make, however, because it can be localized to one file in the **instance** directory. If an effect and a cause are both in the same **Subsystem**, then it is simple to link them (if they are not already linked) or de-link them (if they are already linked). In other words, given that a right side column problem and a left side column problem (a cause) are both defined in the same **Subsystem**, it is easy to connect or disconnect them.

For a particular effect, the **IsCausedBy** slot lists all of the problems in the same **Subsystem** that could cause the effect. (The **IsCausedBy** slot also contains the names of other **Subsystems** where other root causes of the effect could be). Conversely, for any cause, the **Causes** slot lists all of the effects in the same **Subsystem** that this problem could cause. (The **Causes** slots are inverses of the **IsCausedBy** slots.) When an **IsCausedBy** slot is edited, the associated **Causes** slot(s) must be edited to be consistent.

Any other changes to the logic can be quite difficult. They may require changes to the ***Init** and ***DeInit** functions (where * is the name of the **Subsystem**) for one **Subsystem**, changes to the **cepes.kal** and **cepes.bin** files, and changes to multiple other **Subsystem** files in the instance directory.

The second phase of this project entails development of an automated system for editing and updating the CEPES program.

8 CEPES DRAFT USERS' GUIDE

This chapter can serve as an independent guide for end users on how to run CEPES. It is equivalent to the help file presented to end users when the help button is selected.

General Instructions

Using the Coal Energy Plant Expert System (CEPES) is easy. Little or no typing is required. Selecting items or answering questions is done by using the mouse to point at the appropriate area on the screen and then "clicking" the left or right button on the mouse.

CEPES constantly provides on-screen instructions that direct you to select something on the screen and then push (click) with the left or right button of the mouse. In general, the left mouse button is used to answer direct questions from CEPES that appear in the center of the screen, and the right button is used in all other instances. You do not need to worry about memorizing this rule, however, as instructions will always appear on the screen. Just follow the instructions carefully.

Configurations

CEPES allows you to create and use different central energy plant configurations. For example, some plants may use a vibrating feeder or a screw feeder, or some may not have an electrostatic precipitator. Each configuration must have a name given to it by the person defining the configuration. Each time you use CEPES, it will begin by asking you whether you want to use an old configuration or define a new one. If you select **NEW**, you will be asked questions about the new configuration and asked to name it. After naming, the configuration will be available for future use. If you select **OLD**, you will be given a menu of configuration names and asked to select one to work on.

Systems and Subsystems

In CEPES, the plant is broken down into five major **Systems**, each with a number of **Subsystems**. These **Systems** and **Subsystems** are:

1. **Coal Handling**
   - Coal Storage
   - Feeders/Conveyors
     - Apron
     - Belt
     - Bucket
     - Chute
     - Drag
     - En masse
     - Flight
     - Reciprocating
     - Vibrating
   - Raw Coal Bunker
   - Feeder
     - Belt Weightscale
     - Chute
     - WeighLarry
   - Coal Hopper

2. **Boiler**
    - Boiler Tubes
    - Boiler Operation
    - Feeder Distributor
    - Forced Draft Fan
    - Fly Ash Reinjection
    - Grates
        - Dumping grate
        - Traveling grate
        - Vibrating grate
        - Overlapping grate
    - Heat Recovery
    - Overfire Air Jets
    - Refractory Surfaces

3. **Pollution Control**
    - Baghouse (if applicable)
    - Multicyclone (if applicable)
    - Electrostatic Precipitator (if applicable)
    - Induced Draft Fan
    - Stack/Chimney

4. **Solid Waste**
    - Fly Ash Handling
    - Fly Ash Silo/unloading
    - Bottom Ash Handling
    - Bottom Ash Silo/unloading

5. **Water Treatment**
    - Boiler (water side)
    - Condensate
    - Deaerator
    - Softener

You can always move from one system or subsystem to another by selecting the item on the screen. If a desired item is not visible, click on the **HISTORY** button in the lower right hand corner of the screen. This will allow you to go to any screen you have previously used. If you want to go somewhere you have not been yet in this session, select the **INTRODUCTION screen**. This will allow you to choose from any available **System** and Subsystem.

For each **Subsystem**, CEPES engineers have defined a number of possible problems and the causes for each problem. Appendix D contains a detailed outline of the problems and their likely causes.

Data Entry and Troubleshooting

When you select a **Subsystem**, you will be presented with a three column screen format. The right side will show problems that can be observed in a **Subsystem** (e.g., erosion, pluggage). The left side will show what can go wrong (e.g., gas flow, damp coal) in this **Subsystem** that could cause the problems on the right. The middle column will show other **Subsystems** that could cause the problems listed on the right (e.g., pluggage in the coal feeder may cause pluggage in the coal hopper).

When you see something wrong in a **Subsystem**, go to that **Subsystem** and click on the name of the item. Follow the instructions on the screen and answer the question that appears in the center.

When you tell CEPES that something is wrong, it will change the color of the appropriate box to red. Other boxes that should now be checked out will be turned yellow. If you tell CEPES that something is OK or within specification, it will turn that box green. This color coding will allow you to track the possible cause of a problem, and those causes that have already been ruled out. By simply selecting the yellow colored boxes after a problem has been entered, you will quickly find the cause or causes of a problem.

CEPES can be especially helpful when a problem in one **Subsystem** can be due to either causes in the same subsystem or other subsystems. By selecting the yellow-colored items (in any order) and following the instructions on the screen, you can trace problems throughout the plant and find the cause.

Function Items

CEPES has eight function items that will be seen in the lower right hand corner, although not all eight will be on the screen at all times. All items are activated by placing the mouse pointer on the item and pushing the right side button.

QUIT - Use this item to exit CEPES. You will be asked to confirm the action when this item is selected.

HISTORY - Selecting this item causes a list of screen names to appear. These are the screens that you have used in the current session. Move to one of these screens by pointing at the screen name with the mouse and clicking the left button. This feature is especially useful when you are going back and forth between subsystems.

PRIORITY - CEPES is programmed to contain the best order for investigating the causes of a problem. To find the most likely cause of any problem, select the priority item with the right mouse button.

PROBLEM - If you are working on multiple problems, you may lose track of what has and hasn't been fixed. Selecting the Problem item (with the right mouse button) produces a list of any problems that you have entered into CEPES but have not found the cause of. Move to the screen for this problem by responding to the menu query that comes up.

CLEAR - If you wish to reset CEPES, position the mouse pointer over this item and push the right mouse button. This function item resets all red, yellow, and green color items back to white.

HELP - Selecting this item with the right mouse button produces the Help message.

DRAWING - Selecting this item with the right mouse button displays a bitmap drawing associated with the active subsystem or problem. If no drawing is available, a message is posted.

TOOLS - Choosing this item activates an external executable program associated with the active subsystem or problem.

9 CONCLUSION

The first phase of development of the Central Energy Plant Expert System (CEPES) project has combined the selection of hardware and software with preliminary knowledge engineering to yield an operational diagnostic program.

CEPES minimally requires an 80286 PC with 4 Mb of RAM, 20 Mb of free space on the hard disk, color graphics capabilities, and a two-button mouse. A recommended configuration for CEPES would consist of an 80386 PC platform with 8 Mb of RAM and 30 Mb of free hard disk space, VGA color, and a two button mouse.

The Kappa tool of Intellicorp (GSA price: $2500) was selected for its high scores in the four required areas: (1) "smartness," (2) performance capabilities, (3) user interface, and (4) ability to communicate with other software.

The CEPES program is broken down into five parts, to match the main systems of a coal-fired central energy plant. Each system is further divided into subsystems that may or may not appear in a specific plant. There are standard subsystems that are automatically implemented during the configuration of a CEPES session, and optional subsystems unique to a particular plant that may also be chosen. This feature makes the CEPES program very versatile in that CEPES is not limited to any single specific plant configuration. CEPES can be easily adapted throughout the many Army facilities.

CEPES runs in a menu-driven user-friendly environment that even personnel with little computer experience can operate. (The layout of the CEPES diagnosis flow was designed to resemble the existing troubleshooting guides that operators are familiar with.)

The CEPES program leads the operator through each screen with concise instructions. As the operator proceeds, CEPES logically runs through the possible causes for the indicated problems. CEPES is able to distinguish between fuel problems, equipment problems, or operating problems.

CEPES gives the operator the ability to view many of the technical aspects that may cause problems, and if a problem is corrected, to go back and instruct CEPES of the correction. CEPES then displays any remaining causes of problems.

Phase two of the CEPES project (currently underway) involves the development of a CEPES system editor and the expansion of the knowledge base to include additional coal combustion technologies. Phase two will also include detailed beta testing of the completed CEPES program and the development of a technology transfer plan.

REFERENCES

Barber, G.R., "LISP vs. C for Implementing Expert Systems," *AI Expert* (February 1987), pp 28-31.

Beerel, Annabel C., *Expert Systems: Strategic Implications and Applications* (Halsted Press, 1987).

Betz Handbook of Industrial Water Conditioning (Betz Laboratories, 1980).

Frenzel, Louis E., Jr., *Understanding Expert Systems* (Howard W. Sams, 1987).

Galperin, A., S. Kimhi, and M. Segev, "Knowledge-Based System for Optimization of Fuel Reload Configurations," *Nuclear Science and Engineering*, Vol 102 (January 1989), p 52.

REFERENCES (Cont'd)

Hayes-Roth, Frederick, Donald A. Waterman, and Douglas B. Lenat, eds. *Building Expert Systems* (Addison-Wesley Publishing Company, 1983).

Hunt, Daniel V., *Artificial Intelligence & Expert System Sourcebook* (Chapman & Hall, 1986).

KAPPA User's Guide, Version 1.2, KAP1.2-UG-MSW2-1 (Intellicorp, Inc., 1991).

Kozlik Gregory W., Kevin W. Bleakley, and Brian C. Skinner "Artificial Intelligence System Optimizes Boiler Performance," *Power Engineering*, Vol 92, No. 2 (February 1988), p 44.

Moriguchi, Syouji, Tatsuji Tanaka, and Ishikawa Keiko, "Expert Shell for Power Systems Diagnosis," *International Workshop on Artificial Intelligence for Industrial Applications 1988* (Kyushu Electric Power Company, Inc./Toshiba Corporation, 1988), p 590.

Moshage, Ralph, et al., *Application of Expert Systems for Diagnosing Equipment Failures at Central Energy Plants*, Draft Technical Report (U.S. Army Construction Engineering Research Laboratories [USACERL], March 1992).

Proceedings: 1987 Conference on Expert-System Applications in Power Plants, EPRO CS-6080 (Electric Power Research Institute, May 27-29, 1987), pp 1-29.

Rich, Elaine, and Kevin Knight, *Artificial Intelligence*, 2d ed. (McGraw-Hill 1991).

Richards, John, and Martin J. Savoie, *U.S. Air Force Central Heating Plant Tuneup Workshop*, Special Report (SR) E-90/03/ADB142292L (USACERL, 1990), Vol 1-16.

Rolston, David W., *Principles of Artificial Intelligence and Expert Systems Development* (McGraw-Hill, 1988), pp 169-178.

Shapiro, Stuart C., ed., *Encyclopedia of Artificial Intelligence*, Vol 1 & 2 (Wiley, 1990).

Steam: Its Generation and Use (Babcock and Wilcox, 1978).

Tanimoto, Steven L. *The Elements of Artificial Intelligence* (Computer Science Press, 1990).

Walker, Terri C., and Richard K. Miller, *Expert Systems 1986: An Assessment of Technology and Application* (SEAI Technical Publications, 1986).

Wang, Y., and A.I. Soler, "An Expert System for the Mechanical Design of Shell and Tube Heat Exchangers," *National Heat Transfer Conference, Vol 108: Heat Transfer Equipment Fundamentals, Design, Applications and Operating Problems* (1989), p 147.

Waterman, Donald A., *A Guide to Expert Systems* (Addison-Wesley Publishing Company, 1986).

Woodruff, Everett B., Herbert B. Lammers, and Thomas F. Lammers, *Steam Plant Operation* (McGraw-Hill, 1984).

APPENDIX A: Expert Systems Technology

This Appendix presents an overview of expert systems technology. Because expert systems are applications of artificial intelligence (AI), the first section provides a brief discussion of some important AI concepts. The next section discusses some of the characteristics and benefits of expert systems. The third section explores expert systems applications and includes a brief review of expert systems in heating plants. The fourth section describes how the main components of an expert system function, and the fifth section discusses the expert system development process including a description of expert system building tools.

Artificial Intelligence

Artificial intelligence (AI) is a field of computer science concerned with what comprises understanding human intelligence, and developing computer programs that exhibit intelligence. The aim of AI software and hardware is to create computers that can mimic some of the functions of the human brain. Expert systems is one major application area of AI.

"Intelligence" includes such elements as the ability to learn or understand from experience, to acquire and retain knowledge, and to solve problems by reason (Hunt 1986, pp 1-2). A fundamental component of intelligence is knowledge. In AI, knowledge representation focuses on methods for efficiently modeling knowledge so that it is easily accessible for application to problem solving within the context of an artificial intelligence computing system. Much early AI research focused on the development of systems that possessed general problem solving knowledge.

Artificial intelligence problem solving can be viewed as a search among alternative solutions to a problem in an attempt to determine the best solution. The search proceeds under the guidance of one or more control strategies, or search techniques, from an initial state to a goal state. The implicit set of all possible paths that the search might take is called the search space (Hunt 1986, pp 231-232 and Rich 1991, pp 29-31). One of the greatest challenges in AI research has been to develop efficient and effective methods to limit the enormous search spaces associated with real-world problems. Techniques have been designed to limit the search space by using a variety of formal search strategies or by incorporating shortcuts derived from information about the nature and structure of problems or tasks associated with a particular domain of knowledge. Such limiting strategies and shortcuts are referred to as heuristics. Heuristic problem solving is one of the most important concepts in AI (Hunt 1986, p 125, Rich and Knight 1991, p 41).

Computer languages used to program artificial intelligence constituted an important area of AI research. The two most well known AI programming languages are LISP (LISt Processing) and PROLOG (PROgramming LOGic). These languages can accommodate the specialized requirements of AI for symbol manipulation, deduction, and implementation of various strategies for searching alternative paths from their initial to goal states. In addition to these and other AI programming languages, specialized programming environments known as knowledge engineering languages are widely used (Tanimoto 1990, pp 494-499).

Hardware for artificial intelligence applications is of two types: conventional computer systems at the mainframe, minicomputer, and microcomputer levels, and specialized computing systems known as AI workstations or LISP machines. Most early AI development was carried out on conventional mainframe computer or minicomputer systems, and these classes of computers continue to be used heavily today. In addition, a wide assortment of AI programming and knowledge engineering language software

is available for use on microcomputers. AI technology became available for PCs in 1985 and has seen much success since then. Although still limited when compared with large mainframe applications, the performance of the PC-based AI and expert system applications are adequate for many purposes. Consequently, many individuals and smaller businesses now use expert systems on relatively inexpensive, common PCs that cost between $3000 and $6000, rather than relying on expensive mainframe hardware or specialized LISP machines.

AI workstations are computing systems specifically designed to address the requirements of artificial intelligence applications. These machines have a number of specialized features that facilitate AI work. For example, they have high-speed processors and large memory capabilities that enable them to deal with the heavy demands of AI search and knowledge representation. Their high-resolution, bit-mapped displays allow for development of sophisticated graphics. Their advanced software environments, which include AI programming languages, knowledge engineering languages, and extensive programming support facilities, address the specialized programming needs of AI development (Tanimoto 1990, pp 496-497).

Expert Systems

An expert system is an artificial intelligence computer program that uses knowledge and inference to address problems that normally require human expertise to resolve. The knowledge in an expert system consists both of the commonly accepted facts in the domain and the heuristic knowledge, or rules of thumb, that the best experts use in decisionmaking. Expert systems typically function as advisors or consultants to human users in making decisions or solving problems in the system's domain (Hunt 1986, pp 104-105).

Components of an Expert System

An expert system consists of five basic components (Hunt 1986, pp 106, 133-135, 147, 261, 270):

1. A **knowledge base** of facts related to the domain

2. An **inference engine**, or rule interpreter, which controls the search of the knowledge base

3. A **working memory**, or data base, which keeps track of the data input, new facts inferred, and the like, in the solution of the problem being worked on

4. A **user interface**, which allows for easy interaction with the system by its intended users and by the system developers. A very important feature of the user interface is an **explanation facility**, which allows a user of the system to query the system's reasoning process and facilitates system debugging.

Differences Between Expert Systems and Conventional Programs

The mere fact that a computer program yields a result comparable to one that an intelligent person would achieve does not make it an expert system. Expert systems differ from conventional programs in several important respects (Hayes-Roth, Waterman, and Lenat, 1983, pp 3-6):

1. **Knowledge**: A conventional program manipulates data while an expert system manipulates knowledge. In an expert system, knowledge is represented symbolically, using strings of characters that represent real-world concepts such as "main entry," "infection," "H7101 regulator." These symbols are organized into a knowledge base of facts about the domain. An expert system uses this knowledge base to solve problems by searching through and pattern-matching among the symbols.

2. **Heuristic problem solving:** A conventional program solves problems through a repetitive algorithmic process whereas an expert system uses heuristic and inferential reasoning. Heuristic reasoning is a shortcut or rule-of-thumb learned through experience that an expert applies to eliminate unproductive paths in solving a problem. The algorithmic approach is intended to guarantee a solution; the heuristic approach does not guarantee a solution, but allows problem solving to take place in domains where the search space is so large that an algorithmic approach would be impossible. It is the use of heuristics, for example, that allows human beings to successfully complete a game of chess despite the fact that there are an estimated 10^{120} possible combinations of moves.

3. **Program structure:** In a conventional program, factual knowledge about the problem being addressed tends to be implicit and intermixed in the program code with procedural instructions for processing data. In an expert system, the knowledge base and the control structure (the inference engine) are separate. The expert system knowledge base can therefore be updated without affecting the inference engine, making program modification and debugging much easier than in conventional programs. In addition, it is possible for different knowledge bases to function with the same inference engine (although for large-scale problems the inference engine will probably need at least some tailoring to each knowledge base).

4. **Self-knowledge:** An expert system can keep track of and display the logical path it used to arrive at a problem solution. A conventional program does not explain how it achieved its results, and the logical process it followed is often difficult to track through its code.

Benefits of Using Expert Systems

When applied appropriately, the expert systems technology can offer many potential benefits:

1. Expert systems make scarce expertise more widely available within the organization thereby helping nonexperts achieve expert-like results.

2. They free human experts for other activities instead of repeatedly solving problems better relegated to an expert system.

3. They promote a standardized, consistent approach to solving relatively unstructured tasks.

4. They enhance organizational effectiveness and efficiency by making readily available solutions to difficult problems that might otherwise require time-consuming research or consultation with experts to solve.

5. They provide a means for capturing and storing valuable knowledge that might be lost if an expert left the organization.

6. Because machine knowledge does not deteriorate with time or disuse as human knowledge can, they provide a means for permanent retention of highly complex knowledge.

7. They perform consistently even at high level tasks where humans might perform inconsistently because of fatigue or loss of concentration (Beerel 1987, pp 31-33; Waterman 1986, pp 32-39).

Uses of Expert Systems

Expert systems have been developed to perform a variety of functions in a wide range of domains. Table A1 lists the broad functional categories of expert systems application and gives an example of a possible application area for each (Hayes-Roth, Waterman, and Lenat 1983, pp 13-16; Waterman 1986, pp 32-33).

Some examples of the broad domain categories in which expert systems have been developed are (Waterman 1986, p 40):

- Aerospace
- Agriculture
- Chemistry
- Computers
- Education
- Electronics
- Engineering
- Energy management
- Finance
- Geology
- Law
- Information management
- Mathematics
- Manufacturing
- Medicine
- Meteorology.
- Military science

Existing expert systems range from small-scale efforts to very large and carefully documented research projects and production systems. Since many systems are known only within the organizations where they originated, it is impossible to estimate how many expert systems have been developed or are in use today. One source which includes systems "commercially available, proprietary programs used in house, and projects still in the prototype stage" identifies 475 systems (Walker and Miller 1986, p 13). Some of the expert systems in the area of boiler heating plants known are:

1. An expert system in its developmental stages by Northern Indiana Public Service Co. (NIPSCO) (Kozlik, Bleakley, and Skinner 1988, p 44).

2. FUELCON, a prototype system developed to optimize fuel reload configurations by Kimhi and Segev (Kimhi and Segev 1989, p 52).

3. An expert system for electric power systems fault analysis developed and now in use by Kyushu Electric Power of Japan (Moriguchi, Tanaka, and Ishikawa 1988, p 590).

4. A prototype system developed to aid in the mechanical design of Tubular Exchanger Manufactures Association (TEMA) type shells and tube heat exchangers (Wang and Soler 1989, p 147).

5. ESCARTA, a protype system in testing that helps analyze boiler tube failures, developed by EPRI/University of Texas (*Proceedings: 1987 Conference on Expert-System Applications in Power Plants*, pp 1-29).

6. TUBEFAIL (Boiler Tube Failure Analysis System), a prototype system in testing developed by Central and Southwest Services.

7. EXACT, a system sponsored by EPRI and developed to troubleshoot gas turbine faults, was field tested in 1987 at the Jersey Central Power and Light, Gilbert Power Station.

8. TIP, a system developed by New York State Electric & Gas Corporation to monitor heat rate in thermal plants.

Table A1

Functional Categories and Applications for Expert Systems

Category	Application
Interpretation	Image analysis
Prediction	Weather forecasting
Diagnosis	Medical diagnosis
Design	Computer configuration
Planning	Job-shop scheduling
Monitoring	Power plant regulation
Debugging	Software correction
Repair	Automobile maintenance
Instruction	Intelligent tutoring
Control	Battlefield management

How Expert Systems Function

This section discusses the components of an expert system in more detail.

The Knowledge Base

Knowledge can be considered a collection of related facts, procedures, models, and heuristics that can be used in problem solving or inference systems. Knowledge varies widely in both content and appearance. It may be specific, general, exact, fuzzy, procedural declarative etc.

The knowledge base is the component of an expert system where facts and rules pertinent to problem solving in the domain are represented. Methods for representing expert knowledge in the knowledge base may be either declarative (representing facts or assertions) or procedural (representing actions) or a combination of the two (Hunt 1986, pp 146-149). There are several commonly used methods to organize and represent knowledge in a knowledge base. Some examples of declarative methods are semantic networks, logical representation schemes, object-attribute-value triplets, and frames. The leading example of the procedurals method is the production rule approach (Hunt 1986, pp 146-149; Tanimoto 1990, p 152).

Semantic nets were originally designed as a way to represent the meanings of English words. In a semantic net, information is represented as a set of nodes connected to each other by a set of labeled arcs, which represent relationships among the nodes. It is useful to think about semantic nets using a graphical notation as shown in Figure A1, even though they cannot be represented as such inside a program.

The relationships represented in the net shown in Figure A1 include the following:
"FTX01 is a firetube boiler"
"A firetube boiler is a boiler"
and "Boilers have burners."

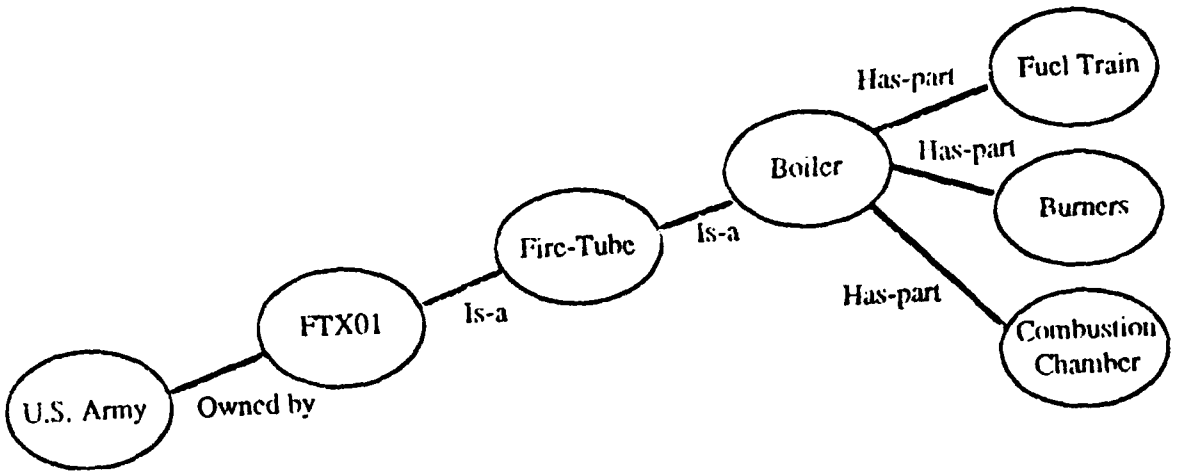

Figure A1. Semantic Net.

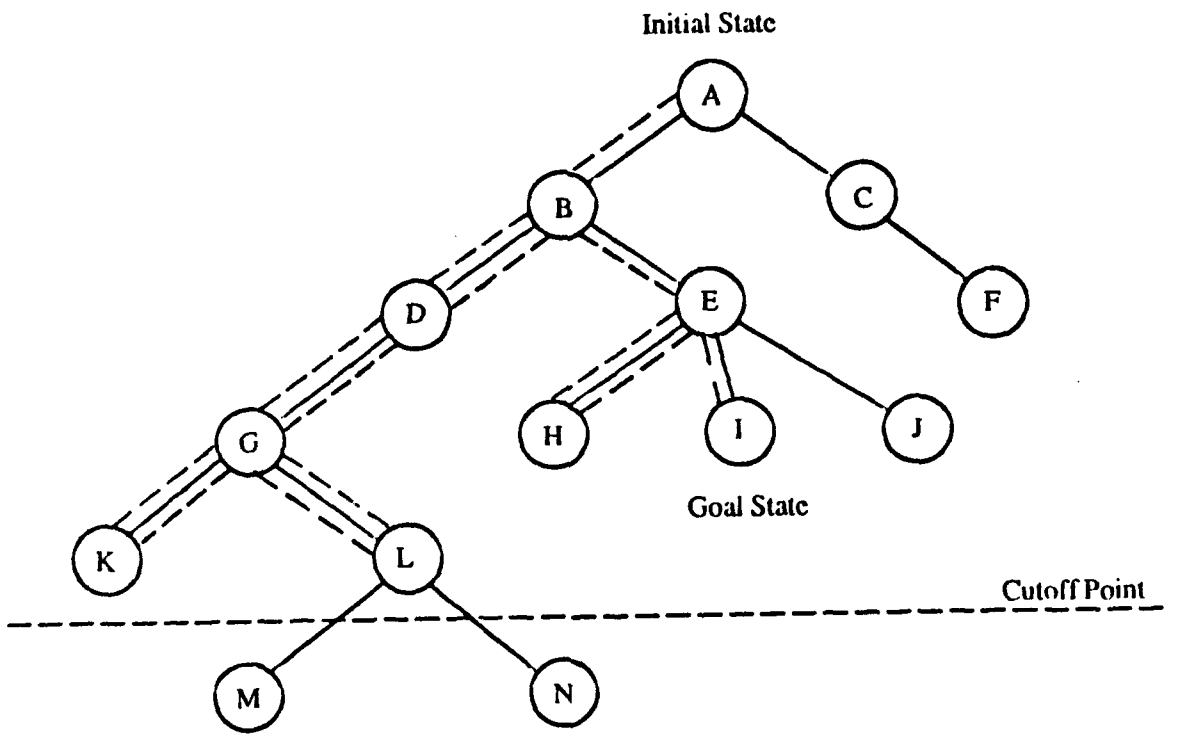

Figure A2. Depth-First Search.

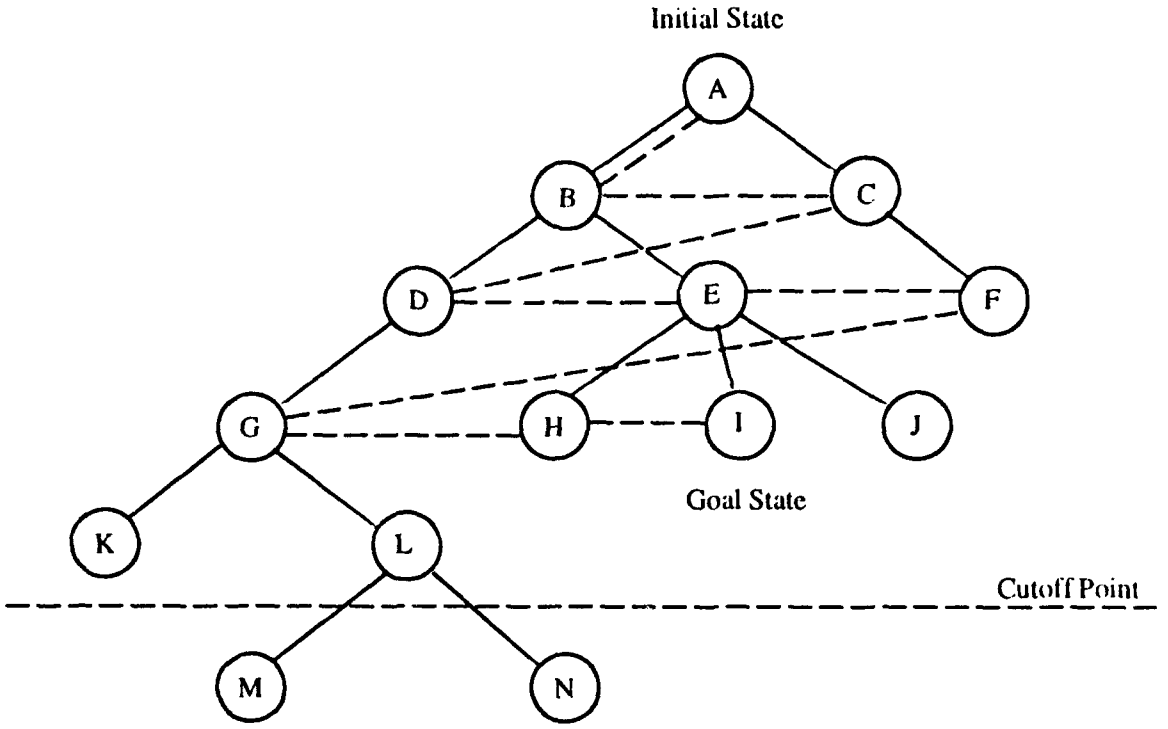

Figure A3. Breadth-First Search.

Examples of arcs illustrated in the net are "is-a" and "has-part". The arcs express how the connected nodes relate and allow for the inference of new facts. For example, from this semantic net one could infer that FTX01 is a boiler and that FTX01 has burners, even though neither of these facts was explicitly stated.

An important property of the semantic net is "property inheritance"; nodes lower in the net can inherit properties from higher nodes so that properties applying to all levels of a hierarchy need not be repeated at each level. In Figure A1 for example, the parts of a boiler can be stored once at the "Boiler" level rather than having to be repeated at the type of boilers and the individual boiler levels (Rich and Knight 1991, pp 203-204).

Among logical representation schemes, the most commonly employed is predicate logic. In predicate logic a proposition consists of objects, persons, concepts, and the like (arguments) about which something is stated (the predicate). For example, the proposition "A component of Boiler is the burner" might be stated in the form of predicate logic as

has component (boiler, burner)

where "boiler" and "burner" are the arguments and "has component" is the predicate, which in this case expresses a relationship between the arguments. Predicate logic lends itself well to inferences. For example, if we add another proposition "A component of the burner is the flame," stated as

has component (burner, flame)

it can be inferred from these two propositions that flame is a component of boiler, although this was not explicitly stated (Rich and Knight 1991, pp 137-138).

In the object-attribute-value triplet method of representing factual knowledge, objects are entities in the domain, attributes are properties associated with the object, and these attributes may possess values. For example, in the triplet:

boiler-efficiency-76 percent

"76 percent" is the value of the attribute "efficiency" associated with the object "boiler." An advantage of this method of knowledge representation is that it facilitates data gathering by the system through questions posed to the user in the form: "What is the [value] of [attribute x] of [object y]?" (Hunt 1986, p 183).

Frames are very powerful and versatile data structures that are especially good for representing stereotyped knowledge about an object, concept, or event. Like a semantic net, frames can be readily organized into a hierarchial network of nodes and relationships, with a frame constituting each node. A frame is subdivided into a collection of attributes called "slots." Values may then be associated with the attributes. In some cases default values may be assigned. Slots can also be associated with procedural attachments that are executed when information in the slot changes. Examples of such procedural attachments are the "if-added" procedure, which executes when new information is placed in the slot, and the "if-needed" procedure, which executes when information is needed in the slot but is not available (Hunt 1986, p 183 and Tanimoto 1990, pp 134-137).

For example, a system using frames might work this way: suppose that a frame-based fuel order system included a frame called "Special Order." Such a frame would include slots for the elements of information needed to process a special order, such as an order number, a description of the fuel type, price, vendor code, and claim date. Most of these slots would contain explicit values. Slots whose initial values could be predicted, such as claim date, might have such values generated by default. Some slots might have procedural attachments. For example, an "if-added" procedure attached to the vendor code slot could search a vendor data base for the full name and address associated with the code.

The production rule method is the most commonly used technique for expert system knowledge representation. Production rules typically take the form IF-THEN, where the IF portion describes a condition, antecedent, or situation, and the THEN portion describes the resulting action, consequence, or response (Hunt 1986, p 203). Production rules are formal representations of heuristics. For example, a personal heuristic could be stated as a production rule as follows:

IF the chance of rain is higher than 50 percent, THEN take an umbrella.

In a rule-based expert system, facts known about the current situation are compared against the domain knowledge expressed as a set of such rules. When the IF portion of a rule is satisfied, the THEN portion is executed. This may result in a new fact being inferred and added to the working memory for possible matches with the IF portion of other rules, or may cause the action specified by the THEN portion to be taken (Rich and Knight 1991, p 36).

Knowledge Engineering

Expert systems use knowledge derived from human experts to guide the analysis and decisionmaking process. (When combined with a natural language computer program, this system results in a powerful learning tool that performs many valuable functions for its user.) Surpassing mere storage and retrieval

functions of conventional computer programs, expert systems provide an "expert-in-a-can" that helps the user in all facets of the relevant analysis. Diagnostic functions, monitoring, analyzing, planning, and explaining are all features of expert systems.

Codification of the chain of rules provided by human experts is a function of the "knowledge engineer." Knowledge engineering is a very conceptual field since human experts tend to have a level of pragmatic expertise developed over time that is not easily recalled in detail. Knowledge engineering requires the expert to step back to a basic stage of learning, recall a concept, and restate it in explicit form. As the knowledge of multiple human experts is incorporated into the "knowledge base," the level of system expertise rises above that of the individual expert.

The goals of the knowledge engineer are: (1) to program in modular format to allow for ease of modification; and (2) to explain in the program why a logic structure was evoked at any given time. For example, when the program uses a rule to explain a certain set of facts that a human expert questions, the program should be able to explain why that rule was evoked instead of another one. This type of reiteration highlights inconsistencies in procedures and data, and frees the human expert to redefine rules while learning from the program.

The simplest expert strategy is to reference each entity separately. Under this scheme of knowledge representation, all entities are independent of each other and may or may not be connected by rules. Therefore, if an entity is assigned a value dependent on other values, it would have to be connected by a rule. Then these rules themselves can be executed in a different order by "metarules," rules that control other rules.

The other scheme is based on the fact that some entities show a number of similarities and can thus be conveniently grouped together in a class. The entity will now have certain characteristic by virtue of being a member of a class and would be automatically assigned a value when the class is assigned a value without the use of an exclusive rule to do so. Under the simplistic scheme of representation, different properties of the same physical object would be separate entities, thus giving the programmer the entire set of entities to track without any grouping to make information storage and retrieval easier. The class/member kind of representation is useful for expert systems developed for process plants.

The Inference Engine

The inference engine is the control structure that organizes, controls, and executes the steps followed by the expert system in searching its knowledge base to arrive at a solution to the current problem. There are two basic search approaches: blind search and heuristic search.

In the absence of a guiding search strategy, the blind search involves consideration of all possible paths from the problem's initial state to a goal state. This might be acceptable for small problems, but would be inefficient for large-scale problems. To make them more efficient, blind searches are guided by search strategies. These include depth-first search, breadth-first search, and forward and backward chaining (Tanimoto 1990, p 177; Hunt 1986, p 62).

The depth-first search pursues a single path in the search tree until either a goal state, a dead end, or an arbitrarily designated cutoff depth is reached. If a dead end or cutoff point is reached, the system will backtrack to a point where another path can be pursued. Depth-first search is potentially more economical in its use of memory than breadth-first search. However, a serious deficiency of this method is that it does not ensure an optimal solution. In a large-scale problem, an arbitrary cutoff depth is necessary to prevent the system from using tremendous amounts of time pursuing unproductive search paths to great depths. But if the cutoff point is too shallow, a solution will never be reached, and if the

cutoff point is too deep, the solution path will be non-optimal (Shapiro 1990, p 996). Figure A2 gives an example of a depth-first search.

The breadth-first search examines all nodes in each level of the search tree before moving on to the next level. This method will find the shortest path to a solution, if one exists, but is not practical if the solution is deep in the search tree because each deeper level of the search tree becomes much more complex, and each level must be generated before the next level can be examined (Shapiro 1990, p 995; Hunt 1986, p 64). Figure A3 gives an example of a breadth-first search.

Forward chaining is a data-driven search method. The inference engine starts with known facts that it tries to match with facts in the knowledge base. When such matches occur, new facts are inferred, which can then be matched with other facts. This process continues until no new conclusions can be reached. At this point either a goal state has been reached or, if there were no facts to support a goal state, the search has failed. In either case additional goals can then be sought (Hunt 1986, pp 112-113; Shapiro 1990, pp 91-92).

The backward chaining method starts from a potential goal state and works backward through the search tree seeking facts that support that goal. If the available facts do not support the goal, the search fails. Another potential goal is selected, and the search is renewed. Forward chaining is more appropriate when the number of initial states is greater than the number of goal states. When the number of goal states exceeds the number of possible initial states, backward chaining is more efficient (Hunt 1986, pp 58-59; Shapiro 1990, p 824).

Although the blind search techniques discussed above are sometimes used, the expert systems searching process is generally modified by the use of heuristic search to limit the number of alternative solution paths that must be considered. A heuristic search generally involves a process of evaluating the current node in the search tree and predicting the quality of succeeding nodes for their desirability as subsequent nodes in the path to the goal. Two examples of heuristic search techniques are difference reduction and hill climbing (Shapiro 1990, pp 578, 996-997).

Difference reduction (also referred to as means-ends analysis) uses a combination of forward and backward chaining to shorten the distance between the current node and a goal state by setting subgoals. For example, suppose that a certain heuristic would attain the desired goal state. Suppose further that it is not possible to apply the heuristic from the current node, but that there is a nearby node from which it could be applied. Employing the concept of difference reduction, the ultimate goal would be temporarily set aside in favor of the subgoal of reaching the nearby node, enabling use of the desired heuristic. By applying this process repeatedly, smaller and smaller subproblems, each with search spaces much smaller than the original problem, can be solved. When all the subproblems are solved, the main problem would also be solved (Hunt 1986, p 89; Rich and Knight 1991, pp 94-97).

In a hill climbing approach, when the evaluation of a node reveals that it is not the goal state, the difference between that node and the goal state is calculated. A comparison of the sequence of calculated differences indicates whether the search is moving closer to or farther away from the goal state. If movement is away from the goal state, the search backtracks until a new search path can be taken (Rich and Knight 1991, pp 65-67).

Working Memory

The working memory is the dynamic memory where the current status of an expert system consultation is stored. It contains the initial information provided to the system to enable the search process to start. As rules are examined and executed, the working memory is updated to contain new facts inferred, values established, etc., which are then available for further use in the decisionmaking process.

The working memory also keeps track of the rules the system has examined and executed and in what sequence so that the reasoning process employed can be provided to the user if required.

User Interface

The user interface software permits interaction between the user and the expert system. The interface may contain pre-formulated questions and menus to facilitate the collection of data needed by the system to search its knowledge base. The interface also provides the means to display the solution reached by the system.

For the expert system to be most useful, the user interface should include an explanation facility. This permits the user to ask the system to display the reasoning process by which a particular result was achieved. The explanation facility not only enhances the system's credibility but also greatly facilitates debugging when unexpected or erroneous results are produced.

Expert System Development Processes

Suitable Problem Definition

Before getting started on an expert system development project, a problem suitable for application of this technology must be identified. Expert systems are not well suited for many types of problems and should be applied only when they are possible, justified, and appropriate (Waterman 1986, p 127).

For an expert system to be possible, the task to be carried out must:

1. Require only cognitive (i.e., not physical) skills and must not require "common sense" reasoning.

2. Be related to an area of human expertise. There should be experts in the domain who can articulate their methods and who are in general agreement as to what constitutes solutions to the problems the expert system is intended to solve.

3. Fall within a reasonable range of difficulty. If solving the problem is a task that cannot be taught to a novice by an expert, or if the task takes days or weeks for an expert to carry out, the size and cost of an expert system to tackle the task would be prohibitive. On the other hand, if a large problem can be segmented, its component parts might be suitable for expert systems.

Expert System Building Tools

There are a large number of programming environments, or expert system building tools, now available to assist the knowledge engineer in the construction of an expert system. These tools fall into two main classes: programming languages and knowledge engineering languages (or shells). These programming environments are supported by a variety of facilities.

Two types of programming languages are used in building expert systems:

1. Problem-oriented languages, such as C and PASCAL, which were designed for conventional software development,

2. Symbol-manipulation languages, such as LISP and PROLOG, which were designed to represent and perform operations on concepts expressed as symbols, for example as list structures or logical representations of concepts.

Expert systems have been developed using virtually all the major programming languages. For example, C, with its speed and flexibility, has become increasingly popular as an expert system building tool (Frenzel 1987, pp 113-115). However, the symbolic manipulation languages possess special characteristics that make them especially suitable for use as expert system development tools. LISP, for example, features flexible symbol manipulation, automatic memory management, and uniform treatment of program code and data.

In the United States, LISP is the most widely used AI programming language. Developed in the 1950s by John McCarthy, one of the pioneers of AI, LISP has retained its popularity due to its versatility and powerful capabilities. Many versions of LISP are available for all classes of hardware, and, as noted above, there exists a special class of computer known as the LISP machine that has specialized features to support LISP programming. In Europe, PROLOG is the most popular AI language. PROLOG is based on predicate logic and contains its own built-in inference engine. As with LISP, many versions of PROLOG are available. Besides these two, a number of other AI programming languages have been developed.

Knowledge engineering languages, sometimes referred to as "shells," are specialized programming environments tailored for expert system development. The basic components of an expert system shell include a knowledge representation facility, an inference engine, and a variety of support capabilities. Some shells, such as those developed by removing the knowledge base from an existing expert system, are relatively specialized, emphasizing one particular knowledge representation scheme and one principal inferencing technique. More generalized and versatile are the large hybrid tools, which support multiple knowledge representation schemes and inferencing techniques and feature very sophisticated support facilities.

Shells can be very useful for rapid prototyping of expert systems. These shells can facilitate knowledge-base construction by providing good knowledge-representation facilities. However, the work of actually formulating the knowledge is not the job of a shell; this generally requires interactive dialogues between a knowledge-base building tool and one or more humans.

Commercially available knowledge engineering languages exhibit a great deal of variety with respect to their hardware requirements, knowledge representation methods, and inferencing techniques. Furthermore, existing tools are subject to modification, and new tools are being brought onto the market. Selection of a tool for a development project must therefore be based on a careful analysis of the most current information (Rolston 1988, pp 169-178).

The prototype development system for an energy plant expert system was envisioned as being PC-based, using a commercially available expert system shell. The software structure (including the information input, knowledge engine, alternative selection process, rule modification process, and output format) was designed to reflect the expertise of a select group of individuals known for their abilities in diagnosing equipment problems at heat plants. Monitoring and diagnosing such problems can be a complex task. A prototype system was meant to initially perform only modest fault diagnosis, and possibly to add more vigorous diagnostic capabilities in the future.

For this reason, an IBM PC/AT or equivalent was considered adequate. Other equally capable, expensive, higher performance workstations do exist, but the superfluous increase in speed could not justify the higher cost for this particular application. Intel's 80386 and Motorola's 68020 microprocessors (CPUs) are now commonly available for the PC environment and will soon be available for less than $5000. These high-performance workstations include those manufactured by Sun, Apollo, Dec, NEC, and Hewlett-Packard.

Implementation of an expert system requires the user to make a fundamental choice between creating the expert system development tools in-house or buying the required software on the commercial market. The correct decision relies solely on the system's performance requirements and intended applications.

The need for very powerful, extensive, or specialized expert systems has, in the past, usually demanded user-created development tools. Today, this situation persists at the extreme end of the spectrum, but is rapidly fading out. Commercially available PC software and hardware has become more powerful and versatile. This new wave of PC software and hardware can create increasingly specialized, extensive expert systems, offering the extra advantage of compatibility with standard business software.

This raises the question of economic feasibility. An absolute need for user-created expert systems development tools will make the system costly. Programming in LISP is expensive, as knowledgeable LISP programmers are in short supply. Many expert systems developers have tried to circumvent this shortage by programming in C language, but this creates problems since, for AI, C is a less efficient operating environment than LISP. This is due to C's limited capabilities for symbolic manipulation. These functions, then, must be programmed on top of C, and thus increase complexity and memory requirements (Barber 1987, pp 28-31). Another reason for the high cost of user-developed expert systems tools is the extravagant price of LISP-dedicated machines and similar systems. It was decided, therefore, that a commercially available expert system software package should be used for developing the prototype in this project.

In evaluating the available expert system shells, the focus was on the various features that would enhance task efficiency and provide flexibility for future modifications. Features deemed to meet these requirements include:

- an ability to create real-time graphics to produce a user friendly interface
- an ability to interact with combustion equipment, sensors, and instrumentation
- an ability to access external databases
- ease of interface with external software
- speed of execution in a run-time environment
- a math library and an ability to evaluate math equations
- an explanation of how the knowledge representation paradigm relates to the monitoring and diagnosing task.

APPENDIX B: KAL File Codes

Spreader Stoker Subsystem name	CEPES Subsystem Names	Instance Files	Subsystem ThreeLetter Code
Coal storage	CoalStorage	COALSTOR .KAL	cst
Belt feeder/conveyor	Belt	BELT .KAL	blt
Apron feeder/conveyor	Apron	APRON .KAL	apr
Reciprocating feed/conv	RecipFeed	RECIP .KAL	rcp
Drag feeder/conveyor	Drag	DRAG .KAL	drg
Flight feeder/conveyor	Flight	FLIGHT .KAL	flt
En Masse feeder/conveyor	Enmasse	ENMASSE .KAL	enm
Vibrating feeder/conveyor	VibrateFeede	VIBFEED .KAL	vbf
Bucket feeder/conveyor	Bucket	BUCKET .KAL	bkt
Chute feeder/conveyor	ChuteFeeder	CHUTFEED .KAL	chf
Coal bunker	CoalBunker	COALBUNK .KAL	cbk
Chute to Hopper	ChuteToHopper	CHUTEHOP .KAL	chh
WeighLarry	WeighLarry	LARRY .KAL	whl
Belt Weighscale	BeltWeighScale	WGHSCALE .KAL	bws
Coal hopper	CoalHopper	COALHOP .KAL	hop
Feeder distributor	FeederDistribu	FEEDDIST .KAL	fdr
Traveling grate	TravelGrate	TRAVEL .KAL	trg
Dumping grate	DumpingGrate	DUMPING .KAL	dpg
Vibrating grate	VibrateGrate	VIBGRATE .KAL	vbg
Overlapping grate	OverlapGrate	OVERLAP .KAL	olg
Forced draft fan	ForcedDraftFan	FORCED .KAL	fdf
Overfire air jets	OverfireAirJet	OVERFIRE .KAL	oaj
Fly ash reinjection	FlyAshReinject	REINJECT .KAL	far
Fly ash handling	FlyAshHandling	FLYHAND .KAL	fah
Fly ash silo	FlyAshSilo	FLYSILO .KAL	fas
Bottom ash handling	BottomAshHandl	BOTHAND .KAL	bah
Bottom ash silo	BottomAshSilo	BOTSILO .KAL	bas
Refractory surfaces	RefractorySurf	REFRAC .KAL	rfs
Boiler fireside tubes	Firetubes	FIRETUBE .KAL	bft
Boiler operation	Boiler Operati	BOILER .KAL	bop
Economizer	Economizer	ECONOM .KAL	eco
Air heater	AirHeater	AIRHEAT .KAL	aht
Multicyclone	Multicyclone	MULTICYC .KAL	cyc
Electrostatic precipator	ESP	ESP .KAL	esp
Baghouse	Baghouse	BAGHOUSE .KAL	bag
Hot process softener	HotProcess	PROCESS .KAL	hot
Sodium zeolite softener	SodiumZeolite	ZEOLITE .KAL	zeo
Deaerator	Deaerator	DEAIR .KAL	dea
Condensate	Condensate	CONDENSE .KAL	con
Boiler water side	BoilerWater	BOILWTR .KAL	bwr
Induced draft fan	InducedDraftFan	INDUCED .KAL	idf
Stack/chimney	StackChimney	STACK .KAL	stk
Coal Quality	CoalQuality	COAL .KAL	cqu

APPENDIX C: CEPES Configurations

System	Standard	Option	Optional	Instructions
	Subsystem	Description	Subsystem	
Water	Condensate	Softener	Hot Process	must select one
Treatment	Deaerator		Sodium Zeolite	
	Boiler Watersidc			
Coal	Coal Storage	Conveyors or	Apron	select at least
Handling	Coal Bunker	Feeders	Belt	one, but no
	Coal Hopper		Recip Feed	more than four
			Drag	
			Flight	
			En Masse	
			Bucket	
			Chute Feeder	
		Coal Feeder	Weigh Larry	must select one
			Chute to Hopper	
			Belt Weigh Scale	
Boiler	Feeder Distributor	Grates	Dumping Grates	must select one
	Forced Fan Draft		Traveling Grates	
	Overfire Airjets		Vibrating Grates	
	Boiler Tubes		Overlapping Grates	
	Refractory Surfaces			
	Boiler Operation	Heat Recovery	Economizer	must select one
		Unit	Air Heater	or neither
Polution	Induced Fan Draft	Particulate	Baghouse	must select one
Control	Stack Chimney	Removal	ESP	or neither
			Multicyclone	yes or no
			Collector	
Solid	Fly Ash Handling			
Waste	Fly Ash Silo			
	Bottom Ash Handling			
	Bottom Ash Silo			

APPENDIX D: CEPES Function Documentation

This appendix provides a brief description of every function written in CEPES.

Color Functions

IsGreen? - Given a subsystem or problem as an argument, this function returns TRUE if the argument is green and FALSE if it is not green.

IsRed? - Given a subsystem or problem as an argument, this function returns TRUE if the argument is red and FALSE if it is not red.

IsYellowOrRed? - Given a subsystem or problem as an argument, this function returns TRUE if the argument is either red or yellow. If not, the function returns FALSE.

AllGreen? - This function is called with the name of a subsystem as the argument. If all of the problems associated with these subsystem are green, then TRUE is returned. If any problem is not green, then FALSE is returned.

NoYellowOrRed? - This function is called with the name of a subsystem as the argument. It returns TRUE if there are no problems in this subsystem that are yellow or red. If there is one problem that is yellow or red, then FALSE is returned.

AnyRed? - The argument to this function is a subsystem. If any problem is this subsystem is red, then TRUE is returned. If none are red, then FALSE is returned.

MakeProblemsWhite - This function is called with the name of a class of problems. (Each subsystem is linked with a unique class of problems in CEPES.) Each individual problem in this problem class is now made white.

MakeYellow - Given a subsystem or problem as an argument, this function will make it yellow. If the argument is a problem, then this function will first check to see if the problem is already green. If it is, then the change to yellow will not be executed.

MakeWhite - Given a subsystem or problem as an argument, this function will make it white.

Display Functions

DisplaySubsystem - Whenever a subsystem box is to be displayed, this function is called with the subsystem as an argument to it. Before displaying the subsystem, this function must check the status of the problems associated with this subsystem to determine the proper color for the subsystem.

DisplayProblem - Whenever a problem box is to be displayed, this function is called with the problem as an argument to it. Before displaying the problem, this function checks to see if the status of other problems in other subsystems should cause this problem to be made yellow. If not, then the pre-existing problem color is used to display it.

BuildMiddleColumn - When given the name of a subsystem, this function examines what other subsystems can cause problems in this subsystem and builds a list out of this information.

FormatSubsystemColumn - Given the list constructed by BuildMiddleColumn, this function will format the display of the middle column in the three column displays.

Problem Processing Functions

Problems are represented in the right-most column in the three column format of CEPES subsystem screens.

ProblemPresentAction - This function is executed whenever the user indicates that a problem is present. It takes no argument. The current active problem is turned yellow and posted on a list of problems to be solved. This function also determines what other problems and subsystems need to be made yellow by examining the IsCausedBy slot for the current active problem.

ProblemAbsentAction - This function is executed whenever the user indicates that a problem is absent. It takes no argument. The current active problem is turned green and removed from the list of problems to be solved (if it was there).

Cause Processing Functions

Causes are represented in the leftmost column in the three column format of CEPES subsystem screens.

NotInSpecAction - When the user indicates that a cause is not within specification, this function turns that cause red and makes all of the problems that this might cause to be yellow. It does not take an argument.

JustPutInSpecAction - When the user indicates a cause has just been returned to specification (by corrective action), this function changes the cause to green. The corrective action is now evaluated by calling the CheckFix function.

AlreadyInSpecAction - When a cause is examined and found to be already within specification, this function is invoked. It makes the cause green.

Drawing Functions

These four functions are involved in the presenting and hiding of the bitmap drawings associated with subsystem and problems. None of them invoke arguments.

SubsystemDrawingAction - This function is called when users indicate they want to see a drawing of a subsystem. It creates a new window with the proper title and displays the drawing file.

ProblemDrawingAction - This function is called when users indicate they want to see a drawing of a problem. It creates a new window with the proper title and displays the drawing file.

ClearSubsystemDrawingAction - This clears the drawing and window that SubsystemDrawingAction displays.

ClearProblemDrawingAction - This clears the drawing and window ProblemDrawingAction displays.

External Function Execution Functions

Two functions are responsible for executing the externally defined executable files that may be attached to a subsystem or a problem, SubsystemFunctionExecution and ProblemFunctionExecution.

Right Action Functions

CEPES has three types of "boxes": systems (e.g., coal handling, pollution control), subsystems (e.g., feeder distributor, belt conveyor, fly ash handling), and problems/causes of problems (just called causes), such as pluggage or belt alignment. When the user selects one of these, one of the three functions described below is invoked.

SystemRightAction - Invoked when a system box is selected by the user. This function hides the current screen and activates the initiation function for the next screen.

SubsystemRightAction - When a subsystem box is selected by the user, this function checks to see if the problem file for this subsystem has been loaded into CEPES. If not, the file is loaded. The screen is now cleared and the initiation function for the subsystem is now executed. The initiation function will now present the three column format screen.

ProblemRightAction - A user selects a problem/cause box when a technical information is desired or when an observation is to be entered. This function opens a new window in which the text information is displayed (e.g., general instructions, general problem/cause description, corrective action information (for causes only), list of possible causes (for problems only). It also displays the appropriate button images that let the user enter status information about the problem or cause.

Cause Priority Functions

There are four functions involved when a user wants CEPES to display what causes of a problem should be given priority.

OrderAction - This is the first function activated when the user selects the priority button. It presents a menu of the problems associated with the activated subsystem on this screen. When the user selects a problem, it checks to see whether that problem is red or not. If the problem is red, the OrderRedAction function is called with the selected problem sent as an argument. If the problem is not red, the PostOrderAction function is called with the selected problem sent as an argument.

PostOrderAction - This function formats the IsCausedBy of the selected list into a two column, numbered format for display.

OrderRedAction - This function takes the IsCausedBy slot of the selected problem, makes a copy of it, and removes any causes that have been ruled out.

PostRedAction - This function formats and displays the copy of the IsCausedBy slot made by OrderRedAction.

Repair Validation Function

CheckFix - When the user indicates that a cause has been returned to specification, this function asks if any of the problems has been remedied. If one or more problems have been remedied, the problem(s) is made green and the cause that was repaired is marked as having caused a failure.

Problem Migration Functions

There are two functions that help users move from one problem to another.

SelectNotFixedProblem - The SelectNotFixedProblem function is activated by selecting the Problem button in the lower right hand corner. This function creates the menu of outstanding problems and asks the user to select one. The selected problem is sent to the GoToNotFixedProblem function.

GoToNotFixedProblem - This function hides the current screen and activates the Initiation function for the subsystem containing the selected problem.

Background Drawing Functions

The Kappa product requires special functions for drawing the lines that connect systems or subsystems. These functions are not activated in the unusual manner (i.e., invocation by name in a command line). Rather they are linked to a drawing image and automatically invoked whenever that drawing image is shown.

For example, CEPES contains a drawing called SystemBackground. SystemBackground has a slot called DrawFunction and the value of DrawFunction is DrawSystems. Whenever the SystemBackground image is displayed, the DrawSystems function is automatically invoked by the Kappa product. The DrawSystems contains the coordinates and commands necessary for drawing the lines that connect the CoalHandling, Boiler, WaterTreat, SolidWaste, and PollutionCon systems.

DrawSystems - described above.

DrawProblems - DrawProblems is attached to the ProblemBackground drawing. DrawProblems draws the lines that form the familiar three column format when displaying the problems associated with a particular subsystem.

DrawBoiler - This function is attached to the BoilerBackground drawing. It draws the lines that connect the boiler tubes, boiler operation, feeder distribution, forced draft fan, fly ash reinjection, grates, heat recovery, overfire air jets, and refractory surfaces.

DrawSolidWaste - This function is attached to the SolidWasteBackground drawing. It draws the lines that connect fly ash handling, fly ash silo, bottom ash handling, and bottom ash silo.

DrawWater - This function is attached to the WaterBackground drawing. It draws the lines that connect the boiler (water side), condensate, deaerator, and softener.

DrawPollution - This function is attached to the PollutionBackground drawing. It draws the lines that connect the baghouse, multicyclone, electrostatic precipitator, induced draft fan, and the stack/chimney.

DrawCoal - This function is attached to the CoalBackground drawing. It draws the lines that connect the coal storage, feeders/conveyors, coal bunker, feeder, and coal hopper.

Background Configuration Functions

These functions display and hide the images that are used repeatedly.

SubsystemScreenConfigure - This function sets the screen title to the proper value, adds the screen name to the history list if it is not already there, and calls the SetSubsystemBackground function.

SetSubsystemBackground - This function is responsible for displaying most of the template items in the three column screen format. The function displays the ProblemBackground (see the DrawProblems function), the subsystem title, other column titles, and the buttons in the lower right side corner of the screen.

HideSubsystemBackground - Hides everything shown by SetSubsystemBackground.

HideProblemButtons - This function hides the status buttons (e.g., AlreadyInSpecButton, ProblemPresent-Button, etc.) shown when a particular problem/cause is selected by the user.

Display Functions for Configurable Subsystems

Several subsystems in CEPES are configurable. These subsystems contain optional substitutions (e.g., grates can be traveling, vibrating, dumping, or overlapping). When displaying and hiding these subsystems, the proper configuration for that subsystem must be checked. The following functions perform that checking before displaying or hiding these subsystems:

- DisplayFeederConveyors
- HideFeederConveyors
- DisplaySoftener
- HideSoftener
- DisplayGrates
- HideGrates
- DisplayCoalFeeder
- HideCoalFeeder
- DisplayHeatRecovery
- HideHeatRecovery
- DisplayPartRem
- HidePartRem.

CEPES System Control Functions

These functions control the presentation of screens that are not related to any specific subsystem or problem.

RunApplication - This is the first CEPES function invoked on startup. It sets up some variables and presents the block flow diagram on the opening screen.

ClearBlockAction - When the "Click here to begin" box is selected with the mouse, this function is activated. It clears the screen and calls the BeginInit function.

BeginInit - This function shows the explanatory text in a window and the "Click here to continue" bar on the bottom of the screen.

BeginDeInit - This function is activated when the "Click here to continue" bar shown by BeginInit is selected. It hides the images shown by BeginInit and called by the Startup function.

Startup - Startup has many commands in it, but its primary purpose is to determine if the user wants to create a new plant configuration or analyze an old one. If the user wants to analyze a old one, Startups loads the proper file from the config directory.

HelpInit - This sets up the help screen.

HelpDeInit - This hides the help screen.

HistoryInit - This sets up the history file where the session history is kept.

HistoryCloseOut - This function closes the history file whenever the session is ended or when the user clears all of the colors (with the Clear button).

NewHistoryInit - NewHistoryInit sets up new history files when the user has performed a clear action.

HistoryAction - This function has nothing to do with the recording of the session history. It is the function that is activated when the user selects the history button in the lower right side corner, allowing the user to move to any screen previously viewed.

Configuration Functions

Because the configurations of stoker-fired coal plants may vary, CEPES allows users to customize the arrangements of subsystems. The user configures a plant by telling CEPES that a new configuration is to be created. There are eight configuration functions.

Six of the configuration functions (one for each configurable subsystem, present menus to users from which the appropriate subsystem are selected. The menu choices are loaded from the systems.kal file in the config directory. This file contains all of the possible subsystems in a stoker-fired coal plant:

- ConfigureFeederConveyors
- ConfigureSoftener
- ConfigureGrates
- ConfigureCoalFeeder
- ConfigureHeatRecovery
- ConfigurePartRem.

A seventh function, ConfigureSystem, serves as a master function that invokes the six specific configuration functions. The eighth and final function, SaveConfiguration, saves the plant configuration into a file named by the user.

System Init and DeInit Functions

For each of the five systems, there are initiation and de-initiation functions. The Init functions present the appropriate subsystem image for the system and the function buttons in the lower right side corner.

The DeInit hides everything that was presented by the Init functions:

- SolidWasteInit
- SolidWasteDeInit
- WaterTreatInit
- WaterTreatDeInit
- PollutionConInit
- PollutionConDeInit
- CoalHandlingInit
- CoalHandlingDeInit
- BoilerInit
- BoilerDeInit.

Fake SubSystem Functions

When the subsystems in the Solid Waste system are displayed, boxes for the Boiler and Pollution Control system are also shown. These boxes are colored orange because they are inert images not connected to the real systems. When the user selects them, messages are posted to that effect. The following two functions present those messages:

- FakeBoilerAction
- FakePollutionControlAction.

Miscellaneous Functions

PruneIsCausedBy - The IsCausedBy list for each problem may contain the names of subsystems that are not appropriate for any given configuration. This function takes a problem name as an argument and prunes the inappropriate subsystems for the IsCausedBy slots.

FileExists? - Given a file name as an argument, this function will return TRUE if the file exists and FALSE if it does not.

ClearAction - One of the function buttons in the lower right corner of the screen allows the user to clear all of the subsystem and problem colors (i.e., to make them all white). This function is invoked when this button is selected. Besides resetting the colors, it also invokes the NewHistoryInit function to start a new history session.

APPENDIX E: CEPES Diagnosis Flow

KEY: SUBSYSTEMS.....ALL CAPS
Problems.....Initial Caps
causes.....all lower case

COAL HANDLING

COAL STORAGE
- Spontaneous Combustion
  - pile consist
  - dry area
  - storage layers
  - COAL QUALITY

BELT
- Excess Wear
  - belt speed
  - belt alignment
  - belt lubrication
  - firmly set
  - belt tension
  - belt clean
  - COAL QUALITY
  - COAL STORAGE
- Pluggage
  - belt speed
  - foreign objects
  - belt condition
  - damp coal
  - COAL QUALITY
- Insufficient Capacity
  - belt condition
  - damp coal
  - coal spilling
  - COAL QUALITY
- Erratic Feeding
  - belt condition
  - damp coal
  - coal spilling
  - COAL QUALITY

RECIP FEEDER
- Excess Wear
  - shaft speed
  - shaft stroke
  - lubrication
  - COAL QUALITY
  - COAL STORAGE
- Insufficient Capacity
  - shaft speed
  - shaft stroke
  - damp coal
  - foreign objects
  - lubrication
  - COAL QUALITY
- Pluggage
  - shaft speed
  - shaft stroke
  - damp coal
  - foreign objects
  - lubrication
  - COAL QUALITY
- Erratic Feeding
  - shaft speed
  - shaft stroke
  - damp coal
  - foreign objects
  - lubrication
  - COAL QUALITY

VIBRATE FEEDER
- Excess Wear
  - shaft springs
  - vibrations
  - damp coal
  - COAL QUALITY
  - COAL STORAGE
- Insufficient Capacity
  - shaft springs
  - vibrations
  - damp coal
  - foreign objects
  - COAL QUALITY
- Erratic Feeding
  - shaft springs
  - vibrations
  - damp coal
  - COAL QUALITY
  - foreign objects
- Pluggage
  - shaft springs
  - vibrations
  - damp coal
  - foreign objects
  - COAL QUALITY

EN MASSE
- Excess Wear
  - start stop
  - shutdown operation
  - en masse speed
  - COAL QUALITY

- Insufficient Capacity
  - feed rate
  - en masse speed
  - foreign objects
  - damp coal
  - COAL QUALITY
- Pluggage
  - foreign objects
  - feed rate
  - damp coal
  - COAL QUALITY
- Erratic Feeding
  - foreign objects
  - feed rate
  - damp coal
  - COAL QUALITY

BUCKET
- Excess wear
  - start stop
  - shutdown operation
  - feed rate
  - chain inspect
  - belt align
  - COAL QUALITY
- Pluggage
  - packed coal
  - feed rate
  - damp coal
  - buckets secure
  - COAL QUALITY
- Insufficient Capacity
  - feed rate
  - damp coal
  - belt align
  - buckets secure
  - buckets mining
  - COAL QUALITY
- Erratic Feeding
  - packed coal
  - feed rate
  - damp coal
  - buckets secure
  - COAL QUALITY

DRAG
- under development

FLIGHT
- under development

APRON
- under development

WEIGH LARRY
- under development

BELT WEIGH SCALE
- under development

CHUTE TO HOPPER
- Insufficient Capacity
  - foreign objects
  - damp coal
  - COAL QUALITY
- Pluggage
  - foreign objects
  - BUCKET
  - BELT
  - COAL BUNKER
  - CHUTE FEEDER
  - EN MASSE
  - RECIP FEEDER
  - VIBRATE FEEDER
  - COAL QUALITY
- Erratic Feeding
  - foreign objects
  - damp coal
  - COAL QUALITY

CHUTE FEEDER
- Erratic Feeding
  - foreign objects
  - damp coal
  - COAL QUALITY
- Pluggage
  - foreign objects
  - damp coal
  - COAL QUALITY
- Insufficient Capacity
  - foreign objects
  - frozen coal
  - ratholing
  - COAL QUALITY

COAL BUNKER
- Pluggage
  - foreign objects
  - COAL QUALITY
  - BELT
  - RECIP FEED
  - VIBRATE FEED
  - BUCKET
  - EN MASSE
  - CHUTE FEEDER
- Insufficient Capacity
  - foreign objects
  - damp coal

COAL QUALITY
Erratic Feeding
foreign objects
damp coal
COAL QUALITY

COAL HOPPER
Insufficient Capacity
foreign objects
hopper level
reg gate
damp coal
cutoff gate
COAL QUALITY
VIBRATE GRATE
Pluggage
foreign objects
cutoff gate
BELT
RECIP FEED
VIBRATE FEED
BUCKET
EN MASSE
COAL BUNKER
CHUTE TO HOPPER
CHUTE FEEDER
COAL QUALITY
Erratic Feeding
foreign objects
reg gate
damp coal
COAL QUALITY

POLLUTION CONTROL

ESP
Carbon Burnout
gas flow
applied voltage
corona strength
COAL QUALITY
TRAVEL GRATE
DUMPING GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
OVERFIRE AIRJETS
Excess Emissions
gas flow
applied voltage
corona strength
electrode misalign
spark rate
gas temp
plate rapping
clean plates
opacity
anti sway
air infiltration
uneven air flow
TRAVEL GRATE
DUMPING GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
COAL QUALITY
Erosion
gas flow
applied voltage
corona strength
TRAVEL GRATE
DUMPING GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
OVERFIRE AIRJETS
COAL QUALITY

BAG HOUSE
Carbon Burnout
dirty bags
gas flow
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
OVERFIRE AIRJETS
DUMPING GRATE
TRAVEL GRATE
VIBRATE GRATE
Excess Emissions
dirty bags
opacity
static pressures
gas temp
gas flow
COAL QUALITY
DUMPING GRATE
TRAVEL GRATE
VIBRATE GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
OVERFIRE AIRJETS

MULTI CYCLONE
Carbon Burnout

gas flow
COAL QUALITY
DUMPING GRATE
TRAVEL GRATE
VIBRATE GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
OVERFIRE AIR JETS

Excess Emissions
gas flow
opacity
air infiltration
static pressures
gas temp
oxygen conc
DUMPING GRATE
TRAVEL GRATE
VIBRATE GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
COAL QUALITY

Erosion
gas flow
COAL QUALITY
DUMPING GRATE
TRAVEL GRATE
VIBRATE GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
OVERFIRE AIR JETS
FLY ASH HANDLING

STACK CHIMNEY
Carbon Burnout
COAL QUALITY
OVERLAP GRATE
DUMPING GRATE
VIBRATE GRATE
TRAVEL GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
OVERFIRE AIR JETS
FLY ASH HANDLING
ESP
MULTICYCLONE
BAG HOUSE

Particulate Emission
COAL QUALITY
OVERLAP GRATE
DUMPING GRATE
VIBRATE GRATE
TRAVEL GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
OVERFIRE AIR JETS
FLY ASH HANDLING
ESP
MULTICYCLONE
BAG HOUSE

SO2 Emission
COAL QUALITY
OVERLAP GRATE
DUMPING GRATE
VIBRATE GRATE
TRAVEL GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
OVERFIRE AIR JETS
ESP
MULTICYCLONE
BAG HOUSE

Corrosion
COAL QUALITY
OVERLAP GRATE
DUMPING GRATE
VIBRATE GRATE
TRAVEL GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
INDUCED DRAFT FAN
OVERFIRE AIR JETS
ESP
MULTICYCLONE
BAG HOUSE

Smoking
COAL QUALITY

INDUCED DRAFT FAN
Insufficient Capacity
dampers vanes
COAL QUALITY
ESP
MULTICYCLONE
BAG HOUSE
FLY ASH HANDLING

Corrosion
dampers vanes
COAL QUALITY
OVERLAP GRATE
DUMPING GRATE
VIBRATE GRATE

TRAVEL GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIR JETS

Erosion
dampers vanes
COAL QUALITY
OVERLAP GRATE
DUMPING GRATE
VIBRATE GRATE
TRAVEL GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIR JETS
ESP
MULTICYCLONE
BAG HOUSE
FLY ASH HANDLING

Smoking
dampers vanes
COAL QUALITY
OVERLAP GRATE
DUMPING GRATE
VIBRATE GRATE
TRAVEL GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIR JETS
ESP
MULTICYCLONE
BAG HOUSE
FLY ASH HANDLING

SOLID WASTE

FLY ASH SILO
under development

BOTTOM ASH SILO
under development

FLY ASH HANDLING
carbon burnout
plugged lines
COAL QUALITY
OVERLAP GRATE
DUMPING GRATE
VIBRATE GRATE
TRAVEL GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIR JETS
ESP
MULTICYCLONE
BAG HOUSE
INDUCED DRAFT FAN

BOTTOM ASH HANDLING
Carbon Burnout
COAL QUALITY
OVERLAP GRATE
DUMPING GRATE
VIBRATE GRATE
TRAVEL GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIR JETS
Clinkers
COAL QUALITY
OVERLAP GRATE
DUMPING GRATE
VIBRATE GRATE
TRAVEL GRATE
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIR JETS

BOILER

AIR HEATER
under development

OVERFIRE AIRJETS
under development

FLY ASH REINJECT
under development

ECONOMIZER
under development

DUMPING GRATE
Corrosion
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Segregation
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
COAL HOPPER
Pressure Drop

grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS
COAL HOPPER

Uneven Coalbed
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Uneven Ashbed
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Uneven Burning
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Clinkers
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Carbon Burnout
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Warped Grate
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

RECIPROCATING GRATE

Corrosion
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Segregation
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
COAL HOPPER

Pressure Drop
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS
COAL HOPPER

Uneven Coalbed
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Uneven Ashbed
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Uneven Burning
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Clinkers
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Carbon Burnout
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Warped Grate
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

VIBRATE GRATE

Corrosion
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Segregation
grate security

COAL QUALITY
FEEDER DISTRIBUTOR
COAL HOPPER

Pressure Drop
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
COAL HOPPER
OVERFIRE AIRJETS

Uneven Coalbed
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Uneven Ashbed
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Uneven Burning
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Clinkers
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Carbon Burnout
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN

Warped Grate
grate security
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

TRAVEL GRATE

Corrosion
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Segregation
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
COAL HOPPER

Pressure Drop
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
COAL HOPPER
OVERFIRE AIRJETS

Uneven Coalbed
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Uneven Ashbed
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Uneven Burning
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Clinkers
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Carbon Burnout
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

Warped Grate
grate speed
COAL QUALITY
FEEDER DISTRIBUTOR
FORCED DRAFT FAN
OVERFIRE AIRJETS

BOILER OPERATION

Insufficient Capacity
COAL QUALITY
DUMPING GRATE
OVERLAP GRATE
VIBRATE GRATE

TRAVEL GRATE
FORCED DRAFT FAN
OVERFIRE AIRJETS
FEEDER DISTRIBUTOR
INDUCED DRAFT FAN

Not Efficient
boiler flow
COAL QUALITY
boiler opacity
DUMPING GRATE
OVERLAP GRATE
VIBRATE GRATE
TRAVEL GRATE
ESP
FORCED DRAFT FAN
OVERFIRE AIRJETS
FEEDER DISTRIBUTOR
FLY ASH HANDLING
INDUCED DRAFT FAN
MULTICYCLONE
BAG HOUSE

Boiler Opacity
COAL QUALITY

Boiler Turndown
COAL QUALITY

FORCED DRAFT FAN

Smoking
dampers vanes
air supply
COAL QUALITY
DUMPING GRATE
OVERLAP GRATE
VIBRATE GRATE
TRAVEL GRATE
OVERFIRE AIRJETS
FEEDER DISTRIBUTOR

Insufficient Capacity
dampers vanes
air supply
COAL QUALITY
DUMPING GRATE
OVERLAP GRATE
VIBRATE GRATE
TRAVEL GRATE
OVERFIRE AIRJETS
FEEDER DISTRIBUTOR

FEEDER DISTRIBUTOR

Excess Wear
feed plate
plate adjustment
rotor speed
damp coal
COAL QUALITY

Pluggage
plugged blades
damp coal
foreign objects
COAL QUALITY
COAL BUNKER
BELT
RECIP FEED
COAL HOPPER
BUCKET
EN MASSE
CHUTE TO HOPPER
CHUTE FEEDER
VIBRATE FEEDER

Insufficient Capacity
feed plate
plate adjustment
rotor speed
damp coal
COAL QUALITY

Erratic Feeding
feed plate
plate adjustment
rotor speed
damp coal
COAL QUALITY

TUBES FIRE SIDE

Slagging
COAL QUALITY
DUMPING GRATE
OVERLAP GRATE
VIBRATE GRATE
TRAVEL GRATE
OVERFIRE AIRJETS
FEEDER DISTRIBUTOR
FORCED DRAFT FAN

Corrosion
COAL QUALITY
DUMPING GRATE
OVERLAP GRATE
VIBRATE GRATE
TRAVEL GRATE
OVERFIRE AIRJETS
FEEDER DISTRIBUTOR
FORCED DRAFT FAN

Erosion
COAL QUALITY
DUMPING GRATE
OVERLAP GRATE
VIBRATE GRATE
TRAVEL GRATE

- OVERFIRE AIRJETS
- FEEDER DISTRIBUTOR
- FORCED DRAFT FAN

Fouling
- COAL QUALITY
- DUMPING GRATE
- OVERLAP GRATE
- VIBRATE GRATE
- TRAVEL GRATE
- OVERFIRE AIRJETS
- FEEDER DISTRIBUTOR
- FORCED DRAFT FAN

REFRACTORY SURFACES

Slagging
- COAL QUALITY
- DUMPING GRATE
- OVERLAP GRATE
- VIBRATE GRATE
- TRAVEL GRATE
- OVERFIRE AIRJETS
- FEEDER DISTRIBUTOR
- FORCED DRAFT FAN

Corrosion
- COAL QUALITY
- DUMPING GRATE
- OVERLAP GRATE
- VIBRATE GRATE
- TRAVEL GRATE
- OVERFIRE AIRJETS
- FEEDER DISTRIBUTOR
- FORCED DRAFT FAN

Erosion
- COAL QUALITY
- DUMPING GRATE
- OVERLAP GRATE
- VIBRATE GRATE
- TRAVEL GRATE
- OVERFIRE AIRJETS
- FEEDER DISTRIBUTOR
- FORCED DRAFT FAN

WATER TREATMENT

BOILER WATER

Carryover
- drum maint
- level control
- blowdown
- blowdowm maint
- testing
- chemical feed
- CONDENSATE
- SODIUM ZEOLITE
- HOT PROCESS

High TDS
- blowdown
- blowdown maint
- testing
- chemical feed
- CONDENSATE
- SODIUM ZEOLITE
- HOT PROCESS

Corrosion
- pH
- DEAERATOR

Sludge
- blowdown
- blowdown maint
- testing
- chemical feed
- mud blowdown
- CONDENSATE
- SODIUM ZEOLITE
- HOT PROCESS

Tube Failure
- blowdown
- blowdown maint
- testing
- chemical feed
- mud blowdown
- CONDENSATE
- SODIUM ZEOLITE
- HOT PROCESS

Scale
- blowdown
- blowdown maint
- testing
- chemical feed
- CONDENSATE
- SODIUM ZEOLITE
- HOT PROCESS

DEAERATOR

Corrosion
- steam temp
- vent opening
- chemical feed

High Oxygen
- steam temp
- water flow
- steam press
- DA maint
- operation

Water Temp
- water flow
- nozzle maint

CONDENSATE
- Hardness
  - contamination
  - polishing
- Condensate Lack
  - operation
  - excess oxygen
  - chemical feed
  - maintenance
- Corrosion
  - excess oxygen
  - chemical feed
  - maintenance

SODIUM ZEOLITE
- Carryover
  - flow rate
  - resin
  - regenerate cycle
  - leaking valves
  - maintenance
- Pressure Drop
  - flow rate
  - resin
  - maintenance
  - drain manifold
  - control valves
- Corrosion
  - regenerate cycle
  - maintenance
  - salt dose
- Salt Use
  - resin
  - control valves

HOT PROCESS
- Scale Sludge
  - hardness
  - silica
  - backwash cycle
  - suspended solids
  - chemical feedrates
  - auto desludge
  - sludge blowoff
  - pluggage
  - maintenance

BOILER OPERATION
- Carryover
  - backwash
  - chemical feedrates
  - auto desludge
  - pH
  - dissolved solids
  - CONDENSATE
- Corrosion
  - chemical feedrates
  - auto desludge
  - maintenance
  - pH

USACERL DISTRIBUTION

Chief of Engineers
ATTN: CEHEC-IM-LH (2)
ATTN: CEHEC-IM-LP (2)
ATTN: CERD-M
ATTN: CECC-P
ATTN: CERD-L
ATTN: CECW-P
ATTN: CECW-PR
ATTN: CEMP-E
ATTN: CEMP-C
ATTN: CECW-O
ATTN: CECW
ATTN: CERM
ATTN: CEMP
ATTN: CERD-C
ATTN: CEMP-M
ATTN: CEMP-R
ATTN: CERD-ZA
ATTN: DAEN-ZCM
ATTN: DAEN-ZCE
ATTN: DAEN-ZCI

CECPW
ATTN: CECPW-F 22060
ATTN: CECPW-TT 22060
ATTN: CECPW-ZC 22060
ATTN: DET III 79906

US Army Engr District
ATTN: Library (40)

US Army Engr Division
ATTN: Library ('3)

US Army Europe
ATTN: AEAEN-EH 09014
ATTN: AEAEN-ODCS 09014

INSCOM
ATTN: IALOG-I 22060
ATTN: IAV-DEH 22186

USA TACOM 48397
ATTN: AMSTA-XE

US Army Materiel Command (AMC)
Redstone Arsenal 35809
ATTN: DESMI-KLF
Jefferson Proving Ground 47250
ATTN: STEJP-LD-F/DEH
Letterkenny Army Depot
ATTN: SDSLE-ENN 17201
Pueblo Army Depot 81008
ATTN: SDSTE-PUI-F
Dugway Proving Ground 84022
ATTN: STEDP-EN
Tooele Army Depot 84074
ATTN: SDSTE-ELF
Yuma Proving Ground 85365
ATTN: STEYP-EH-E
Tobyhanna Army Depot 18466
ATTN: SDSTO-EH
Seneca Army Depot 14541
ATTN: SDSSE-HE
Aberdeen Proving Ground
ATTN: STEAP-DEH 21005
Sharpe Army Depot 95331
ATTN: SDSSH-E
Fort Monmouth 07703
ATTN: SELFM-EH-E
Savanna Army Depot 61074
ATTN: SDSLE-VAE
Rock Island Arsenal
ATTN: SMCRI-EH
ATTN: SMCRI-TL
Watervliet Arsenal 12189
ATTN: SMCWV-EH
Red River Army Depot 76102
ATTN: SDSRR-G
Harry Diamond Lab
ATTN: Library 20783
White Sands Missile Range 88002
ATTN: Library
Corpus Christi Army Depot
ATTN: SDSCC-ECD 78419

FORSCOM
ATTN: Facilities Engr (12)

National Guard Bureau 20310
ATTN: Installations Div

Fort Belvoir 22060
ATTN: CETEC-IM-T
ATTN: CECC-R 22060
ATTN: Engr Strategic Studies Ctr
ATTN: Australian Liaison Office

TRADOC
ATTN: DEH (13)

US Army Materials Tech Lab
ATTN: SLCMT-DEH 02172

USARPAC 96858
ATTN: DEH
ATTN: APEN-A

AMMRC 02172
ATTN: DRXMR-AF
ATTN: DRXMR-WE

CEWES 39180
ATTN: Library

CECRL 03755
ATTN: Library

USA AMCOM
ATTN: Facilities Engr 21719
ATTN: AMSMC-IR 61299
ATTN: Facilities Engr (3) 85613

Military Traffic Mgmt Command
ATTN: MTEA-GB-EHP 07002
ATTN: MT-LOF 20315
ATTN: MTE-SU-FE 28461
ATTN: MTW-IE 94626

Fort Leonard Wood 65473
ATTN: ATSE-DAC-LB (3)
ATTN: ATZA-TE-SW
ATTN: ATSE-CFLO
ATTN: ATSE-DAC-FL

Norton AFB 92409
ATTN: Library

Engr Societies Library
ATTN: Acquisitions 10017

Defense Nuclear Agency
ATTN: NADS 20305

Defense Logistics Agency
ATTN: DLA-WI 22304

Walter Reed Army Medical Ctr 20307

US Military Academy 10996
ATTN: MAEN-A
ATTN: Facilities Engineer
ATTN: Geography & Envr Engrg

416th Engineer Command 60623
ATTN: Gibson USAR Ctr

Naval Facilities Engr Command
ATTN: Facilities Engr Command (8)
ATTN: Division Offices (11)
ATTN: Public Works Center (8)
ATTN: Naval Constr Battalion Ctr 93043
ATTN: Naval Civil Engr Laboratory (3)
93043

US Army HSC
Fort Sam Houston 78234
ATTN: HSLO-F
Fitzsimons Army Medical Ctr
ATTN: HSHG-DEH 80045

Tyndall AFB 32403
ATTN: AFESC Program Ofc
ATTN: Engrg & Srvc Lab

USA TSARCOM 63120
ATTN: STSAS-F

American Public Works Assoc. 60637

US Army Envr Hygiene Agency
ATTN: HSHB-ME 21010

US Gov't Printing Office 20401
ATTN: Rec Sec/Deposit Sec (2)

Nat'l Institute of Standards & Tech
ATTN: Library 20899

Defense Tech Info Center 22304
ATTN: DTIC-FAB (2)

209
08/93

This publication was reproduced on recycled paper.

☆ U.S. GOVERNMENT PRINTING OFFICE: 1993 — 3510-S/80054

Cont.

SUPPLEMENTARY

INFORMATION

ERRATA SHEET

Your organization recently received USACERL Interim Report FE-93/07, *Development and Use of the Coal-Fired Central Energy Plant Operations Expert System (CEPES)* dated August 1993. Please make the following changes in the report.

a. On the cover: In the upper right corner, change FE-93/07 to FE-93/20.

b. On the perforated user evaluation sheet, change FE-93/07 to FE-93/20.

c. On the report documentation page (SF 298): In block 8, change IR-93/07 to IR-FE-93/20.

Thank you.

ERRATA ADA 274196