Combining Factual and Heuristic Knowledge in Knowledge Acquisition*

Fernando Gomez, Richard Hull, Clark Karr
Department of Computer Science
University of Central Florida
Orlando, Fl 32816
gomez@cs.ucf.edu (407) 823-2764
Bruce Hosken, William Verhagen
Grumman

Abstract

A knowledge acquisition technique that combines heuristic and factual knowledge represented as two hierarchies is described. These ideas have been applied to the construction of a knowledge acquisition interface to OPERA (Expert System Analyst). The goal of OPERA is to improve the operations support of the computer network in the space shuttle launch processing system. The knowledge acquisition bottleneck lies in gathering knowledge from human experts and transferring it to OPERA. OPERA's knowledge acquisition problem is approached as a classification problem-solving task, combining this approach with the use of factual knowledge about the domain. The interface has been implemented in a Symbolics workstation making heavy use of windows, pull-down menus, and other user-friendly devices.

1 Introduction

The goal of OPERA (Expert System Analyst; Adler, 1989) is to improve the operations support of the computer network in the space shuttle launch processing system. The checkout, control and monitor subsystem (CCMS) is a distributed computer network, which integrates software, microcode, display switches and hardware interface devices. OPERA is intended to function as a consultant to the operations staff assigned to each CCMS task. Two basic expert systems form OPERA: the Real Time System Error Manager (RTSEM) and the Problem Impact Analyst (PIA). When an error occurs, RTSEM displays information on this error obtained from a data base of errors. This information, although based on the CCMS message catalog information, contains experiential knowledge that "resides in the head of the human experts, not in texts." The knowledge acquisition bottleneck that the designers of OPERA are presently experiencing is in gathering this knowledge from the human experts and transfering it to OPERA in a form assimilable by the data structures and algorithms of the expert system. OPERA contains about one hundred thirty of these errors, but the actual number of errors in the computer network is greater than one thousand. Hence, OPERA is short in its knowledge base by a factor of ten. The goal of this project is to build a knowledge acquisition interface by means of which a domain expert without knowledge of OPERA or expert systems will be able to transfer his/her knowledge about the computer network errors to OPERA.

*This research is being funded by NASA-KSC Contract NAG-10-0058

OPERA is not a diagnostic expert system whose task is to identify or recognize a problem or error from a set of symptoms and other data. When an error occurs the computer network identifies the error with a code number. Then, OPERA's task is not one of deciding which error has taken place, but rather one of printing the pertinent information concerning that error. This information basically consists of the probable causes of the error, diagnostic advisories (actions to be performed to find out the causes of the error in case they are unclear) and the steps to be taken to correct it, called operational advisories. Table 1 depicts the information about a typical error.

Table 1. Information about a typical error.

Message depicted on the firing room consoles

{ FEP 141 ($$$$) MICROCODE DID NOT RECEIVE AN ACKNOWLEDGE SIGNAL FROM THE I/O ADAPTER, DATA ACQUISITION HAS BEEN INHIBITED. MICAS=$$$$, NSB=$$$$ }

{ ** TERMINAL ERROR FOR THE GSE FEP. THE I/O ADAPTER DID NOT SEND AN ACKNOWLEDGE SIGNAL TO THE MICROCODE DURING THE OPERATION INDICATED BY MICAS.

Probable cause(s):

1. I/O Adapter failed.
2. GSE Option Plane failed.
3. I/O Adapter port on 4-port controller failed.
4. FEP T/R failed.

Operations advisory:

1. Halt CPU, and record CPU registers. Push CPU through recovery.
2. If redundant FEP hasn't taken over, configure another FEP, or $CLAI existing FEP again.
3. $SPRCVE
4. If redundancy isn't available, and original FEP fails to $CLAI, then troubleshoot per following diagnostic advisory.
5. Lookup the MICAS in the microcode listings, and verify the operation being executed at the time of the anomaly.

Diagnostic advisory:

1. $DPLORT LI 5
2. SEQ FEPID1, If errors occur, I/O Adapter thumbin may assist troubleshooting.
3. GSE M02
4. SEQ FEVTR1
   (loop T/R via RCVS). }

When malfunctions occur, messages like this one (in the figure it is displayed in the first set of braces) appear on the firing room consoles of the system engineers monitoring launch activities. The error message designator, FEP 141, indicates the sub-system of the problem (in this case, the Front End Processor), and the error number. Dollar signs are used as place holders for actual hexadecimal addresses. This error occured because the FEP's Input/Output adapter did not send an acknowledgement to the microcode during the operation indicated by the address in the MICAS register. OPERA's response to this message is as follows (OPERA output is the text in the second set of braces). The text, denoted by two asterisks, is a note field obtained from the network system's documentation. This is provided to the system engineer as a convenience so that that he/she does not need to take the time to consult the manuals.

Probable causes for this error are listed next. Causes are listed such that the first probable cause is the most likely, the second is the second most likely, etc. More than one

problem cause may apply to the error. For this particular error, the probable cause is a failed piece of hardware; from the most specialized piece of hardware, the I/O Adapter, down to the most general, the FEP's transmit/receive circuitry.

After the probable causes, the operations advisory is listed. This set of advisories details what should be performed to remedy the situation while the launch is currently underway. Because of this requirement, any action that would jeopardize the launch can not be included in this advisory. Step 4 mandates that if a redundant FEP is not available, the potentially failing FEP is taken off-line and is given a more thorough examination using the diagnostic advisory.

The diagnostic advisory consists of a series of actual diagnostic programs to execute that may determine the cause of the problem. These procedures can not be run on any equipment that is necessary to the continued success of the launch.

However, OPERA has nothing to do with the content of this information. This has been gathered by human experts who are familiar with the computer network. Experts may disagree strongly about the content of this information, but, again, OPERA does not help the experts to gather this information, or to choose between disparaging information. Of course, the value of OPERA as a consultant to the humans who are monitoring the network depends directly on the appropriateness and correctness of the information printed by OPERA.

2 OPERA: A Classification Problem-Solving Task

At first sight, one may think that the task of building a knowledge acquisition interface for OPERA is just one of building a data entry program that will transform the English text about the errors given by the experts into the data structures of OPERA. This clearly will not affect the operation of OPERA. But if the information about the errors is incomplete or incorrect, OPERA would be of very little use to the humans monitoring the computer network. It is clear that the acquisition of the correct knowledge from the experts is essential, if OPERA is to serve a credible role as consultant.

Although OPERA has not been designed as a classification task (Gomez and Chandrasekaran, 1984; Clancey, 1985), and, as a result, there is not a hierarchy of concepts mediating the knowledge about the errors, the knowledge for each error gathered by human experts and printed by OPERA clearly constitutes a classification task. In classification problem-solving, knowledge is organized into a hierarchy of concepts. Top concepts in the hierarchy represent the most general concepts. Lower concepts in the hierarchy are refinements of the upper concepts. The main idea behind this methodology is that concepts, rather than lower level constructs such as rules or procedures, provide the criteria to analyze and organize domain knowledge and acquire knowledge from experts. This translates into the following knowledge acquisition maxim: "Do not ask a domain expert for the rules or procedures he/she uses in analyzing an error or problem, ask him/her for the concepts that he/she uses to conceptualize or classify the error, and 'then you can ask him/her for the rules or procedures." From a problem solving point of view, the hierarchy forces the expert to make explicit the high level conceptual steps (nodes in the hierachy) which he/she will have to consider in determining the probable causes, advisories, and diagnostic steps for a given error. From a knowledge acquisition point of view, approaching this task as a classification task becomes a necessity if the knowledge acquisition interface is going to go beyond

a data entry program, which would merely prompt the user for the probable causes, advisory, etc. The knowledge acquisition interface uses the hierarchy to automatically depict knowledge stored in the upper concepts upon request of the human expert. Then, while a human expert is adding knowledge about an error, he/she may decide to consult knowledge that he/she has stored in the upper concepts. The detailed way in which this is done is explained in section 5.

The knowledge of most domains may be divided into *factual or hard* and *heuristic or soft*. Heuristic knowledge is problem-solving knowledge about a domain. In most cases, there is no concensus among experts about how this knowledge should be organized, what constitutes this knowledge, its activation, etc. This situation is reflected in the saying: "each expert has her/his own book." The trouble shooting knowledge that diagnosticians have clearly falls within this type of knowledge. In contrast to *heuristic* knowledge, *factual* knowledge reflects the way things are. There is little disagreement among experts about what constitutes this type of knowledge. The knowledge that a pathologist has about the human body clearly falls within this category. These two types of knowledge are not dichotomous ones, but rather there is a rich interrelation between them. The heuristic problem-solving knowledge of a diagnostician may have need of the factual knowledge, especially in those cases in which the solution of a problem cannot be obtained directly by applying some right-at-hand rules.

The object of this paper, however, is not to explore the relation between problem-solving on one hand, and heuristic and factual knowledge on the other hand, but rather to investigate the relation between knowledge acquisition and these two types of knowledge. In the next two sections, we show the role that these two types of knowledge play in knowledge acquisition within the domain of the CCMS network.

3 A Factual Knowledge Hierachy for the CCMS Network

In the domain of CCMS network errors, a taxonomy of errors may be built based on the structural components of the network. This classification hierarchy is based on "hard" knowledge and does not follow any heuristic principles. It reflects the way things are. Figure 1 depicts a portion of this hierarchy. The three children of the root node, stand for Front End Processor Messages, Input/Output System Messages and Operating System Integrity Messages. The FEP Messages are in turn divided into four categories: Ground Support Equipment, Launch Data Bus, Pulse Coded Modulation and Uplink messages. These in turn are subdivided into further categories. The IOS2 submessages listed are not terminal nodes, but instead are categories that in turn are subdivided into other categories. Finally, the terminal nodes of this hierarchy will consist of individual error messages.

The relevance of this hierarchy for knowledge acquisition is that knowledge stored under the nodes of this hierarchy may be used by the human expert while she/he is in the process of adding experiential/heuristic knowledge about individual errors. The knowledge stored under these concepts are causes, advisories and corrective steps. This knowledge, as we have been reiterating, is factual and resides in the manuals describing the CCMS network. Some of this knowledge may be very relevant to a domain expert when he/she is entering the causes, advisories, etc. for a specific error. This is similar to the situation of a physician who finds it necessary to consult a medical text book about the functions of organs, while diagnosing a patient.

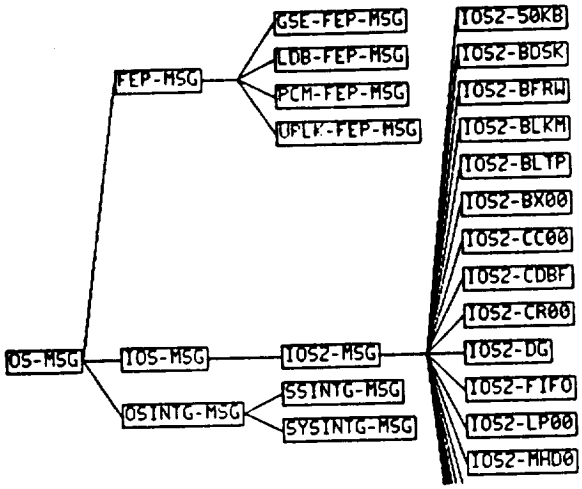

Figure 1: A Portion of the Hard Knowledge Hierarchy of the CCMS Network

The knowledge in the hierarchy is organized following strict inheritance rules. That is, every piece of knowledge in an upper-concept is true of all its subconcepts. As concepts approach the tip nodes, the knowledge becomes more specific. The user may traverse this hierarchy by using the mouse either in a top-down or in a bottom-up fashion. Or he/she may visit any concept without following any predetermined order. The knowledge will be displayed to him/her by the interface. Then, she/he may decide to consult the knowledge or use that knowledge in its entirety or partially (see section 5).

4 A Heuristic Knowledge Hierarchy for the CCMS Network

The place of the concepts in this hierarchy, and the knowledge stored under each concept, do not obey strict or hard rules; rather, they depend on the way in which a given human expert approaches the solution of a problem. As a consequence, heuristic classification hierarchies vary from expert to expert. Figure 2 depicts an elaborated heuristic hierarchy. When a domain expert starts using the Interface, he/she has at her/his disposal the factual hierarchy and an initial heuristic hierarchy similar to the one depicted in Figure 2 but much less detailed. This initial heuristic hierarchy is provided to the expert as a basis for him/her to start building his/her own hierarchy. Of course, he/she may disagree with the structure and/or content of the hierarchy, and as a consequence he/she may decide to change this initial hierarchy to conform to his/her view of the problem-solving knowledge.

In building this hierarchy, an expert is instructed to proceed in a top-down manner. The Interface walks a domain expert who is unfamiliar with the interface through the following steps:

- What are the most general categories (software, hardware, etc.) that come to your mind when the error, say, FEP-132 occurs"?
- Once you have determined that the error is, say, a software problem, which subcategories within the software do you think about?
- Which advisories and/or causes are known for a given category?

Once the domain expert has acquired some familiarity with the interface, the knowledge acquisition process concentrates on entering advisories about individual errors. During this process, the domain expert may decide to modify the heuristic hierarchy, by adding new links, altering existing links or deleting or adding advisories stored under the nodes. But, in most cases, the expert may use the knowledge stored by him/her in the heuristic hierarchy and in the factual hierarchy in order to build knowledge about individual errors. This is explained in detail in the section below.

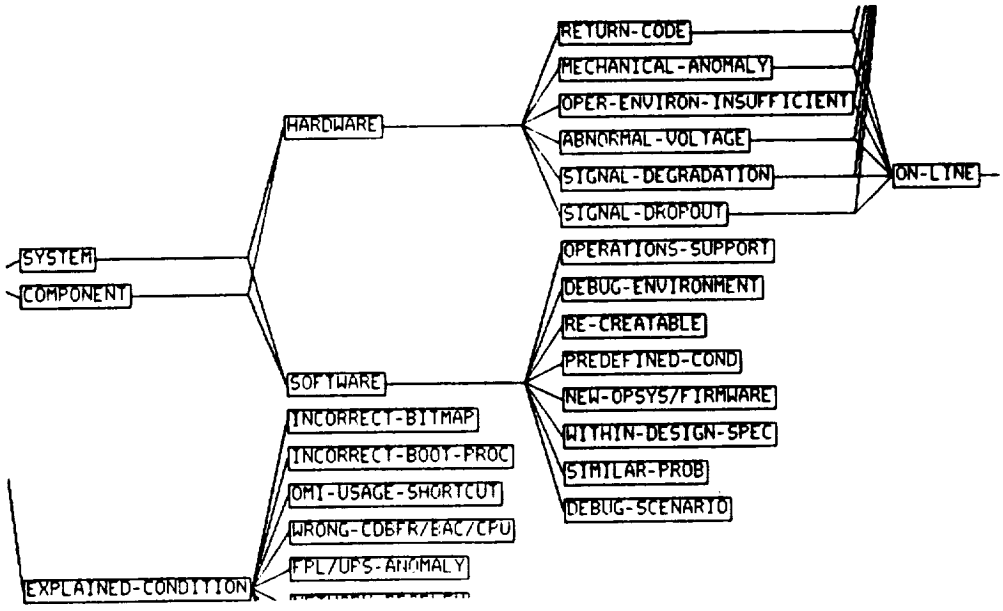

Figure 2: A Portion of an Elaborated Heuristic Hierarchy for the CCMS Network

5 A Walk Through The Interface

The interview process has two phases; the first is the construction or modification of the domain expert's error classification hierarchy, and the second is the generation of OPERA advisories. These two phases need not be strictly ordered and can be interleaved, i.e., domain experts are not forced to construct their final classification hierarchies before any advisories are created, but rather they are free to change their hierarchies at any time. To minimize the amount of startup time and to give the domain expert an idea of what we are after, we provide an elaborated error classification hierarchy designed from Grumman systems engineer Bill Verhagen's hierarchy (see Figure 2). This hierarchy provides systems engineers unfamiliar with the interface a starting point from which they can begin to coalesce their experiential knowledge of the CCMS network. While initial interviews require some instruction and typically last several hours, a given interview session can be accomplished in as little as 30 minutes, depending on the amount of information to be elicited.

5.1 Creating and Editing OPERA Advisories

The primary goal of the interface is the acquisition of knowledge about error messages. Currently the data collected are exported to the OPERA system in the form of advisories enumerating the probable causes, operational advisories, and diagnostic advisories for specific errors generated by the CCMS network. The first step in creating an advisory is choosing the error message to describe. The user is presented a menu of error messages that were previously specified by the Knowledge Engineer. The error messages on this menu reflect those errors that the Knowledge Engineer is in-

terested in collecting information about. The user is free to choose any message on the menu.

5.1.1 Placing Errors in the Heuristic Hierarchy

Once an error message has been chosen, the user is asked to place the error within his current heuristic hierarchy. To aid the user in this task, the interface provides help in the form of status register decodings and notes provided by the Knowledge Engineer.

Given this help, the user should be able to place the error in his heuristic hierarchy. Placing the error within the heuristic hierarchy is a matter of specifying which node is to become the error's parent. If a suitable parent does not exist in the hierarchy, the user is given a chance to create and place the parent in the hierarchy at that time. It may be, however, that the parent of the parent (grandparent of the original error message) does not exist in the hierarchy. Again, the user may create and place the grandfather in the hierarchy. This process can continue as long as necessary until the chain of new error categories can be linked to a node in the hierarchy (see Figure 3). Once the error has been inserted into the hierarchy, the interface gives the domain expert the opportunity to create the list of probable causes, operational advisories, and diagnostic advisories associated with the error.

5.1.2 Causes, Operational Advisories, and Diagnostic Advisories

Adding and editing cause and advisory information is quite simple. A pop-up menu is presented that allows the user to pick between changing probable causes, operational advisories, or diagnostic advisories. Once an area has been selected the interface allows the user to: add new lines of information, edit specific lines, rearrange the order of lines, or delete lines. Each line consists of free-form text keyed in by the user or mouse-selected from default information contained in the factual and heuristic hierarchies. Figure 4 shows the interface screen during the entry of probable cause data for the FEP 132 error.

5.1.3 Using the Default Information

As mentioned above, the user may create advisories by selecting text, via the mouse, from the factual and heuristic hierarchies. The texts available to be selected are those default advisories constructed by the domain expert and knowledge engineer and stored in the heuristic and factual hierarchies. When the user is to the point of entering in a line of text of the probable causes, operational advisory, or diagnostic advisory, the system displays the default advisories in the lower right window pane of the interface screen (see Figure 5). How the interface determines which default advisories are displayed in this pane is described below.

First, the interface must determine whether the user has chosen to display information from the factual hierarchy, from his own heuristic hierarchy, or both. This determination is based on the option the user has chosen using the *Select Inheritance* command (the default option is to show both). If the user has chosen to display both or has simply taken the default, the interface will collect default advisories from both hierarchies, displaying the user's own defaults at the top of the window. This is done under the assumption that the expert will feel that his own default advisories are more relevant than those of the knowledge engineer. If the user chooses one or the other type of knowledge, the interface will collect only the default advisories from the corresponding hierarchy.

Given that the system knows which hierarchy or hierachies to collect the default advisories from, the interface then uses the hierarchy's structure to decide which advisories to display. For example, suppose the FEP 132

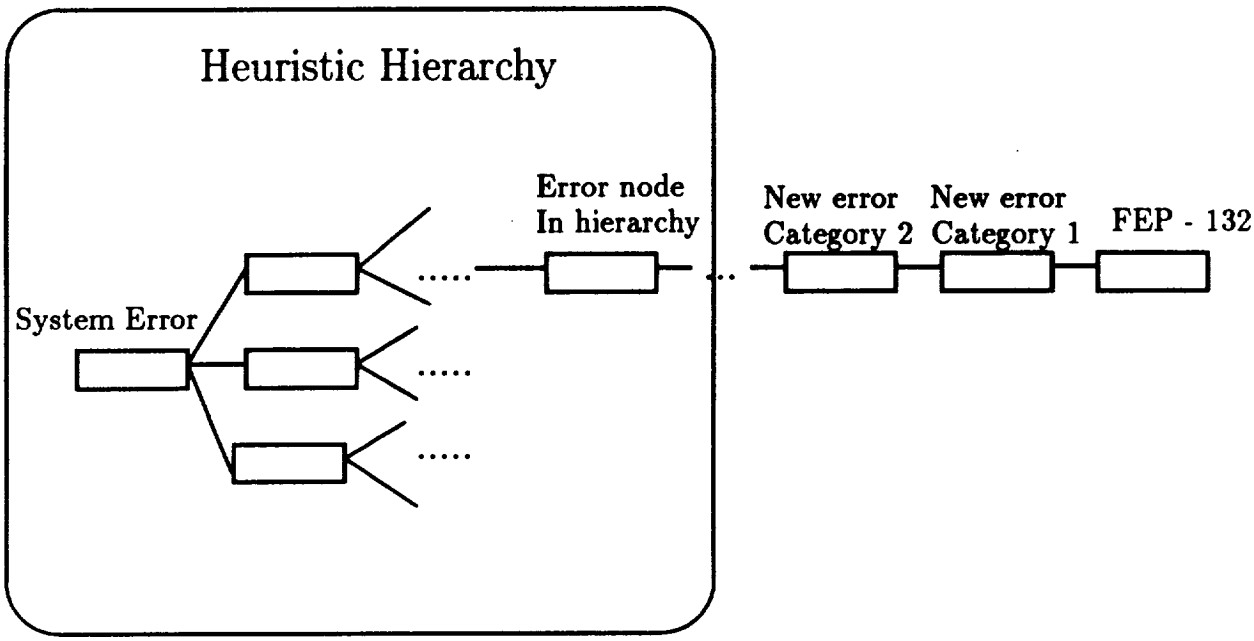

Figure 3: Adding Error Message to the Heuristic Hierarchy

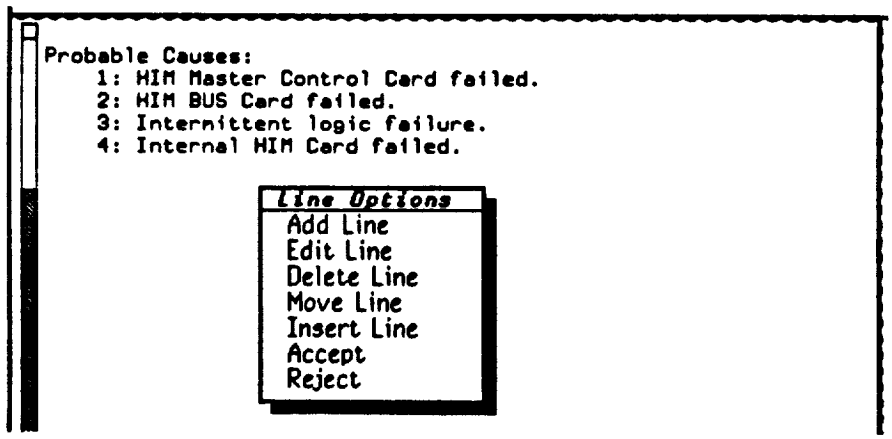

Figure 4: Entering Probable Cause Information

error used above was classified as a mechanical anomaly and the user had chosen to use default advisories from his own hierarchy. The interface would begin collecting default advisories from the mechanical anomaly node in the hierarchy. These advisories are the most specific and will be displayed at the top of the window pane. The interface then traverses up the hierarchy to the ancestors of the mechanical anomaly node. The default advisories for each ancestor are collected and added to the list of advisories to be displayed after the advisories found in mechanical anomaly. This process continues until the root node is reached along each ancestral path. The by-product of this process is a list of all the advisories from the parent of the error we are describing up to the root of the hierarchy in order from most specific to most generic.

Once the advisories have been collected, the user can select them using the mouse and include them, as is, in his description, or modify them in anyway he chooses. This means that the domain expert does not need to store "perfect" advisories, but can store advisory templates that can be modified as necessary. This greatly enhances the flexibility of the interface.

Another enhancement stems from the fact that the default advisories themselves are stored in a hierarchical structure. This allows different levels of default information to exist and be used by the domain expert. One example we encountered where this was useful was in the specification of diagnostic advisories. Typically, a diagnostic advisory includes reference to sequences of diagnostic programs that should be executed. The case may be, however, that an entire diagnostic sequence need not be run but only several of its sub-tests. To accommodate this situation the domain expert may specify that a default advisory has its own children. This allows the system to recognize that an advisory detailing a sequence of tests may have children that are the actual sub-tests. For example, the diagnostic sequence SEQ CP1 - CPU DIAGNOSTIC PART 1, has the following nine sub-tests:

```
"TST02 - XORB TEST"
"TST03 - REGISTER ADDRESSING TEST"
"TST04 - R2 DATA INTEGRITY TEST"
"TST05 - BLM/BRX TEST"
"TST06 - R3-R15 DATA INTEGRITY TEST"
"TST07 - ABRB TEST"
"TST08 - NOP TEST"
"TST09 - LDX TEST"
"TST10 - IBR TEST"
```

The user can chose the string "SEQ CP1 - CPU DIAGNOSTIC PART 1" to include in his advisory by clicking the left button of the mouse when the mouse cursor is above this text, or he can see the associated sub-tests by clicking the right button. If there were sub-sub-tests, these could be viewed by clicking the right button again. Returning to a higher level is accomplished by clicking the middle button of the mouse. In summary, clicking the left mouse button selects the text under the mouse cursor, clicking the middle button takes the user up one level in the advisory hierarchy, and the right button takes the user down one level in the advisory hierarchy.

When the user has finished entering his description of an error, he may choose to save it in his hierarchy or simply abort. Saving the information amounts to creating the necessary frames and their fillers in the expert's hierarchy. If the user does not abort, the expert's hierarchy is redisplayed with the new error node included. This concludes the discussion of the error description process.

5.2 Modifying the Structure of Heuristic Hierarchies

Maintaining the domain expert's error classification hierarchy is one of the most important tasks of the interface. Several powerful options have been implemented to allow the

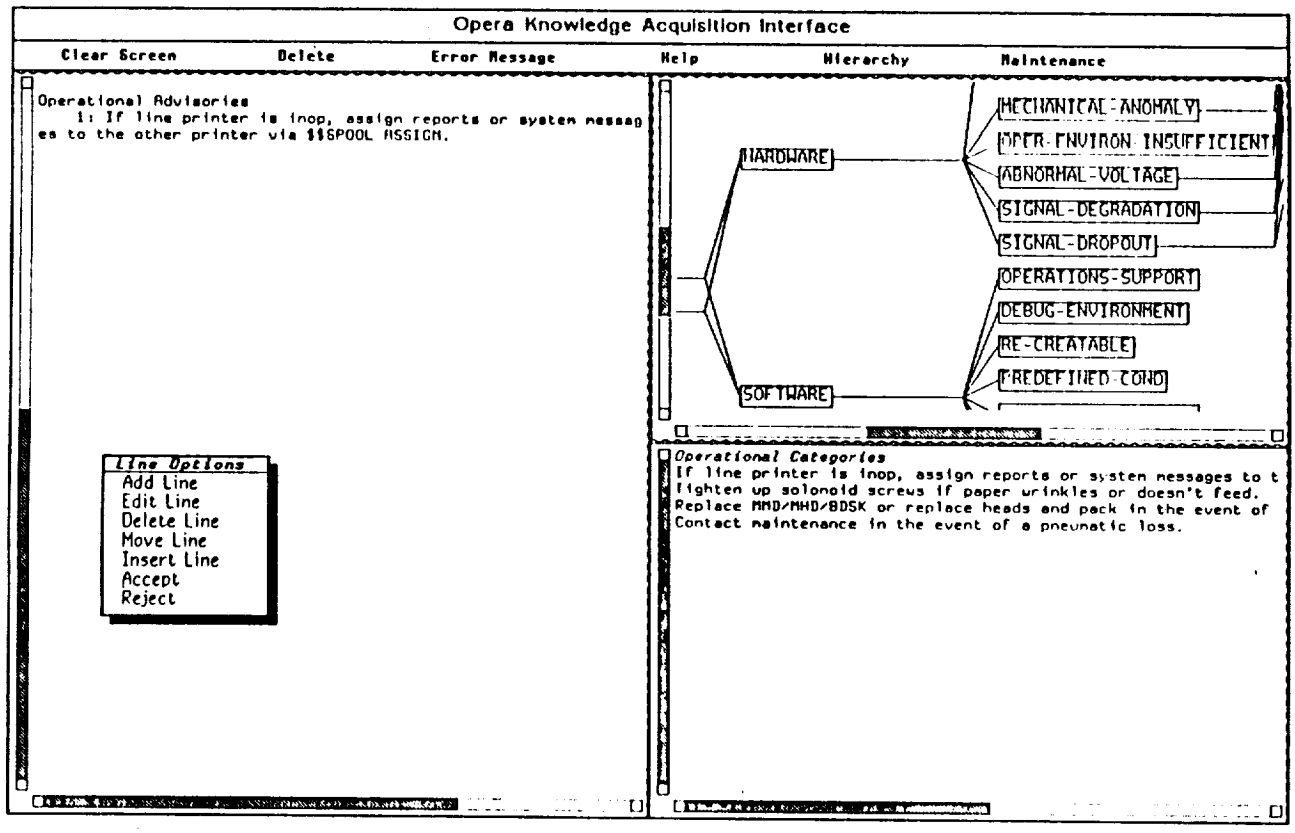

Figure 5: Interface Main Screen with Default Advisories

user to quickly and easily change the structure of his/her hierarchy. These options include adding new error categories, adding and deleting links between errors or error categories, and moving sub-hierarchies from one place in the hierarchy to another.

5.2.1 Adding New Error Categories

New error categories are added to the domain expert's error classification hierarchy using the *Add Error Category* option. This option allows the user to create a new error category and place it in the hierarchy. The user is prompted for the name of the new category and the category that is to be its parent. The parent category must exist in the hierarchy and may be given either by typing its name via the keyboard or by clicking on its graphical representation using the mouse (all the nodes in the domain expert's hierarchy are mouse selectable). After this information is given, the interface redisplays the hierarchy reflecting the addition of the new category.

5.2.2 Adding and Deleting Links

Links or inheritance paths can be added to and deleted from the expert's hierarchy using the *Add New Link* and *Delete Link* options. In the case of adding a new link, the user is prompted for a child category and a parent category. The system checks to see if the parent node is a descendant of the child node, and if it is, the attempt is aborted. This constraint guarantees that cycles will not be created by adding new links. Deleting a link is similar to adding a new one. The user is prompted for the parent and child nodes that define the end-points of the link. Assuming that the parent and child nodes given indicate an existing link, the system proceeds to remove the link and redisplay the hierarchy. Because the error classification hierarchy is a tangled hier-

archy, child nodes may have multiple parents and deleting any one of them does not effect the child node. Deleting the last link between the child and the rest of the hierarchy, however, effectively removes the child and any of its descendents that are not attached to the rest of the hierarchy through their own links (see Figure 6).

5.3 The Restructure Hierarchy Option

Should the user wish to radically restructure his/her heuristic hierarchy, the *Restructure* option can be used. This option allows the user to move sub-hierarchies from one parent node to another. The user is prompted for the root node of the sub-hierarchy he would like to move and its new parent. If the root node has several existing parents, a menu containing the names of these parents is displayed and the user is expected to click on the name of the parent node that he wishes to break away from. Constraints involving the creation of cycles and validity of node names are enforced to prevent corruption of the hierarchy. When the constraint checks are passed the hierarchy is redisplayed.

5.4 Modifying Default Information Within Heuristic Hierarchies

Information stored in the interior nodes (error category nodes) of the heuristic hierarchy is modified using the *Edit Category Data* option. With this option, users can explain the reasoning behind their classifications and create or edit default operational and diagnostic advisories. Default advisories containing the domain expert's experiential knowledge are displayed and used during the creation of OPERA advisories. Each default advisory and the expert's reasoning about his classification consists of one or more lines of text.

5.5 Specific Tools For OPERA

A special maintenance menu is provided to the knowledge engineer so that he can: add new errors to be described to the system, dump the collected advisories in a format readable by OPERA, and change the structure of the factual hierarchy. To add a new error, the knowledge engineer must enter the information that the domain expert is going to need before he can describe the error. This includes the status register values of any register inserts, the actual text of the error message, the formats of the register inserts, the notes from the CCMS documentation about this error, and the placement of the error within the factual hierarchy.

Dumping the collected advisories to OPERA is done by simply clicking a menu option. The user is then asked for the filename of the dump file. The data output is in a pseudo-LISP form that OPERA can directly input. Data may be dumped at any time and as many times as needed. Changing the structure of the factual hierarchy is handled similarly to changing the structure of the expert's hierarchy. The knowledge engineer uses the same restructuring commands that are available to the domain expert for changing heuristic hierarchies.

6 Design of the Two Hierarchies

The basic unit of information in our representation is a frame representing a single node within a hierarchy. A node (frame) may represent a root, a leaf, or an internal node within a hierarchy. Each node is known by a "node name" that is specified as an ASCII string (without spaces) by the expert creating the node. Associated with each node are two types of information: first, the information that details the hierarchy (i.e. the expert) to which the node belongs, its parent

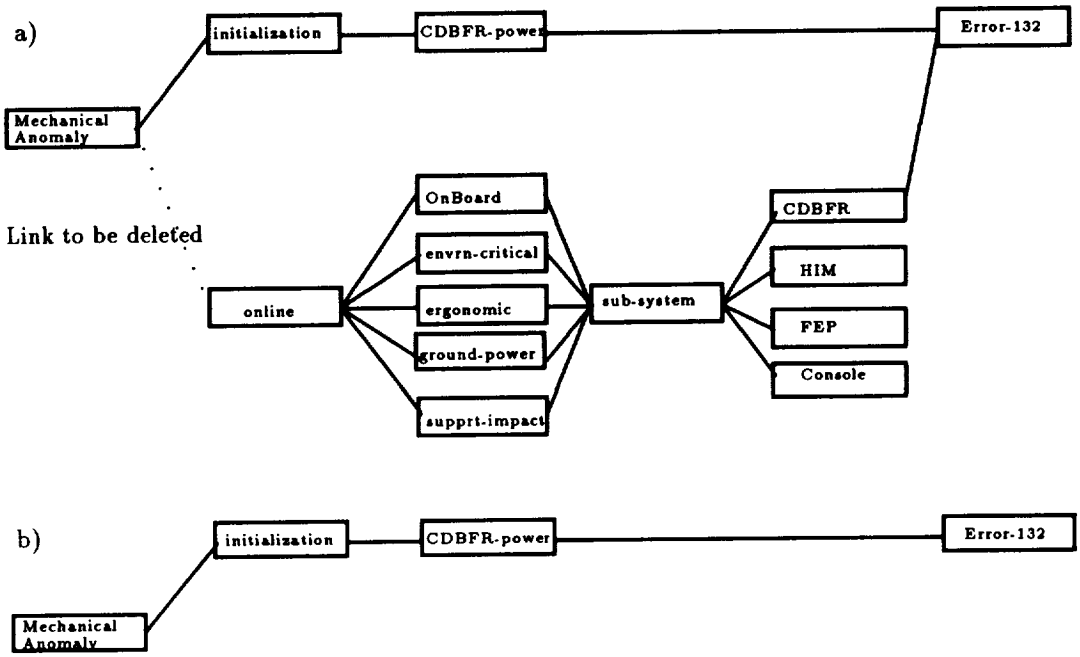

Figure 6: Deleting Links Between Nodes in the Heuristic Hierarchy: a) before, b) after

nodes, and its child nodes within the hierarchy; and second, the domain information that node stores within that hierarchy.

The frame structure for specifying nodes is as follows. A node is identified by a "name". The information detailing a node's position and connections within a hierarchy are stored under the property "*inherit*" while the domain information is stored under the property "*frame*". Within each property, the top level slot names the expert creating the node. This expert name uniquely identifies the hierarchy to which the node belongs. Within the "*inherit*" property, under the expert name are two slots, "children" and "parents", that identify the links within this expert's hierarchy. In frame notation, a node is defined as:

```
( <node-name>
  ( *frame*
    ( <expert-name1>
      ( <domain-data  ...))
    ( <expert-name2>
      ( <domain-data  ...))
              . . .
    ( <expert-nameN>
      ( <domain-data  ...)))
  ( *inherit*
    ( <expert-name1>
      ( children ( <node-name> ...))
      ( parent ( <node-name>)))
    ( <expert-name2>
      ( children ( <node-name> ...))
      ( parent ( <node-name>)))
          . . .
    ( <expert-nameN>
      ( children ( <node-name> ...))
      ( parent ( <node-name>)))))
```

The OPERA Interface is built upon a variety of primitive functions that control access to information within the entire data structure. An expert is limited to his/her hierarchy and the factual hierarchy defined by the knowledge engineer. The system's primitives control the inheritance of information within an expert's hierarchy and from the factual hierarchy to the expert's hierarchy. An expert is unaware that the data structure (frame) storing his information also stores other experts' information. Duplicate names for internal nodes

within heuristic hierarchies create no problems for keeping the domain experts' information separate.

In the OPERA domain, heuristic hierarchies share leaf nodes describing individual errors. All information entered about an error by any number of experts is recorded within the one frame describing the individual error. Contradictory and conflicting information among experts is kept segregated within each expert's subframe. In this fashion, the knowledge structure supports multiple, conflicting views of the domain without destroying the integrity of any expert's information.

A priori knowledge about the domain is stored in a hierarchy with the expert name: "FACTUAL". The system's primitives recognize "FACTUAL" as identifying the factual hierarchy. Information within the factual hierarchy is available to domain experts as they define their hierarchies and enter specific information about individual errors. The system uses the factual hierarchy to display suggestions and/or possible text for the expert to consider, modify, and incorporate in his/her hierarchies. The system prevents experts from altering the factual hierarchy.

This knowledge structure with its primitives allows multiple experts to define heuristic hierachies (which can be tangled) reflecting their view of the domain, to interact with an *a priori* knowledge base without contaminating it, and to enter information into a single data representation without fear of corrupting information entered by other experts. At the same time, all information is available to the knowledge engineer in a consolidated form requiring little manipulation to make sense of the information.

7 Conclusions and Future Research

A knowledge acquisition framework that makes use of factual and heuristic knowledge has been described. This technique has been applied to the acquisition of advisories and probable causes about errors that occur in the computer network controlling the space shuttle launch processing system. The knowledge acquisition interface is currently running on a Symbolics 3653 under version 8.1 of the Genera operating system. The implementation is in the process of being converted to run under CLIM (Common Lisp Interface Manager) in Allegro Common Lisp on a SUN platform. SUN workstations are much more common at the Space Center than Symbolics machines, and this migration should provide systems engineers with greater accessibility to the interface. Information about approximately 50 error messages has already been collected from 7 experts. While these error messages are primarily concerned with the Front End Processing sub-system, we are expanding our efforts to recruit experts with knowledge about the other sub-system messages.

Although we have applied these ideas to the construction of a knowledge acquisition interface for OPERA, and some of the components of the interface are OPERA dependent, (e.g., the final dumping of the advisories into OPERA data structures), the interface has a range of application that goes beyond OPERA. In principle, any domain that can be analyzed into a factual and a heuristic hierarchy as described in the body of the paper falls within the scope of the interface. Of course, this description is very general and some domains are going to have idiosyncracies that will require special mechanisms to handle them. However, if one stays within the area of determining the probable causes and advisories of computer network errors, then the interface can be used in many subdomains with very

minor modifications.

Table 2. A Portion of the Data Dumped From the Interface to OPERA.

```
** MSG-CAUSES: **
((FILTERS) 1.  GSE Option Plane has
               failed)
((FILTERS) 2.  GSE FEP Option Plane
               microcode has failed)
((FILTERS) 3.  GSE FEP 4-port con-
               troller has failed)
((FILTERS) 4.  GSE FEP CPU failed)

** DIAGNOSTIC-ADVISORY: **
((FILTERS) 1.  MO2 on the data
               acquisition plane)
((FILTERS) 2.  SEQ FEPI01)
((FILTERS) 3.  SEQ CP1, CPU Diag-
               nostic Part 1)
((FILTERS) 4.  SEQ CP2, CPU Diag-
               nostic Part 2)
((FILTERS) 5.  SEQ OPD, Option Plane
               Diagnostic)
((FILTERS) 6. $DPLORT LI 4)
((FILTERS) 7. $DPLORT LI 5)

** INSERT-FORMAT: **
(INSERT1 ASCII CPU-NAME-INTERPRET
 CPU-NAME)
(INSERT2 HEX MDT-CDT-PTR-DECODE
 MDT-CDT-PTR)

** OPS-ADVISORY: **
((FILTERS) 1.  Note the MDT/CDT
               Pointer Address in
               the error message)
((FILTERS) 2.  If ACTIVE GSE FEP,
               verify that STANDBY
               GSE FEP is O.K.)
((FILTERS) 3.  Halt the CPU and
               record CPU regs)
((FILTERS) 4.  Perform applicable
               data retrieval progs
               $SPRCVE, $SPBLOK,
               $SPSNPR)

** MSG-TEXT: **
  FEP 142
  INSERT1
  MICROCODE DETECTED INVALID
  MEASUREMENT/COMMAND TYPE CODE,
  DATA ACQUISITION INHIBITED,
  MDT/CDT PTR =
  INSERT2
```

We are planning to incorporate in the interface some of the ideas described in (Gomez and Segami, 1990; Gomez and Segami, 1991). We are targetting two possible applications of these ideas. One is the construction of the factual hierarchy from natural language input. The other is to use natural language combined with some elicitation techniques (Boose and Bradshaw, 1987) to build the heuristic hierarchy during the first stages of its construction by domain experts unfamiliar with the interface. The final result will be the construction of a generic knowledge acquisition interface incorporating the automatic construction of hierachies from natural language input.

References

Adler, R., Heard, A., & Hosken, B. (1989). An Expert Operations Analyst (OPERA) for a Distributed Computer Network. *AI Systems In Government (AISIG).* Washington D.C.

Boose, J. & Bradshaw, J. (1987). Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for expert systems. *International Journal of Man-Machine Studies*, 26, 3-28.

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

Gomez, F. & Chandrasekaran, B. (1984). Knowledge organization, and distribution for medical diagnosis. In W. Clancey

& E. Shortliffe, Eds. *Readings in Medical Artificial Intelligence.* Reading, MA: Addison-Wesley.

Gomez, F. & Segami, C. (1990). Knowledge acquisition from natural language for expert systems based on classification problem-solving methods. *Knowledge Acquisition*, 2, 107-128.

Gomez, F. & Segami, C. (1991). Classification Based Reasoning. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3), 644-659.

Information Management