Comprehension and Retrieval of Failure Cases in Airborne Observatories*

Sergio J. Alvarado
Kenrick J. Mock

Computer Science Department
University of California, Davis
Davis, CA 95616

(916) 752-7004
alvarado@cs.ucdavis.edu
mock@cs.ucdavis.edu

Abstract

This paper describes research dealing with the computational problem of analyzing and repairing failures of electronic and mechanical systems of telescopes in NASA's airborne observatories, such as KAO (Kuiper Airborne Observatory) and SOFIA (Stratospheric Observatory for Infrared Astronomy). The research has resulted in the development of an experimental system that acquires knowledge of failure analysis from input text, and answers questions regarding failure detection and correction. The system's design builds upon previous work on text comprehension and question answering, including: knowledge representation for conceptual analysis of failure descriptions, strategies for mapping natural language into conceptual representations, case-based reasoning strategies for memory organization and indexing, and strategies for memory search and retrieval. These techniques have been combined into a model that accounts for: (a) how to build a knowledge base of system failures and repair procedures from descriptions that appear in telescope-operators' logbooks and FMEA (failure modes and effects analysis) manuals; and (b) how to use that knowledge base to search and retrieve answers to questions about causes and effects of failures, as well as diagnosis and repair procedures. This model has been implemented in FANSYS (Failure ANalysis SYStem), a prototype text comprehension and question answering program for failure analysis.

1. Introduction

This paper describes research dealing with the computational problem of analyzing and repairing failures of electronic and mechanical systems of telescopes in NASA's airborne observatories. To understand the reasons underlying this endeavor, consider the process of failure analysis in NASA's KAO (Kuiper Airborne Observatory), an airborne observatory for infrared astronomy (Thronson and Erickson, 1984). First, diagnosing and repairing failures of electronic and mechanical systems in KAO are time-consuming processes that can delay or cancel flights, or decrease valuable in-flight observation time. Although these processes depend on how failure-analysis information is stored and accessed, knowledge of failures in KAO is highly experiential and recorded in telescope-operator's logbooks that must be searched sequentially to determine whether given failures have occurred on previous occasions. Furthermore, even though KAO has a database of procedures for on-the-ground maintenance, this database is not designed to aid in diagnosing and repairing in-flight failures. Finally, given that NASA plans to replace KAO by building SOFIA (Stratospheric Observatory for Infrared Astronomy) (Erickson et al., 1991), then KAO can be viewed as a framework for building and experimenting with a failure-analysis system that can later be adapted for use within SOFIA.

This research has resulted in the development of an experimental system that acquires knowledge of failure analysis from input text, and answers questions regarding failure detection and correction. The

*This research was supported in part by a NASA-ASEE Faculty Fellowship to the first author, and by the NASA-Ames Research Center, Moffett Field, California, under Interchange No. NCA2-721.

model accounts for: (a) how to build a knowledge base of system failures and repair procedures from descriptions that appear in telescope-operators' logbooks and FMEA (failure modes and effects analysis) manuals; and (b) how to use that knowledge base to search and retrieve answers to questions about causes and effects of failures, as well as diagnosis and repair procedures. This model has been implemented in FANSYS (Failure ANalysis SYStem), a prototype text comprehension and question answering program (Alvarado, Braun, and Mock, 1993) initially created for the analysis of failures in the data management system (DMS) of NASA's Space Station.

FANSYS combines domain knowledge, strategies for natural language processing, and strategies for memory search and retrieval in order to understand failure descriptions and answer failure-related questions. Extending the capabilities of FANSYS into the domain of telescope-system failures has required: (1) collecting telescope design and operation logbooks and manuals to develop a model of the telescope system and a library of failure cases; (2) interviewing telescope operators and mission directors to elucidate failure modes, failure causes, failures effects, and procedures for failure detection and correction; (3) analyzing the conceptual content of the information in documents and interviews to model the knowledge structures and processes underlying the diagnosis and repair of telescope failures; and (4) implementing these structures and processes within FANSYS. As a result of these knowledge-engineering tasks, FANSYS can handle failure cases regarding a number of KAO's telescope components and systems, including the compressors, the oscillating secondary mirror (OSM), the airborne data acquisition and management system (ADAMS) and its facility interface (FI), the universal power system (UPS), the telescope inertia positioning system (TIPS), the telescope data acquisition and display system (TDADS), the tracker system, and the compensation system.

2. Comprehension and Retrieval of Telescope Failures in FANSYS

The problem of analyzing and diagnosing failures that occur in mechanical or electronic devices has been addressed by other researchers in artificial intelligence and computer science. Some of their work has been concerned with the development of truth-maintenance systems (de Kleer and Williams, 1987; Struss, 1988), expert systems (Reed and Johnson, 1990), connectionist expert systems (Liu et al., 1991), digraph-based analysis systems (Iverson and Patterson-Hine, 1990; Patterson-Hine and Iverson, 1990; Stevenson, Miller, and Austin, 1991), and case-based reasoning systems (Acorn and Walden, 1992; Hammond and Hurwitz, 1988; Redmond, 1990; Simoudis and Miller, 1990). Among these systems, there are three case-based reasoning systems that perform tasks related to those of FANSYS. First, CELIA (Redmond, 1990) learns cases regarding automobile failures, and can adapt prior solutions to solve new problems by reasoning about plans and providing justification for the new solutions. Second, CASCADE (Simoudis and Miller, 1990) retrieves failure cases regarding the VMS operating system by using a feature network and a validation network that provide access to relevant cases. Finally, Compaq's SMART system (Acorn and Walden, 1992) uses a case-based reasoning engine developed by the Inference Corporation to store and retrieve failure cases regarding Compaq's hardware and software.

Although these case-based reasoning systems deal with the problem of failure diagnosis, they do not address the problem of dynamically building the conceptual representations of failure cases from textual input, and storing those representations according to their conceptual similarities and differences. In contrast, FANSYS involves the use of techniques for parsing input failure descriptions into a conceptual network of cases that maintains the context for subsequent question answering. These techniques include: knowledge representation for conceptual analysis of failure descriptions (Alvarado et al., 1993); strategies for mapping natural language into conceptual representations (Alvarado, 1990; Riesbeck and Martin, 1986); case-based reasoning strategies for memory organization and indexing (Riesbeck and Schank, 1989); and strategies for memory search and retrieval (Dyer and Lehnert, 1982; Lehnert, 1978; Kolodner, 1984).

Comprehension and retrieval of telescope failures in FANSYS involves four major tasks: (1) applying domain-specific knowledge (i.e., telescope-system knowledge); (2) mapping input text into conceptual structures that compose the internal representation of failure cases; (3) representing and

indexing failure cases in memory by creating a knowledge base of cases; and (4) using the knowledge base of cases to answer questions regarding failure causes and effects, as well as failure detection and correction procedures. Input descriptions are segments of telescope-operator's logbooks and manuals, and contain the essential wording of the original descriptions. Here "essential" means that the original case descriptions have been edited in order to remove parts that do not fall within the following categories: failed component or system, failure mode, failure cause, failure effect, failure-detection procedure, failure-correction procedure, flight number, and flight date.

Below is an actual failure description processed by FANSYS. The description is labeled UPS-FAILURE-1 and is a fragment of a UPS failure description from a design report by NSI (1993), a contractor at NASA-Ames Research Center. The case involves a failure detected when light-emitting diodes (LEDs) on the UPS front panel indicate that the UPS is on utility power and its backup battery is disabled.

UPS-FAILURE-1
ITEM NAME: UPS (backup battery).
FAILURE MODE: Disabled UPS (backup battery) — failure during operation.
FAILURE DETECTION/VERIFICATION: The telescope operator realizes by examining the UPS LED panel (UPS-on-Utility on and Backup-Disabled on) that the UPS (backup battery) is nonoperational.
CORRECTIVE ACTION: The telescope operator switches the UPS (backup battery) on.

Text comprehension in FANSYS requires applying domain knowledge in order to build the conceptual representation of input failure descriptions. This domain-specific knowledge includes representations for each of the following classes of concepts: (1) entities, or KAO components (e.g., the UPS); (2) states, or properties of entities (e.g., the state of being nonoperational); (3) events composed of an action performed by or on a entity, the action's precondition, and the result of the action (e.g., the event of powering up the backup battery includes the power-up action, the precondition that the power be off, and the result that the power be on); and (4) procedures that consist of sequences of events involved in failure detection or correction (e.g., the read-out procedure is a detection procedure in which a failure is determined by checking indicators on a LED panel or a computer monitor).

FANSYS contains knowledge of natural language in order to map words and phrases into their appropriate conceptual structures. In FANSYS, text comprehension is performed using the case-based parsing techniques developed by Riesbeck and Martin (1986). FANSYS reads input text and questions in a left-to-right manner, one word or phrase at a time. As each word and/or phrase is read, the lexicon of FANSYS is accessed in order to identify the conceptualization underlying that word or phrase. When a lexical item is recognized, FANSYS activates an instance of the item as well as instances of patterns that encode the syntactic and semantic conditions associated with the lexical item and its underlying conceptualization. Then, FANSYS tests all pattern instances and uses those whose conditions are satisfied. The result of this process is the construction of the conceptual representation of the input.

During text comprehension, FANSYS dynamically builds the knowledge base of failure cases as a result of *remindings* (Schank, 1982) that occur when input descriptions of similar cases are read and their representations are indexed in memory using the case components. As such, text comprehension in FANSYS accounts for: (1) how cases are reorganized and how generalizations are created in memory when remindings occur; (2) how memory search and retrieval processes adapt to memory changes that occur during case indexing; and (3) how cross-contextual remindings occur when reading input failure descriptions involving different telescope components and systems. To model this reminding process, FANSYS organizes failure cases using a hierarchical-organization scheme based on the memory model developed by Kolodner (1984).

Finally, question answering in FANSYS involves using a model of memory search and retrieval developed as an extension to the question-categorization scheme of Lehnert (1978) and the retrieval heuristics of Dyer and Lehnert (1982) and Kolodner (1984). Within this model, answering a question requires analyzing the conceptual content of the question into a set of failure-case components that include

the information given and the information requested in the question. Based on this analysis, search and retrieval processes select memory indices, and traverse memory links in order to locate appropriate instances of knowledge structures to be retrieved. For example, consider the following question answered by FANSYS regarding the failures occurring in one of KAO's flights:

What is the failed component and the failure correction procedure when the flight number is 93-044?

[1] FI (FI-TIPS LINK).

[2] CORRECTION PROCEDURE: THE TELESCOPE OPERATOR LOGICALLY SHUTS DOWN THE TIPS. THE TELESCOPE OPERATOR LOGICALLY STARTS UP THE TIPS.

[3] CORRECTION PROCEDURE: THE TELESCOPE OPERATOR LOGICALLY SHUTS DOWN THE FI (FI-TIPS LINK). THE TELESCOPE OPERATOR LOGICALLY STARTS UP THE FI (FI-TIPS LINK).

To answer this question, FANSYS retrieves the instantiations of the failed component and the failure-correction procedures used in flight 93-044, namely the FI-TIPS link and the rebooting procedures for the TIPS and the FI-TIPS link. Retrieving these instantiated structures requires: (1) indices and memory links from flight numbers to their associated failure cases; and (2) retrieval functions that take flight numbers as input and retrieve failed-components and failure-correction procedures from failure cases. Once the answer is found, it is converted from memory representation into adequate English by using the lexical patterns associated with each class of knowledge structure. That is, text generation in FANSYS involves using the same lexical patterns that are accessed during the text comprehension.

3. Representing Knowledge of Telescope Systems

As originally developed by Alvarado et al. (1993), the model of domain knowledge used in FANSYS encodes concepts in terms of hierarchical data structures called *Memory Organization Packets (MOPs)* (Schank, 1982; Riesbeck and Schank, 1989). Each MOP can be viewed as a *frame* (Charniak, 1977; Minsky, 1975) that consists of five major components: (1) slots, which specify the attributes and/or components of the concept encoded by the MOP; (2) fillers, which correspond to other concepts that describe values associated to the slots; (3) slot-filler links, which connect the slots of the MOP to their fillers; (4) ISA links,[1] which connect the MOP to its instances (I-MOPs) or to its specializations (M-MOPs); and (5) lexical links, which connect the MOP to associated lexical patterns used during text comprehension and text generation. This MOP-based model has been adapted to represent knowledge about five classes of concepts underlying telescope-failure descriptions, namely: entities, states, events, procedures, and cases.

3.1. Entities

An entity is defined as a component of the KAO. In FANSYS, there are two types of KAO entities: systems and messages. Systems are the hardware, software, and human components required for the operation of the KAO, such as the compressors, the telescope inertia positioning system (TIPS), and the telescope operators. Messages are the communication constructs used among or within system entities, such as network tokens, temperature-readout messages, and pressure-readout messages.

To illustrate how entities are encoded in terms of MOPs, consider figure 1, which depicts a view of a MOP hierarchy including several KAO's components. As the figure shows, M-ENTITY is connected to its two specializations, M-SYSTEM and M-MESSAGE, via arrows that denote ISA links. The MOPs that represent KAO's TIPS and compressors are shown as specializations of M-KAO-UNIT, which is a specialization of the MOP for hardware components.

[1] As explained by Rich and Knight (1991), ISA links are general constructs used in knowledge representation to show that a given concept belongs to a class of concepts, such as in the phrase "a telescope operator ISA crew member."

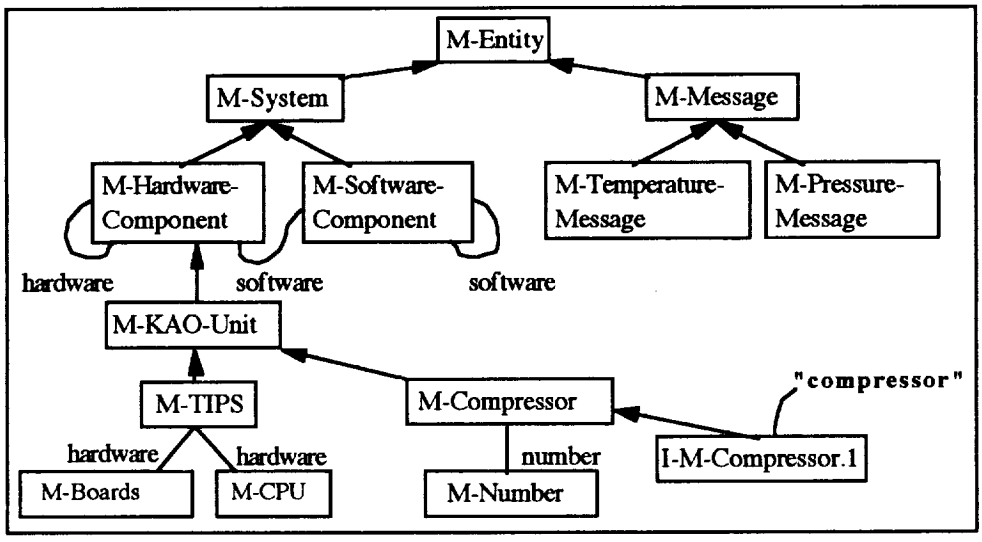

Figure 1. Partial Entity Hierarchy With ISA links (Arrows), Slot-Filler links (Single Lines), and Text Mapping.

Figure 1 also shows that system components may consist of other hardware and software subcomponents, which are connected by slot-filler links depicted as single lines. For instance, M-TIPS is composed of two hardware components, i.e., M-BOARDS and M-CPU. Similarly, M-COMPRESSOR has a number component (i.e., M-NUMBER) that uniquely identifies each compressor. Finally, figure 1 contains I-M-COMPRESSOR.1, an instance of M-COMPRESSOR that is created by FANSYS to represent the input word "compressor."

3.2. States

A state represents a relationship involving one or more KAO components. For example, consider figure 2, which shows instances of a few states. Here, the state of being on refers to the power of an individual component, whereas the state of being physically connected refers to the connection between two hardware components.

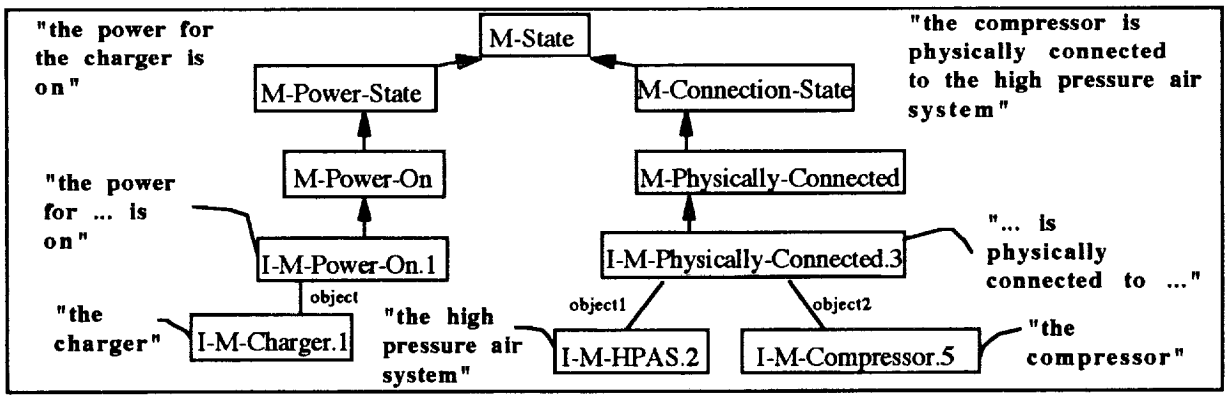

Figure 2. Instances of States.

Figure 2 indicates that states are organized in terms of the ISA and slot-filler links. Each state has one or more slots that are associated with the entities to which the state applies. For example, I-M-POWER-ON.1 has one slot for a given system, and I-M-PHYSICALLY-CONNECTED.3 has two. Figure 2 also shows how these MOPs are used to represent phrases such as "the power for the charger is

on" and "the compressor is physically connected to the high pressure air system." Within the phrases, the devices mentioned map into entity MOPs, while the textual state assertions map into the state MOPS.

3.3. Events

An event represents the occurrence of an action performed on or by an entity, the preconditions for such an occurrence, and the stage change resulting from the action. Each event and its corresponding action involve an actor, and one or more objects that are acted upon. Depending on the action encoded, events fall within one of the following types: (1) powering events, which involve turning on/off entities; (2) diagnostic events, which represent the performance of a diagnostic test; (3) connection events, which encode the establishment of physical or logical connections between different entities; and (4) transmission events, which represent the transmission and reception of messages among entities. For example, figure 3 shows how the sentence "the telescope operator powers up the charger" is represented in terms of an instance of the powering event M-POWER-UP-EVENT.

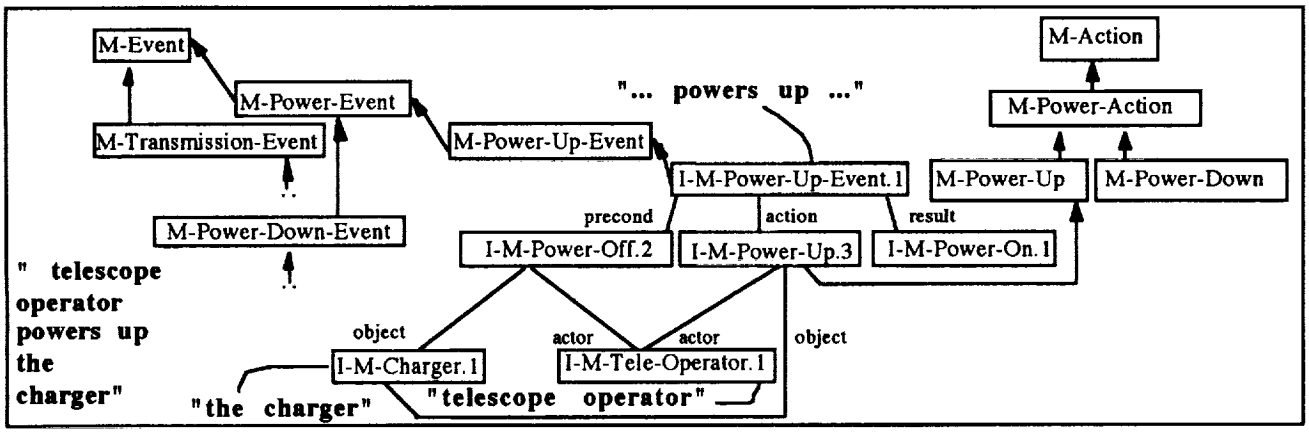

Figure 3. Instance of M-POWER-UP-EVENT.

As figure 3 indicates, I-M-POWER-UP-EVENT.1 is composed of an instance of the action M-POWER-UP, an instance of the state M-POWER-OFF that represents the precondition that the power be initially off, and an instance of the state M-POWER-ON that represents the result that the power be on. Figure 3 also shows that the phrase "powers up" maps directly into I-M-POWER-UP-EVENT.1, which makes explicit the action, precondition, and result associated with this event.

3.4. Procedures

A procedure encodes a causal chain of the steps that: (a) result in a failure mode, its cause, or its effects; or (2) can be used to detect or correct a failure. Each step in the chain contains an event or another procedure. As a result, a procedure can be viewed as a complex series of state transitions that lead from a starting set of precondition states to a final set of result states. For example, M-SOFTWARE-RESET represents a correction procedure in which telescope operators reset a system by logically shutting the failed component off, and then logically starting the failed component back on. The preconditions of this procedure include the fact that the device involved has failed and is powered on, and the result condition is that the device is operational. An instance of M-SOFTWARE-RESET is illustrated in figure 4.

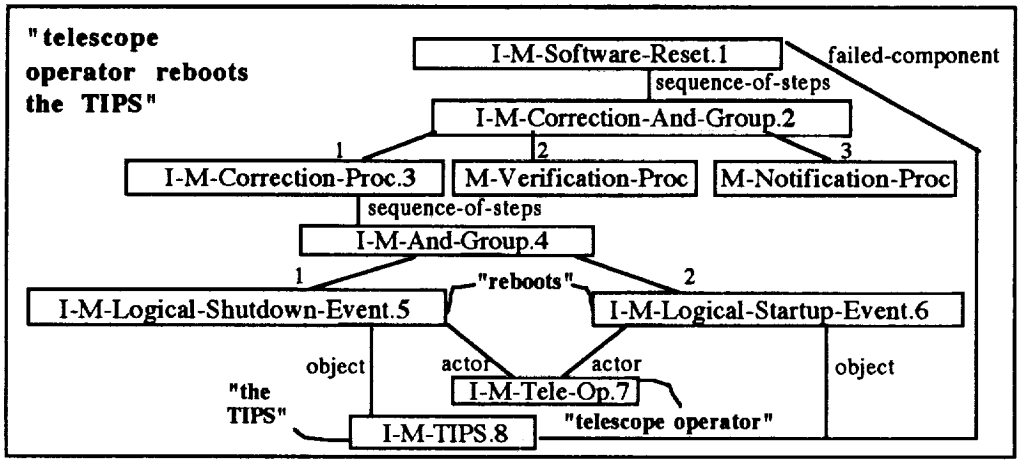

Figure 4. Instance of the procedure M-SOFTWARE-RESET.

Figure 4 shows how I-M-SOFTWARE-RESET.1 represents the sentence "the telescope operator reboots the TIPS." In its sequence of steps, this instance MOP contains a correction procedure that makes explicit the events of logically shutting down and starting up the TIPS. The figure also shows that the representation for the telescope operator and TIPS respectively fill in the object and actor slots of I-M-LOGICAL-SHUTDOWN-EVENT.5 and I-M-LOGICAL-STARTUP-EVENT.6.

3.5. Case Representation

Cases are the top-level organizing structures that package together entities, states, events, and procedures in order to encode failure descriptions. Each case contains slots for the failed component, the failure mode, the failure cause, the failure effect, the failure-detection procedure, the failure-correction procedure, and the flight date and flight number associated with the failure. As an example, consider the text and representation of the failure description COMPRESSOR-FAILURE-1.

COMPRESSOR-FAILURE-1
ITEM NAME: Compressor 3.
FAILURE MODE: Warning Indicator— failure during operation.
FAILURE DETECTION/VERIFICATION: The telescope operator realizes from reading the ADAMS console that the pressure is low in compressor 3.
CORRECTIVE ACTION: The telescope operator switches from compressor 3 to compressor 2.
FLIGHT-DATE: 6-23-93
FLIGHT-NUMBER: 93-048

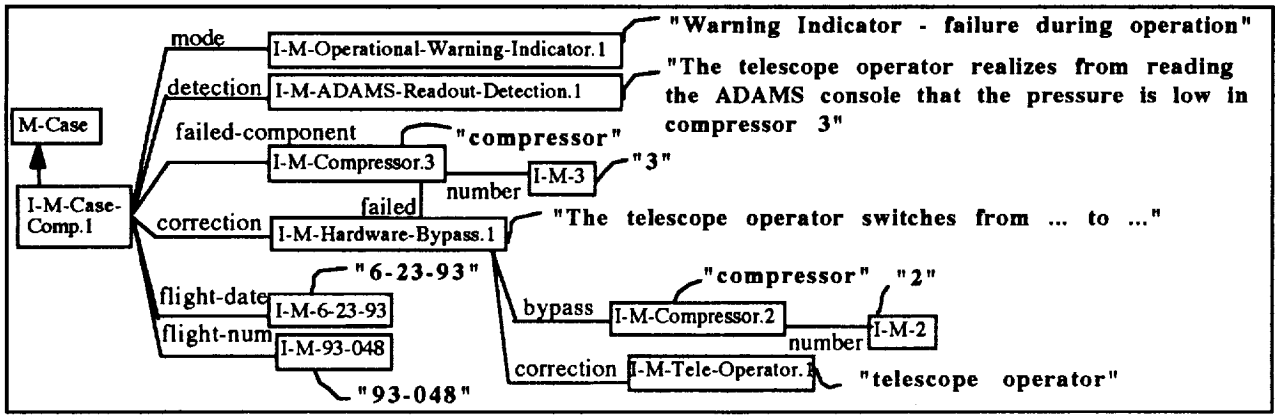

Figure 5. Case Representation of COMPRESSOR-FAILURE-1.

COMPRESSOR-FAILURE-1 indicates that a warning message regarding a compressor failure is displayed to the telescope operator, who subsequently corrects the problem by switching from compressor 3 to compressor 2. This failure description is represented in figure 5 in terms of I-M-CASE-COMP.1, a case instance indexed by the MOP M-CASE. As the figure shows, the slots of I-M-CASE-COMP.1 are filled in with instance MOPs for the given flight date, flight number, failed component, failure mode, detection procedure, and correction procedure. Figure 5 also shows a portion of the slot-filler hierarchy for the correction procedure I-M-HARDWARE-BYPASS, where I-M-COMPRESSOR.3 is the failed component, I-M-COMPRESSOR.2 is the bypass component, and I-M-TELESCOPE-OPERATOR.1 is the correction component.

4. Processing Input Text

As described by Alvarado et al. (1993), comprehension of input text and questions in FANSYS is performed using the case-based parsing techniques provided by DMAP, a Direct Memory Access Parser (Riesbeck and Martin, 1986; Riesbeck and Schank, 1989). In DMAP, parsing is a recognition process, i.e., the goal of the parser is to determine which memory structures best organize the input based upon what the parser has already been exposed to. When the parser reads an input failure description, it searches for the case representations of other descriptions previously processed in order to help it understand the input. The output of the parser is the set of memory structures referenced during the comprehension of the input, the set of new structures added to memory during parsing, and the set of expectations about what may be seen as subsequent input.

Mapping input text into its representation is accomplished through the use of MOPs called *index patterns* (Riesbeck and Martin, 1986; Riesbeck and Schank, 1989). These patterns represent stereotypical mappings of natural language onto their corresponding concepts, as well as processing knowledge that FANSYS applies to aid in constructing case representations. For example, the following pattern:

{ (correction-actor) "switches" "from" (failed) "to" (bypass) } ==> M-HARDWARE-BYPASS.

specifies that the hardware-bypass procedure can be recognized from input text if a concept representing the actor of the procedure is immediately followed by the phrase "switches from," a concept representing the failed object, the word "to," and a concept representing the bypass object. To illustrate how this pattern is used, consider a simplified, annotated trace of the inferences that FANSYS makes when reading the sentence "telescope operator switches from compressor 3 to compressor 2."

Word 1 - "telescope" - activates the pattern:
{ "telescope" "operator" } ==> M-TELESCOPE-OPERATOR.
An expectation for the word "operator" is created.

Word 2 - "operator" - fulfills the expectation from the active M-TELESCOPE-OPERATOR pattern and triggers the creation of the instance I-M-TELESCOPE-OPERATOR.1 for the concept of the telescope operator.

Word 3 - "switches" - activates the pattern:
{ (correction-actor) "switches" "from" (failed) "to" (bypass) } ==> M-HARDWARE-BYPASS.
Here, the actor is constrained to be of the type M-CREW. Given that the active concept I-M-TELESCOPE-OPERATOR.1 is an instance of M-CREW, then this instance is bound to the actor component. Expectations are created for the rest of the words in the pattern, as well as for the failed and bypass components to be of the type M-HARDWARE-COMPONENT.

The pattern { (actor) "switches" (object) "on" } ==> M-POWER-EVENT is also activated, and the actor component is bound to I-M-TELESCOPE-OPERATOR.1. Expectations are created for the rest of the words in the pattern, as well as for the object component to be of the type M-HARDWARE-COMPONENT.

Word 4 - "from" - satisfies the expectation from the M-HARDWARE-BYPASS pattern. However, this word does not match the M-POWER-EVENT pattern and, hence, this index pattern is discarded and the word "switches" is disambiguated as being part of the M-HARDWARE-BYPASS pattern.

Word 5 - "compressor" - activates the pattern { "compressor" (number) } ==> M-COMPRESSOR. An expectation is created for a number.

Word 6 - "3" - satisfies the expectation from the M-COMPRESSOR pattern and triggers the creation of the instance I-M-COMPRESSOR.3 with a filler of "3" for the number slot. This instance satisfies the expectation of the M-HARDWARE-BYPASS pattern activated by the word "switches," and I-M-COMPRESSOR.3 is bound to the failed component of the pattern.

Word 7 - "to" - satisfies the expectation from the M-HARDWARE-BYPASS pattern.

Words 8 and 9 - "compressor" "2" - are processed as words 6 and 7, i.e., the M-COMPRESSOR pattern is activated and its associated expectation is satisfied. As a result, the instance I-M-COMPRESSOR.2 is created with a filler of "2" for the number slot. Because this instance satisfies the final expectation from the M-HARDWARE-BYPASS pattern, I-M-COMPRESSOR.2 is bound to the bypass component of this pattern and the instance I-M-HARDWARE-BYPASS.1 is created.

The final result of this parsing process is the instance I-M-HARDWARE-BYPASS.1 (shown in figure 5) that has the correction component bound to I-M-TELESCOPE-OPERATOR.1, and the failed and bypass components bound to I-M-COMPRESSOR.3 and I-M-COMPRESSOR.2, respectively. In addition, the procedures, events, and states associated with I-M-HARDWARE-BYPASS.1 are also inferred.

The previous example shows the process of parsing a sentence that describes the correction procedure of a case. In addition to the index patterns used in the example, other index patterns are necessary to collect each of the case components described in the input. For example, the pattern:

{ "Corrective" "Action" ":" (correction) } ==> M-CORRECTION-CONTEXT

is used to the recognize the correction procedure as the correction component of the case. Similarly, the pattern:

{ "Failed" "Item" ":" (failed-component) } ==> M-ITEM-CONTEXT

is used to parse the phrase "Failed Item: Compressor 3," and to recognize the compressor as the failed component of the case.

To build the final case representation, FANSYS must know that the input failure description may contain the failed component, the failure mode, the failure cause, the failure effect, the failure-detection procedure, the failure-correction procedure, and the flight date and flight number associated with the failure. To capture these relationships, FANSYS uses the following pattern:

{ (failed-component) (mode) (cause) (effect) (detection) (correction) (flight-number) (date) } ==> M-CASE.

This pattern is a *high level index pattern (HLIP)* (Alvarado, et al., 1993) that indicates that FANSYS should collect the representations of the case components that it finds in the input, and use them as slot fillers for an instance of a case MOP. After all the elements of the case have been collected, the final case representation is constructed, as shown in figure 5.

5. Memory Organization

The MOPs instantiated during text comprehension must be indexed so that they can be accessed during question answering. In FANSYS, memory indexing is accomplished through memory links that organize conceptual relationships among the MOPs that represent entities, states, events, procedures, and cases. As mentioned in Section 3, each MOP in FANSYS may have three types of associated links: (1)

slot-filler links, which connect the slots of the MOP to their fillers; (2) ISA links, which connect the MOP to its instances (I-MOPs) or to its specializations (M-MOPs); and (3) lexical links, which connect the MOP to associated lexical patterns used during text comprehension and text generation. However, these links do not provide a way to organize concepts based on their similarities and differences. As a result, the links are not sufficient to provide access to failure cases during question answering.

In FANSYS, the solution to this problem is to overlay an additional indexing structure on top of the memory hierarchy of ISA links. This additional indexing structure is based upon the memory representation model developed within the CYRUS system (Kolodner, 1984). Kolodner's model allows for the creation of generalizations among similar concepts, as well as the use of such generalizations as indices for the concepts. These indices form a conceptually-rich hierarchy of generalizations where many indices may point to each concept. Within the context of FANSYS, this hierarchy is termed *Generalization-Indexing (GI) memory hierarchy* (Alvarado et al., 1993). Although similar in nature to the indices used by Kolodner in CYRUS, the GI indices employed in FANSYS allow for the attributes of a given concept to be represented hierarchically.

To understand the nature of the memory links used in FANSYS, consider the graph in figure 6, which shows the representation of the event of powering up a KAO component.

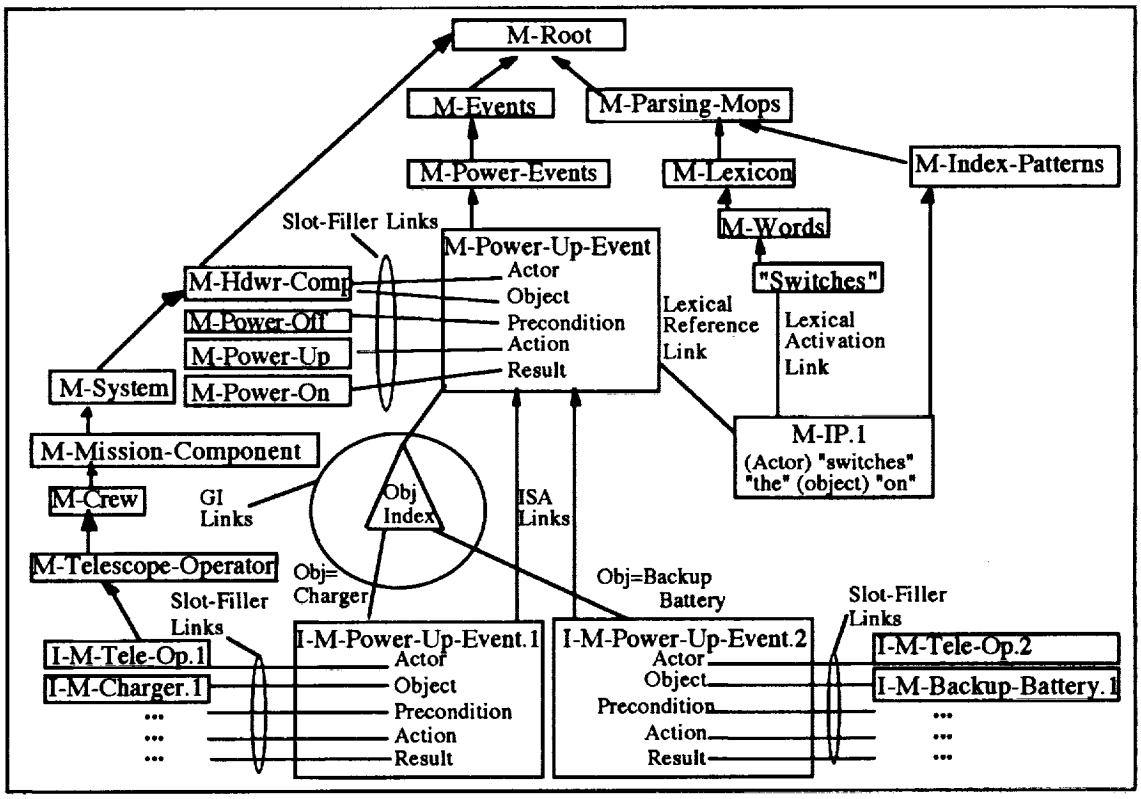

Figure 6. Types of Memory Links.

This figure contains two instances of M-POWER-UP-EVENT. One of the instances involves a telescope operator powering up a charger (I-M-POWER-UP-EVENT.1), and the other instance involves the telescope operator powering up a backup battery (I-M-POWER-UP-EVENT.2). These instances are connected to the abstraction of a general power-up-event (i.e., M-POWER-UP-EVENT) via ISA links depicted by arrowheads. These arrowheads do not indicate indexing or memory access direction, but serve only to differentiate the ISA links from other types of links. The slot-filler links indicate which actors and objects are involved in the instances of M-POWER-UP-EVENT, while the lexical links connect the pattern { (ACTOR) "SWITCHES" (OBJECT) "ON" } to M-POWER-UP-EVENT. Finally, the GI links provide a method to

use M-POWER-UP-EVENT as a generalization for indexing its two instances based upon their differences. In this example, the two instances differ in the object being powered up.

When organizing failure cases in memory, FANSYS builds a memory hierarchy in which each MOP is connected to all its specialization MOPs via GI links that encode the differences among such specializations. Furthermore, each MOP has a set of norms that contain generalized information relating to all MOPs indexed directly below it. As a result of this generalization scheme, the failure cases are located at the bottom of the hierarchy. Thus, the hierarchy organizes MOPs from the general to the specific. This generalization scheme is accomplished in FANSYS through the use of the following algorithm:

Memory-Organization Algorithm

1. Select all components of a new case as features for indexing.
2. Create a new Instance MOP I1 with the components of the new case.
3. Set the current MOP CM to M-CASE, i.e., the top level MOP that indexes cases in FANSYS.
4. Set the norms of CM to the similarities between CM's norms and I1.
5. If none of the indices from CM matches I1, use the components of I1 to create indices pointing from CM to I1. Otherwise, for all indices from CM that match I1, perform one of the following:
   - 5.1. If the index points to an abstraction M1, go to step 4 with M1 as the CM.
   - 5.2. If the index points to an existing Instance MOP I2 then:
     - 5.2.1. Create a new Abstraction MOP M2.
     - 5.2.2. Incorporate the similarities between I2 and I1 as norms of M2.
     - 5.2.3. Establish the differences between I2 and I1 and use these differences to index I1 and I2 from M2.

To illustrate how the algorithm works, consider the simplified memory hierarchies shown in figure 7. Here, two failure cases involving failures of the TIPS are already indexed in memory, and a third case about a compressor failure is being integrated into memory.

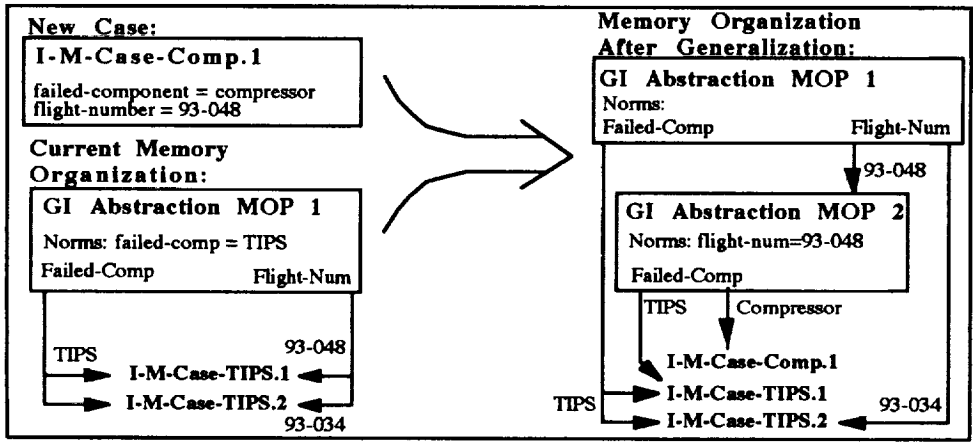

Figure 7. Example of the Application of the Memory-Organization Algorithm.

As this figure shows, I-M-CASE-COMP.1 has the same flight number as I-M-CASE-TIPS.1. When adding I-M-CASE-COMP.1 to the memory hierarchy, a match with I-M-CASE-TIPS.1 occurs along the index "Flight-Num = 93-048." This match results in the creation of the GI-Abstraction MOP2, whose norms contain the similarities between I-M-CASE-COMP.1 and I-M-CASE-TIPS.1, i.e., the flight number 93-048. In addition, the GI-Abstraction MOP2 indexes I-M-CASE-COMP.1 and I-M-CASE-TIPS.1 using their differences, i.e., the failed components. As this example illustrates, the use of the memory-organization algorithm allows FANSYS to make generalizations across failure cases and to cluster similar cases together.

6. Memory Search and Retrieval

To use the hierarchical memory organization during question answering, FANSYS applies search and retrieval strategies based on those developed by Kolodner (1984). FANSYS first parses the input question in order to represent the information given in the question and the information requested in the question in terms of the components of a failure case. Then, FANSYS compares the GI indices in the hierarchy against the question representation, and traverses the indices that match the given information in order to locate failure cases containing the requested information. This traversal process makes use of two major strategies: direct-search method and elaboration method.

The direct-search method involves performing a *depth-first search* (Rich and Knight, 1991) using the indices that match the information given in input questions. That is, FANSYS exhaustively traverses paths in its hierarchical memory that only contain each of the matching indices. As a result, only relevant MOPs within the memory hierarchy are visited. For example, consider the following question answering session:

Question: What is the failure correction procedure for the compressor when the flight number is 93-048?
Given Data: (FAILED-COMPONENT M-COMPRESSOR) (FLIGHT-NUM M-93-048)
Requested Data: (M-FAILURE-CORRECTION)
Results of Direct Search: (I-M-CASE-COMP.1)
Answer: THE TELESCOPE OPERATOR SWITCHES FROM COMPRESSOR 3 TO COMPRESSOR 2.

In this question, FANSYS is asked for the failure correction procedure used in a given flight number to fix a given component. Using the final memory organization in figure 7, FANSYS traverses the index "Flight-Num = 93-048" to reach the GI-Abstraction MOP2, and then traverses the index "FAILED-COMPONENT = Compressor" to reach the failure case I-M-CASE-COMP.1. As this example shows, the process of traversing paths composed of matching indices allows cases to be content-addressable and prevents the scanning of all possible indices.

The direct-search method only works when FANSYS is given enough information to traverse indices all the way down to the failure cases. However, FANSYS should also be capable of handling questions that do not provide enough information to traverse the memory hierarchy and reach specific cases. To solve this problem, FANSYS uses *elaboration techniques* (Kolodner, 1984) to select new indices by generating new pairs of attributes and values based on the given information and the requested information in input questions.

One elaboration method implemented in FANSYS is feature-inference elaboration, which involves relaxing the search constraints to allow the traversal of all indices that have the same attributes as the requested information in input questions. Since the indices are related to the input questions, the search is still restricted to relevant portions of memory. By allowing more indices to be traversed, there is a greater chance of retrieving cases. An example of a question answered by using the feature-inference elaboration is shown below:

Question: What is the failed component when the flight number is 93-048?
Given Data: (FLIGHT-NUM M-93-048)
Requested Data: (M-HARDWARE-COMPONENT)
Results of Direct Search: None found.
Elaborating using requested information.
Results of elaboration: (I-M-CASE-TIPS.1) (I-M-CASE-COMP.1)
Answers: [1] FI (TIPS - FI LINK) [2] COMPRESSOR (THREE)

The above question is similar to the one in the direct-search example, except that the flight number alone is not enough information to retrieve the failure cases. Using the final memory hierarchy in figure 7 and the direct-search method, traversal proceeds from the GI-Abstraction MOP1 to the GI-Abstraction MOP2 along the index "FLIGHT-NUM = 93-048." However, at this point there is no more information provided by the question and FANSYS cannot proceed any further. To retrieve the relevant cases,

FANSYS relaxes the search constraints by allowing traversal of any indices that have an attribute of FAILED-COMPONENT. As a result of the new search, the Compressor and TIPS cases can be retrieved by following the FAILED-COMPONENT indices from the GI-Abstraction MOP2.[2]

7. Current Status and Future Work

FANSYS has been developed at the UC Davis Artificial Intelligence Laboratory on Sun workstations using two public domain software packages: CMU Common Lisp (MacLachlan, 1992) and the GARNET User Interface Management System (Myers et al., 1992). Currently, FANSYS contains a total of fifteen cases representing failures of the KAO components listed in Section 1. These cases are built by FANSYS from input representations of case components, which are then used to create all of the indices and generalizations necessary to perform question answering. Furthermore, FANSYS contains enough lexical knowledge to parse the failure description UPS-FAILURE-1 (see Section 2), and more lexical knowledge is being implemented in FANSYS to allow it to parse other failure descriptions for the failure cases already encoded in memory. In addition, as described by Alvarado et al. (1993), the text comprehension and question answering strategies of FANSYS have previously been tested using failure descriptions involving the data management system (DMS) of NASA's Space Station.

The experience of developing an experimental failure-analysis model in FANSYS has helped elucidate representational constructs and processing strategies required for computer comprehension and retrieval of failure cases. Although FANSYS has provided an initial testbed for this model, there are limitations to FANSYS that indicate directions for future research, including the following:

(1) Acquisition of Domain Knowledge: FANSYS must be able to handle input text that is not directly dependent on the knowledge constructs and lexical entries initially encoded in the system. To address this knowledge-engineering issue, FANSYS must be able to dynamically augment its knowledge during text comprehension. Consequently, the model of text comprehension and question answering implemented in FANSYS must include learning strategies (Kolodner, 1993; Pazzani, 1990; Riesbeck and Schank, 1989) for acquiring and encoding knowledge constructs associated with a given domain or a given natural language.

(2) Subjective Comprehension and Verification of Input Text: Comprehension of input text and questions in FANSYS must also be influenced by the *ideological perspective* (Carbonell, 1981) that FANSYS may have about diagnosis and repair procedures in a given domain. That is, FANSYS must attempt to understand failure descriptions and relate their conceptual content to its own beliefs and justifications involving related and/or similar failures. If FANSYS reads descriptions that are inconsistent with its beliefs, then FANSYS must be able to generate counterarguments as it reads the input text or questions. In order to account for this process of subjective comprehension, FANSYS must have its own ideology, strategies for determining inconsistencies between its beliefs and input text, and strategies for selecting counterarguments. These counterargument strategies should be based on the argument-planning knowledge underlying the taxonomy of argument units proposed by Alvarado (1990).

(3) Belief Inferences from Failure Cases: FANSYS must be able to answer questions that require inferring beliefs from the cases already existing in memory. For example, consider a hypothetical question of the type: "Would FANSYS agree/disagree with advise seeker A over the use of repair procedure P when dealing with failure F?" Here, FANSYS must be able to use memories from previous failure situations to aid in understanding how a repair procedure might be used in a new failure situation. Getting FANSYS to handle hypothetical questions will require modeling the process of *belief inference* (Alvarado, 1990) in relation to *models of ideology* (Carbonell, 1981).

---

[2]If the feature-inference elaboration fails, FANSYS may use another strategy termed *alternate-context elaboration* (Kolodner, 1984), which allows FANSYS to use domain knowledge for inferring a new context for the search.

8. Conclusions

This paper has described techniques for comprehension, organization, and retrieval that are used in FANSYS to deal with failure descriptions and failure-related questions in airborne observatories. A major benefit derived from this work has been the formulation of a computational framework for: (1) integrating domain-dependent knowledge with case-based parsing strategies within text comprehension systems; (2) representing and indexing failure cases; (3) creating generalizations among these cases; and (4) using search and retrieval strategies to answer questions about failure causes and effects, as well as failure detection and correction procedures. The experience of designing and implementing this model of text comprehension and question answering has shed light on some of the basic problems any intelligent computer system must address: knowledge representation, organization, application, and retrieval. The model has helped increase understanding of representational constructs and processing strategies needed to analyze system failures. As such, FANSYS provides a computational framework for developing advice-giving and decision-making systems that may help operate and maintain NASA's aircraft and spacecraft.

References

Acorn, T. and Walden, S. (1992). SMART: Support Management Cultivated Reasoning Technology for Compaq Customer Service. *Proceedings of IAAI-92.* Cambridge, MA: AAAI Press/MIT Press.

Alvarado, S. J. (1990). *Understanding Editorial Text: A Computer Model of Argument Comprehension.* Boston, MA: Kluwer Academic Publishers.

Alvarado, S. J., Braun, R. K., and Mock, K. J. (1993). *FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis* (Tech. Report CSE-93-4). Computer Science Department, University of California, Davis.

Carbonell, J. G. (1981). *Subjective Understanding: Computer Models of Belief Systems.* Ann Arbor, MI: UMI Research Press.

Charniak, E. (1977). A framed PAINTING: The Representation of a Commonsense Knowledge Fragment, *Cognitive Science*, **1**, 355-394.

de Kleer, J. and Williams, B. (1987). Diagnosing Multiple Faults, *Artificial Intelligence.*, **32**, 97-130.

Dyer, M. G. and Lehnert, W. G. (1982). Question Answering for Narrative Memory. In J. F. Le Ny and W. Kintsch (Eds.), *Language and Comprehension.* Amsterdam: North-Holland.

Erickson, E. F., Davidson, J. A., Thorley, G., and Caroff, L. J. (1991). *SOFIA: A Stratospheric Observatory for Infrared Astronomy* (NASA Tech. Memorandum 103840). NASA-Ames Research Center, Moffett Field, CA.

Hammond, K., and Hurwitz, N. (1988). Extracting Diagnostic Features from Explanations, *Proceedings of the Workshop on Case-Based Reasoning (DARPA).* San Mateo, CA: Morgan-Kaufmann.

Iverson, D. L. and Patterson-Hine, F. A. (1990). Object-Oriented Fault Tree Models Applied to System Diagnosis. *Proceedings of the SPIE Applications of Artificial Intelligence VIII Conference.* Orlando, FL.

Kolodner, J. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model.* Hillsdale, NJ: Lawrence Earlbaum.

Kolodner, J. (1993). *Case-Based Reasoning.* San Mateo, CA: Morgan-Kaufmann.

Lehnert, W. G. (1978). *The Process of Question Answering: A Computer Simulation of Cognition.* Hillsdale, NJ: Lawrence Earlbaum.

Liu, H., Tan, A., Lim, J., and Teh, H. (1991). Practical Application of a Connectionist Expert System — The INSIDE Story, *World Congress on Expert Systems*, Orlando, FL: Pergamon Press.

MacLachlan, R. A. (Ed.) (1992). *CMU Common Lisp User's Manual* (Tech. Report CMU-CS-92-161). School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Minsky, M. (1975). Framework for Representing Knowledge, In P. Winston (Ed.), *Psychology of Computer Vision.* New York: McGraw-Hill.

Myers, B. A., Giuse, D., Dannenberg, R. B., Zanden, B. V., Kosbie, D., Marchal, P., Pervin, E., Mickish, A., Landay, J. A., MacDaniel, R., and Hopkins, D. (1992). *The Garnet Reference Manual, Revised Version 2.1* (Tech. Report CMU-CS-90-117-R3). School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

NSI (1993). *UPS System Design Report #21-1117.* NSI, NASA-Ames Research Center, Moffett Field, CA.

Patterson-Hine, F. and Iverson, D. (1990). *An Integrated Approach to System Design, Reliability, and Diagnosis,* NASA Technical Memorandum 102861. Ames Research Center, Moffett Field, CA.

Pazzani, M. J. (1990). *Creating a Memory of Causal Relationships: An Integration of Empirical and Explanation-Based Learning Methods.* Hillsdale, NJ: Lawrence Erlbaum.

Redmond, M. A. (1990). Distributed Cases for Case-Based Reasoning: Facilitating Use of Multiple Cases. *Proceedings of AAAI-90.* Cambridge, MA: AAAI Press/MIT Press.

Reed, N. and Johnson, P. (1990). Generative Knowledge for Computer Troubleshooting, *European Conference on Artificial Intelligence '90.* London: Pitman.

Rich, E. and Knight, K. (1991). *Artificial Intelligence* (Second Edition). NY: McGraw-Hill.

Riesbeck, C. K. and Martin, C. E. (1986). Direct Memory Access Parsing. In. Kolodner, J. and Riesbeck, R. (Eds.), *Experience, Memory, and Reasoning.* Hillsdale, NJ: Lawrence Erlbaum.

Riesbeck, C. K. and Schank, R. (1989). *Inside Case-based Reasoning.* Hillsdale, NJ: Lawrence Erlbaum.

Schank, R. C. (1982). *Dynamic Memory: A Theory of Reminding and Learning in Computers and People.* Cambridge: Cambridge University Press.

Simoudis, E., Miller, J. (1990). Validated Retrieval in Case-Based Reasoning. *Proceedings of AAAI-90.* Cambridge, MA: AAAI Press/MIT Press.

Stevenson, R., Miller, J., and Austin, M. (1991). Failure Environment Analysis Tool (FEAT) Development Status, *Proceedings of AIAA 1991.*

Struss, P. (1988). Extensions to ATMS-based Diagnosis, In Gero (Ed.), *Artificial Intelligence Engineering: Diagnosis and Learning.* Amsterdam: Elsevier.

Thronson, H. A. Jr. and Erickson, E. F. (Ed.) (1984). *Airborne Astronomy Symposium: A Symposium Commemorating the Tenth Anniversary of Operations of the Kuiper Airborne Observatory* (NASA Conference Publication 2353). NASA-Ames Research Center, Moffett Field, CA.