



Automated Text Highlighting of Navy Equipment Failure Messages

K. WAUCHOPE AND E. MARSH

*Navy Center for Applied Research in Artificial Intelligence
Information Technology Division*

AND

M. K. DIBENIGNO

JAYCOR

November 2, 1988

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION / AVAILABILITY OF REPORT			
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE		Approved for public release; distribution unlimited.			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Report 9154			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Naval Research Laboratory		6b. OFFICE SYMBOL (If applicable) Code 5510	7a. NAME OF MONITORING ORGANIZATION Naval Research Laboratory		
6c. ADDRESS (City, State, and ZIP Code) Washington, DC 20375-5000			7b. ADDRESS (City, State, and ZIP Code) Washington, DC 20375-5000		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Office of Naval Research		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO. 61153N	PROJECT NO. 55015-08-01	WORK UNIT ACCESSION NO. DN155-294
11. TITLE (Include Security Classification) Automated Text Highlighting of Navy Equipment Failure Messages					
12. PERSONAL AUTHOR(S) Wauchope, K., DiBenigno, M.K.,* and Marsh, E.					
13a. TYPE OF REPORT		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) 1988 November 2	15. PAGE COUNT 46
16. SUPPLEMENTARY NOTATION *JAYCOR, Vienna, VA					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Natural language understanding Navy Messages		
			Knowledge based systems		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
<p>This report describes Text Reduction System (TERSE) that is a knowledge-based system for highlighting important information in the narrative portion of Navy equipment failure messages—Casualty Reports (CASREPs). The system contains two knowledge bases for message evaluation, one that is equipment-specific and the other equipment-general. The equipment-specific knowledge base contains a structural model of a piece of equipment discussed in one class of CASREPs (a shipboard air compressor), encoded as a network of slot/filler units. Since message writers use a wide variety of descriptive naming conventions in referring to pieces of equipment, it is not possible to provide a complete list of synonyms for each part. Instead, the system must use the equipment model to actively dereference each complex nominal, by finding an equipment unit whose attributes match a structural host-modifier analysis of the noun phrase. When an equipment name is underspecified, a disambiguation algorithm uses the equipment model to select the most likely referent from the ambiguity set of matching units. The system also contains general causal inferencing heuristics that use the equipment model network to infer causal relationships that are believed to be implicit in the message. The</p> <p style="text-align: right;">(Continues)</p>					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Kenneth Wauchope			22b. TELEPHONE (Include Area Code) (202)767-9004	22c. OFFICE SYMBOL Code 5510	

19. ABSTRACT (Continued)

equipment-general portion of the system performs semantic normalization, infers and tags key categories of information, and finally ranks the message clauses by applying user evaluation criteria represented as numeric scores assigned to various patterns of information types. The system is implemented in the KEE expert system shell and runs on a Symbolics machine and Sun workstation.

CONTENTS

INTRODUCTION	1
TEXT ANALYSIS AND REPRESENTATION	2
SYSTEM ARCHITECTURE	5
THE TERSE-RULES RULE BASE	8
THE EQUIPMENT MODEL	11
Nominal Dereferencing	13
Matching	15
Contextual Disambiguation	18
The MODEL Rule Base	20
Model-Based Inferencing	22
SAMPLE RUN	24
IMPLEMENTATION	33
CONCLUDING REMARKS	34
Nominal Dereferencing Limitations	34
Head Nouns	35
Left Modifiers	35
Text Representation Limitations	36
RECOMMENDATIONS	38
Transitioning to Fleet	38
Moving to New Message Domains	38
Reevaluating the Task	39
ACKNOWLEDGMENTS	40
REFERENCES	40

AUTOMATED TEXT HIGHLIGHTING OF NAVY EQUIPMENT FAILURE MESSAGES

INTRODUCTION

In this report we describe a prototype system used to highlight important information in narrative texts, specifically in messages reporting failures of shipboard equipment, Casualty Reports (CASREPs). This system, Text Reduction System (TERSE), is implemented in the KEE* expert system shell. The input to the system is a semantic representation of the message content that derives from a separate linguistic component. The system uses two knowledge sources to extract from the message content the clause(s) most relevant to some end-user application, such as equipment failure trend analysis. The knowledge sources are an equipment model, representing the equipment-specific knowledge of the system, and a set of more general rules, most of which are derived from our pilot SUMMARY system [1,2].

We wrote the SUMMARY system as a production rule system in the OPS5 programming language [3]. The goal of the system was to extract the same information from CASREPs as did a team of NAVSEA contractors who had the task of generating a short textual extract of each message to be used in failure trend analysis. Since the manual extracts rarely contained text that was not present in the original narrative, but usually restated a clause that was already in the message, we took the same approach. We modeled text-extraction knowledge in our system as a saliency rating of certain types of information deemed important to include in the extract, such as causality, results of investigation, and malfunction. The system also contained rules to infer the presence of certain classes of information when not explicitly stated in the message. Numeric scores associated with each type of information represented saliency ratings. The presence of any of these types of information in a message clause would add to the clause's overall score, and finally the highest scoring clause(s) would be chosen as the message extract.

Although SUMMARY's output compared favorably with the manual extracts, it lacked domain-specific expertise. For example, in the message illustrated in Fig. 1, the SUMMARY system chose *Splines were extensively worn* as the extract, in part because that assertion mentions the word *worn* that the system could recognize unambiguously as denoting a bad state. The manual extract contained this assertion, but also included the assertion *Drive shaft was found to rotate freely at the SSDG end*. SUMMARY could not recognize "rotate freely" as connoting an abnormality, therefore it did not endow that assertion with as high a ranking as the other. Indeed, human subjects presented with this sentence out of message context could not agree whether it connoted a positive or negative condition, because the word *freely* could connote either one. Apparently the writers of the manual extract were making use of deeper sorts of knowledge, such as domain-specific expertise and discourse tracking, to infer the negative status of the shaft from the message context.

Manuscript approved July 13, 1988.

*KEE is a product of Intellicorp, Mountain View, California.

<p><i>SAC received high usage during two BECCCE periods.</i> <i>CCS received a report that LO pressure was dropping.</i> <i>Alarm sounded.</i> <i>Loud noises were coming from the drive end during coast down.</i> <i>Drive shaft was found to rotate freely at the SSDG end.</i> <i>Splines were extensively worn.</i></p>

Fig. 1 — Message 1

Although SUMMARY's performance seemed reasonably good, we wanted to investigate whether providing additional, deeper knowledge would create a worthwhile improvement in performance. This report describes the current development version of the system, which contains much more domain-specific knowledge about a particular equipment world (a shipboard air compressor) than did SUMMARY, allowing it to form a deeper understanding of the air compressor equipment failure messages than before. Otherwise the basic concept behind the original rule system has changed little.

The updated TERSE system is implemented in the KEE expert system shell. OPS5 is fundamentally a production rule system with only primitive means for representing structured objects. KEE provided a true frame-based object representation scheme that was ideal for building a highly structured equipment model. In KEE the object-oriented representation is fully integrated with the production rule facility and with an English-like assertion/query language, which allows us to keep the rule-based approach used in SUMMARY while making the rules more readable. Finally, KEE provided a powerful graphics interface that greatly improved product development and demonstration.

The organization of this report is as follows. First we briefly describe natural language procedures and the resulting text representation that is used as input to the TERSE system. These are more fully described in Ref. 2. We then provide an overview of the TERSE system architecture and a discussion of its control routine and database element definitions. Next we describe the knowledge bases used in the TERSE system—the TERSE-RULES knowledge base and the equipment model and its associated rules. This is followed by a full example of system execution of a sample message. Then we evaluate the current TERSE system implementation, and finally we conclude the report with discussions of what will be necessary for the transition of TERSE for use in the Fleet and for future research issues.

TEXT ANALYSIS AND REPRESENTATION

The primary task of the natural language processing procedures used for text analysis is to derive a canonical representation of the information found in the narrative portions of a message. In the earlier version of this system and in TERSE, we have used an approach called *information formatting*. This approach was developed and implemented by Sager et al. at New York University in the Linguistic String Project (LSP) system for natural language analysis [4]. Originally developed for patient records and for journal articles within a medical domain, the system performs natural language analysis on a text and derives information formats; we have adapted this system to the Navy equipment failure domain [5]. The natural language analysis procedure involves four stages of processing: parsing, syntactic decomposition, regularization, and mapping into the information format structure (Fig. 2).

We can think of an information format as being a record structure, with one slot for each type of information that can occur in a class of texts. Because texts in a restricted domain discuss a limited set of objects, descriptions, and actions, semantic classes for these basic types of information can

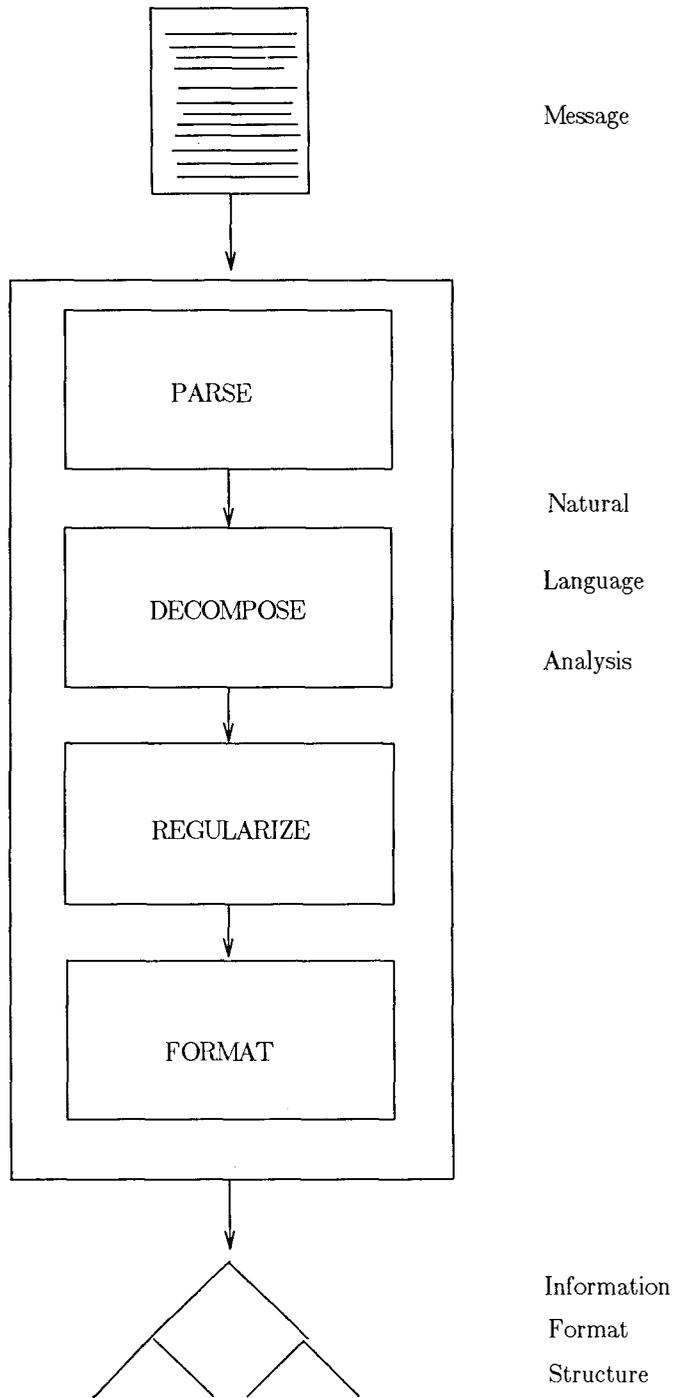


Fig. 2 — Natural language analysis procedures

be derived through techniques of distributional analysis [6]. Distributional analysis generates these classes by looking at co-occurrence properties of words in their syntactic context. We assume that words with similar co-occurrence patterns have similar informational standing in the domain, therefore they are placed in the same class. For example, objects in CASREPs about starting air compressors include, among others, names of pieces of equipment and their component parts, and organizations that operate and maintain the equipment. Descriptions of objects include function, status, and tasks. Actions on objects include such things as repair actions, diagnosis actions, and reporting actions. Figure 3 illustrates several of the semantic categories that were derived by distributional analysis of the CASREP message domain, along with the mnemonic name we have given each category.

ADMIN	action or request for part: <i>forward, report, expedite</i>
FUNC	function performed by equipment: <i>broadcast, communication, operate</i>
INVEST	investigative act: <i>check, isolate, troubleshooting</i>
ORG	personnel or organization: <i>MOTU, ship, technical, originator, technician</i>
PART	equipment, subsystem, or part: <i>antenna, AN/URC-9, controller</i>
PROCURE	action to request, ship, receive, or hold part: <i>deliver, purchase, reorder</i>
PROP	property of part: <i>allowance, clearance, sync, weight</i>
REPAIR	repair action: <i>repair, adjust, overhaul</i>
STASK	ship's task: <i>arrival, assignment, transit</i>
STATUS	equipment status: <i>casualty, fault, malfunction, good</i>

Fig. 3 — Some CASREP semantic categories

Semantic class generally holds constant across syntactic categories; words that are morphologically related by either derivational or inflectional suffixes usually have similar distributions and are therefore categorized similarly. For example, *investigate* and *investigation* are both INVEST words, and both can occur in similar environments: *MOTU 12 investigated problem with SAC* and *MOTU 12's investigation of problem with SAC*. Usually a semantic class defines a word uniquely; thus *SAC* (starting air compressor) is always a PART word within this domain. However, this may not always be the case, since the meanings of some words change with the context. For example, we see in Fig. 4 that the word *receive* occurs in several different environments, but each occurrence has a different

1.	Second generation SAC <i>received</i> onboard for installation.	PROCURE
2.	Antenna was one of 2 primary HF <i>receive</i> antennas.	PROCESS
3.	<i>Received</i> low lube oil pressure alarm on number 2 SAC.	EVID

Fig. 4 — Word meaning in different contexts

meaning. In the first sentence, *receive* indicates a procurement action; in the second sentence, an electrical process; and in the third, the receipt of information by personnel. Each of these meanings is represented by a different semantic class.

Each information format represents a simple assertion or clause in the message. Each word or phrase in the assertion is entered into the appropriate semantic category slot of the format. Also, formats contain slots for tense information and time expressions. Each slot also has subslots to hold modifier information, such as negation and modality.

A second type of record, called a *connective*, connects other formats together. It has one slot representing the connective category and two additional slots to hold the formats being connected. Connectives include both explicit syntactic connectives, such as conjunction markers (i.e. *and*, *or*, and *,"*), and implicit markers, such as different types of complement (infinitival or assertional) and relative clause markers. They also include causal and time-relations that may hold among assertions.

Figure 5 illustrates the information format structure (in list notation, as the language analysis component generates) of the sentence *Drive shaft was found to rotate freely at the SSDG end*, from the message in Fig. 1. The process used to generate this format from raw text is as follows. First, the parsing procedure determines the sentence structure, identifies phrase and clause boundaries and host-modifier relationships, and identifies the scope of conjunctions. In our example sentence, *drive shaft* is identified as the surface SUBJECT noun phrase of the sentence, *was found* as the VERB, *to rotate freely* as the OBJECT, and *freely* is also identified as modifying *rotate*. During syntactic decomposition, variety in sentence structure is reduced by syntactic transformation to simplify mapping into the information formats. Various types of clause, e.g., passives, restrictive relative clauses, infinitival clauses, and sentence fragments, are transformed into simple active assertions, and missing information in reduced clauses is recovered if it is available within the sentence. Also, morphological variants of a word are reduced to a single base form, and most conjoined structures are expanded into conjunctions of complete assertions. Thus, the parse tree for our example sentence is transformed into a tree with the terminal elements *SUBJ find PAST [that] drive shaft rotate freely at SSDG end*, where SUBJ is an empty node since we cannot, at this point, determine who the deep subject of the sentence is, and the OBJECT of the sentence becomes the assertion *drive shaft rotate freely at the SSDG end*. The regularization procedure strips off connectives and places them in front of their arguments; here *find* is identified as a connective of the RELATION type. Its first argument is empty since the deep SUBJECT of the sentence has not been identified, and the second represents the assertion *drive shaft rotate freely at the SSDG end*.^{*} Finally, the arguments of the connectives are mapped into separate formats, and their words are mapped into appropriate slots of the information format structure. In this example, *drive shaft* maps into the PART slot, *rotate* into the FUNC slot, and *freely* into the STATUS slot of the format.

SYSTEM ARCHITECTURE

After the natural language analysis phase has converted a message into information format structures, these structures are passed to the TERSE system. TERSE is composed of four KEE

^{*}Treating *find* as a connective, however, means that the zeroed subject is represented as a format structure. Since each format represents a sentence or clause, this means that TERSE must incorrectly interpret the subject as an empty *assertion*. This inconsistency in the representational scheme is discussed further in the section on "Text Representation Limitations."

```

(CONN (OP (RELATION (HEAD FIND) (TEXT FIND)))
  (ARG1 (FMT))
  (ARG2
    (FMT (TEXT DRIVE SHAFT ROTATE FREELY AT THE SSDG END)
      (PART (HEAD SHAFT) (TEXT DRIVE SHAFT))
      (STATUS (HEAD FREELY)(TEXT FREELY))
      (FUNC (HEAD ROTATE) (TEXT ROTATE))))))

```

Fig. 5 — Information format for
Drive shaft was found to rotate freely at the SSDG end

“knowledge bases,” or code modules: TERSE-RULES, MODEL, SYSTEM, and ICONS. The overall system architecture is shown in Fig. 6(a). TERSE-RULES and MODEL are true knowledge bases that contain the rules and equipment description used by the system in evaluating messages. TERSE-RULES is the original SUMMARY system rulebase with some modifications and augmentations. It is discussed in more detail in the next section of this report. It represents the more domain-independent portion of the system’s knowledge. MODEL incorporates specific knowledge about the particular equipment domain in question. It contains the frame-based representation of the starting air compressor equipment, code for equipment nominal dereferencing, and an additional set of production rules for doing model-based inferencing. The section of this report entitled “THE EQUIPMENT MODEL” discusses the MODEL knowledge base in more detail. SYSTEM contains the code and unit definitions used to translate the input to the system (information formats) into internal database elements, and also contains the control information for running the system. Once installed in the SYSTEM module, the database elements representing the input message can be accessed freely by rules and LISP methods contained in the same or other knowledge bases. ICONS (not shown) contains the definitions of the windows and icons that comprise the user interface to the system.

The system data/control flow is shown in Fig. 6(b). The rules in TERSE-RULES and MODEL are organized into hierarchical rule sets. Both knowledge base and rule set activation are sequenced deterministically by the external control routine contained in the SYSTEM module. After the SYSTEM knowledge base has converted the information format input data into database elements, it passes processing to the MODEL knowledge base. The nominal dereferencing subcomponent of MODEL begins by identifying all references to pieces of equipment in the message. Processing then proceeds to the TERSE-RULES knowledge base where equipment-general word classification takes place. Control then returns to MODEL to perform domain-based causal and state inferencing. Finally, processing is passed back to TERSE-RULES to sequence through the remaining rule sets.

Rules within each rule class are allowed to fire opportunistically. Since we want to extract from the message all instances of the types of information that are used in evaluating a message, we use a forward-chaining (exhaustive) control strategy for rule firing, as opposed to a backward-chaining (sufficiency or goal-directed) strategy. Although the system was intended to be capable of deriving text highlightings tailored for different user goals, a goal-directed approach is not being used. This is because the fundamental classes of information that are conveyed by CASREPs remain constant, and we would accomplish user-tailoring by applying new sets of evaluation criteria to the same set of basic information classes.

Information formats are represented internally as sets of units (frames). The largest unit of data is the unit that represents an entire message. The system processes one message at a time. A message unit contains a slot that is filled by a list of units called **metaformats**. A **metaformat** unit can either be a **conn** unit or a **format** unit.

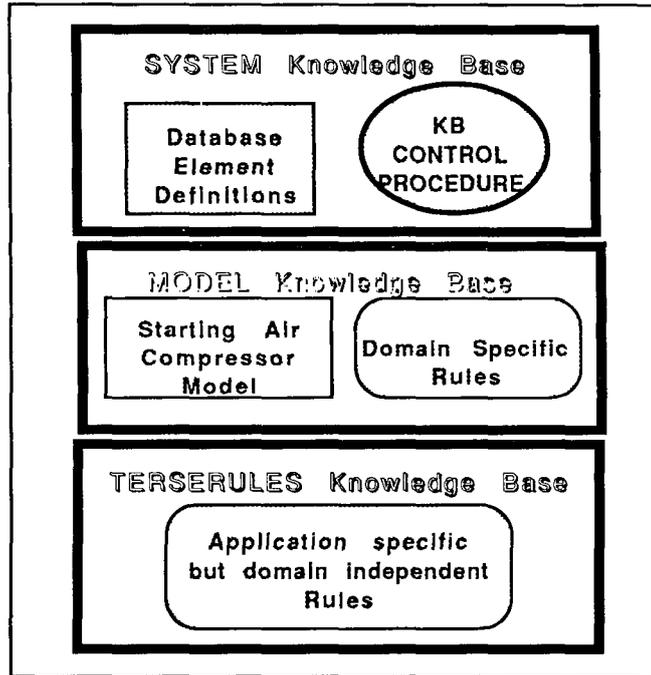


Fig. 6(a) — TERSE architecture

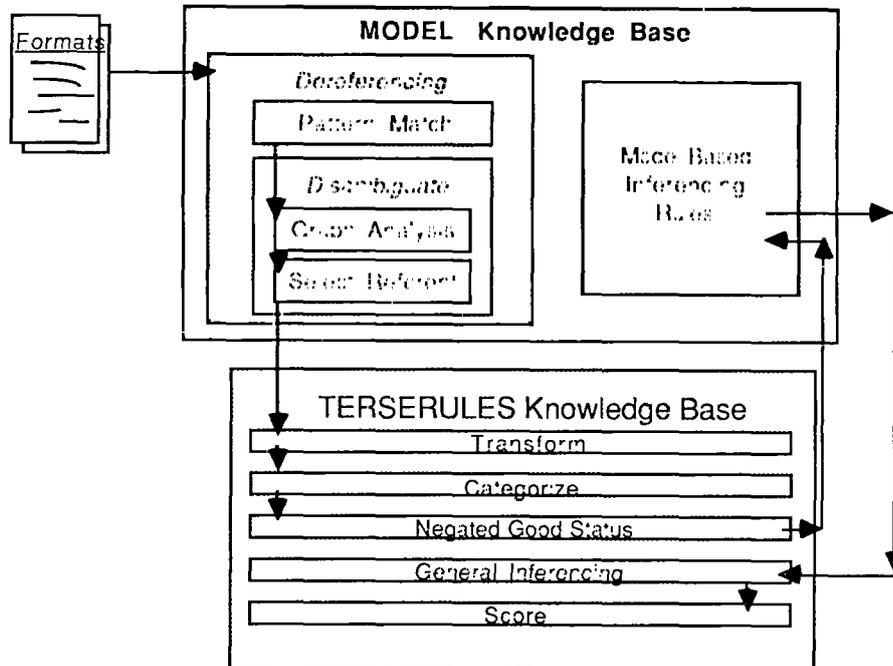


Fig. 6(b) — TERSE data flow

The **conn** unit has three slots that contain pointers to other units. These three slots are *op*, *arg1*, and *arg2*. The *op* slot contains a pointer to a unit that contains the connective word. The *arg1* and *arg2* contain pointers to other **metaformat** units that are the arguments to the connective.

The **format** unit contains a slot for each semantic category in the sublanguage. When a format contains an entry under a particular semantic category, the corresponding slot in the format unit is filled with a pointer to another unit representing the entry itself.

Each semantic category has a unit representing it. These units have slots to hold the head word of the entry, the original text, and slots for all possible modifiers.

Figure 7 illustrates the database units generated by the SYSTEM knowledge base to represent the information format shown in Fig. 5. This database can now be queried by a request such as

(QUERY '(THE HEAD OF (THE PART OF (THE ARG2 OF CONN1)) IS ?WHAT))

which will return the response

(A HEAD OF (THE PART OF (THE ARG2 OF CONN1)) IS SHAFT)

Several other units are contained in the SYSTEM knowledge base. The units are named **explanation-text-displays**, **format-text-displays**, **bequeath-explanation**, and **keepictures.instances**, and they all relate to the pictures and displays that appear when the system is running. The remaining units are **comments**, **bequeath**, and **reduction.system**. **Comments** holds notes for the designers about changes that were made to the system. **Bequeath** is a special type of unit that contains procedural information that is attached to the **conn** unit. Since a connective cannot be picked as the highlighted information, any saliency rating attributed to a connective must be passed down to its arguments until the score ultimately reaches a **format** unit. This procedure handles that process.

Reduction.system is the central control for the system interface. This unit contains slots that contain the procedures to initiate all system functions. Most of these procedures are attached to mouse-sensitive icons in the system interface command panel.

THE TERSE-RULES RULE BASE

The **Terse.rules** class contains five subclasses: **Transform**, **Categ**, **Negated.good.status**, **General.infer**, and **Score**. Rules that are designed to work towards similar goals are grouped in the same rule class. The following is an overview of each of these rule classes accompanied by a description of the member rules in each class.

The **Transform** class of rules performs operations that accomplish further normalization of causal structures. Its objective is to change statements of the form “x is due to y” and “y impaired x” into the canonical form “y caused x.” Removing this variability while retaining the causal implications of the original statement, simplifies subsequent rules, since they do not have to look for the different variations. The rules in this class are as follows.

- **due.to.cause**: Normalize sentences of the form “A is due-to B” into “B caused A.”
- **impair.to.cause**: Change any statement of impairment, as in “A inhibits B,” to “A causes B to be BAD.” Words indicating impairment are *impair*, *inhibit*, *prevent*, *stop*.

```

Unit <CONN1> =
  Class: CONN
  Op: <OP1>
  Arg1: <FORMAT1>
  Arg2: <FORMAT2>

Unit <OP1> =
  Class: OP
  Op-type: RELATION
  Head: FIND
  Text: FIND

Unit <FORMAT1> =
  Class: FORMAT

Unit <FORMAT2> =
  Class: FORMAT
  Func: <FUNC1>
  Part: <PART1>
  Status: <STATUS1>
  Text: (DRIVE SHAFT ROTATE FREELY AT THE SSDG END)

Unit <FUNC1> =
  Class: FUNC
  Head: ROTATE
  Text: ROTATE

Unit <PART1> =
  Class: PART
  Head: SHAFT
  Text: (DRIVE SHAFT)

Unit <STATUS1> =
  Class: STATUS
  Head: FREELY
  Text: FREELY

```

Fig. 7 — Information format database units for
Drive shaft was found to rotate freely at the SSDG end

The **Categ** rules classify the semantic category entries contained in the formats. These categories represent concepts that are considered to be important in the application, or that can be used in later inferencing. In the present application, the concepts of CAUSE, QUANTITY, and BAD are deemed important, and the concepts FAILURE, DAMAGE, and LOSS can be used in inferencing. Certain quantities (zero in particular) and the categories FAILURE, DAMAGE, and LOSS are also subclassified as representing BAD conditions.

- **catgz.cause**: Assign a category of CAUSE to any of the following words: *affect, cause, make, produce, render, result*.
- **catgz.quant**: Assign a category of BAD to any element modified by the word *in-excess-of*.
- **bad.part.malfunc**: In statements where a STATUS entry co-occurs with a PART entry and the STATUS entry has been categorized as BAD, assign the category of MALFUNCTIONING to the PART.

- **catgz.damage:** Categorize a STATUS entry containing any of the following words as DAMAGE: *bent break broken burn chip clog clogged corrode corroded crack damage erode erosion leak scour seize seized shear sheared split warp warped wear wiped worn.*
- **catgz.failure:** Any of the following words in a STATUS entry indicate FAILURE: *abort aborted bad drop erratic erratically failed error fail failure fault faulty improper improperly inadequate loud low malfunction poor slippage slow spot vibration wrong.*
- **catgz.loss:** Any of the following words in a STATUS entry indicate LOSS: *inop inoperative lack lose loss lost unable unusable.*
- **damage.is.bad:** If something has been categorized as DAMAGE, then it should also be categorized as BAD.
- **failure.is.bad:** If something has been categorized as FAILURE, then it should also be categorized as BAD.
- **loss.is.bad:** If something has been categorized as LOSS, then it should also be categorized as BAD.
- **neg.func.malfunc:** The PART in “PART not FUNC” is categorized as MALFUNCTIONING.
- **damaged.part:** A PART co-occurring with a STATUS which has been categorized as DAMAGE is given the category of DAMAGED.

The two rules contained in the **Negated.good.status** class simply check to see if something that was **not** categorized as BAD was modified by a negation or a zero quantity. If such an occurrence is found, the classifier of BAD is added. They are in a separate rule class from **Categ** because they are conditional on certain information *not* having been found by **Categ**.

- **neg.to.bad.status:** If a STATUS entry has not been categorized as BAD but is modified by a negation, then it is categorized as BAD. Example: “not good” → “bad.”
- **zero.to.bad:** If a quantity of ZERO modifies a STATUS entry and the STATUS entry has not already been categorized as BAD, categorize the STATUS as BAD.

General.infer contains a rule that performs domain independent deduction of the existence of a bad state. This class was created to distinguish such domain independent rules from the domain-specific causal and state deduction rules in MODEL.

- **smoke.fire:** If “X cause Y to be bad” then infer that “X is bad” as well.

The **Score** class contains all the rules that assign a score to anything in the information formats. The scores assigned in these rules reflect what patterns of information are considered important in this application.

- **mention.part:** Add 1 point to the format containing a mention of a PART because we are interested in information about parts.
- **investigation.show:** In a statement of the form “INVEST show X,” where the connective is either of the words *show* or *indicate*, add 1 point to X because it indicates what an investigation revealed.

- **evidence:** Add 1 point to any statement containing an element modified by an evidential modifier such as *show, indicate, reveal* because evidence could be an indicator of the root problem. This rule is like **investigation.show**, but handles cases where the evidential word is formatted as a modifier rather than a connective.
- **cause:** In a statement of the form “X caused Y,” add 2 points to X because causes are considered important.
- **cause.may:** In statements of the form “x may have caused y,” the rule **cause** will add 2 points because of the word “cause.” This rule then subtracts 1 point, because the modal indicates uncertainty about the causal relationship.
- **modal.obj:** Add 1 point to the object of a statement of the form “x appears y” because this represents an opinion about a condition. Connectives that trigger this rule are *suspect, appear, believe*.
- **modal.modifier:** This rule is the same as **modal.obj**, but handles cases where the modal word has been formatted as a modifier rather than a connective.
- **find:** Add 1 point to the object of a statement of the form “we found x” because we are interested in the findings of any investigation or discovery. Connectives that trigger this rule are *find, determine, discover*.
- **neg.func.or.process:** Add 8 points to the statement because it indicates that something is not functioning.
- **universal:** Subtract 1 point from a statement containing any element modified by any of the words *all, each, every* because these words indicate that the message writer is making generalizations, and we consider specifics to be more important.
- **problem.is:** Add 1 point to the object of any statement of the form “the X is Y,” where X is one of the following words: *damage, difficulty, defect, failure, fault, malfunction, problem*. The motivation behind this rule is that we are interested in the root of any problem mentioned in the message.
- **bad.status:** Add 10 points to any statement containing a status which has been categorized as BAD. This is considered one of the most important criteria in this application.
- **piece.evidence:** Add 1 point to a statement containing a PIECE entry (particles, fragments, chunks) because such words are evidence of damage somewhere in the system.

THE EQUIPMENT MODEL

The SUMMARY system, before the addition of the domain model, did a certain amount of domain-independent inferring of equipment state and causality: knowing that *part A impaired part B* means that A caused B to be in a negative state, it then inferred that part A must be in a negative state as well. Conversely, however, the message writer might simply assert that parts A and B are both in a negative state and leave the *impair* relation implicit, knowing that the reader would infer this relations by using domain knowledge. The primary role of the equipment model in TERSE is to perform inferences of this sort.

Domain knowledge can also be used in the natural language analysis phase to determine the correct scope of linguistic constituents. For example, in the sentence *Low lube oil and fail to engage alarms sounded*, the correct parse is derived by analyzing the sentence as having a conjoined subject composed of the elements *low lube oil alarm* and *fail to engage alarm*. An incorrect parse derives if we analyze the components of the subject as *low lube oil* and *fail to engage alarms*, which leads to the analysis “the low lube oil sounded, and the fail to engage alarms sounded.” A domain-knowledge approach to ruling out this bad parse is to recognize that lube oil’s function is not to sound but to lubricate, and that there is only one *Fail To Engage Alarm*. Our approach to selection, however, has been to use distributionally determined semantic patterns to constrain analyses. The bad parse above is ruled out by disallowing the conjoining of a STATUS (*low lube oil*)* with an ALARM (*fail to engage alarms*). Hence the model is not required to take part in the selection process.

Two basic approaches are used to understand natural language with a structured domain model. The first is *frame instantiation*, a common technique in knowledge-based natural language analysis. By using this technique, the phrase *starting air regulating valve* locates the VALVE frame and creates an instance of its subclass REGULATING-VALVE. The frame’s *:substance* slot is filled with a pointer to a newly created instance of the AIR frame, whose *:function* slot is filled with the value START. The resulting instance contains general knowledge about what valves do, that this particular valve functions to regulate air, and that the air’s function is to start something. However, it does not know what the valve’s specific role is, or what it is that *starting air* starts. This approach is equivalent to the knowledge we possessed when we first began to work with compressor CASREPs — basic understanding of mechanical equipment, but little specific knowledge about the particular equipment in question. We could guess about equipment status and causality, but without specific knowledge about the particular equipment configuration, many of our guesses turned out to be incorrect.

The second approach, which is the one we have taken, is to build a completely instantiated model containing the actual components known to comprise the equipment in question. The phrase *starting air regulating valve* is then used to locate in the model all those individual components that are valves, whose function is to regulate, and whose processed-substance is an instance of *air* that has *starting* functionality. In such a model we can incorporate not only general knowledge, but also the specific information about what the air starts (the turbine) and where the valve fits into the overall architecture of the compressor. This knowledge corresponds to the expertise of a reader familiar with the particular equipment being discussed, and allows for increased and more accurate inferencing. The drawback to this approach is that we must construct a new equipment model in detail for each new domain (such as electrical system, propulsion system, and air system) of CASREP being processed.

In constructing the model, we consulted Navy equipment manuals, which listed about 350 individual components; 85 of these were incorporated into the model. These 85 components included all the primary functional components of the compressor, and excluded low-level accessory items such as pins, bolts, and washers that did not contribute substantially to a functional understanding of the equipment’s behavior. The lowest level functional items represented were atomic components such as individual gears and shafts. Message writers would frequently refer generically to more primitive physical components such as the splines on a shaft, the teeth on a gear or the blades on an impellor, so these features were also included in the model, but modeled as collective entities rather than as distinct units.

*The selection process also involves attribute computation; in this instance there is a predicate adjective *low* of semantic class STATUS, which causes the STATUS attribute to be computed for the noun phrase as a whole.

The distributionally determined semantic categories that we use represent the basic classes of information that are talked about in CASREPs. The semantic patterns in which these categories occur represent the fundamental types of predications that the CASREP uses to communicate (such as malfunction, investigation, determination of causality or equipment state, or repair action). These explicit predication types serve as our guideline in determining what sort of inferences should be made when using the equipment model. In the message of Fig. 1, a knowledge of the equipment domain leads an expert reader to infer that a causal relationship is being implied between the worn splines and the drive shaft rotating freely, and between the drive shaft rotating freely and the dropping oil pressure. The causal predications were not stated explicitly because the inferences involved are simple ones for the reader to make. Other causal inferences could also be made regarding this message fragment — for instance, that the dropping oil pressure caused the alarm to sound, or that the alarm sounding caused an investigation to take place. However, these inferences are of the “common-sense” variety and do not represent predications that are likely to be explicitly asserted in a Casualty Report, since its communicative function is to report on the causes of equipment failure only. The role of domain-specific knowledge in our application is thus to extract those implied predications of equipment state and causality that could have been stated explicitly but were not because of the simple inferencing involved. To infer common-sense causal relations is not necessary for our application, and to attempt to infer information that the writer had not intended to convey would begin to enter the realm of *fault diagnosis*, which falls outside the task of message understanding.

Domain models can range from simple attribute lists to complex physical simulations, depending on the needs of the current and anticipated future applications of the system. Since it appeared that the inferences the model would have to make in our application would be only easy ones, we elected to implement as simple an equipment model as possible that would be capable of making these inferences. Since it is necessary to construct a new model for each CASREP equipment, we need for the model construction process to be as straightforward as possible. Finally, it was important not only to match the grain of the model against its intended use, but also against the granularity of the input data it would be operating on (information formats). The correctness and content of the formats are largely determined by heuristic means (distributional co-occurrence patterns of broad semantic classes). The extent to which inferences about message content can be made is constrained by the amount and consistency of relational information available in the format representation. Since both the information formatting and SUMMARY system were low-grain in approach, we felt that a similarly heuristic domain model would be a link of comparable strength in this chain.

Figure 6(b) shows the data flow through the MODEL knowledge base. The first step in using the equipment model is to dereference each equipment name in the information format data against the units in the model. This involves a matching phase and a disambiguation phase. We can, then, use the equipment model to perform domain-specific inferencing about the causal relationships between the pieces of equipment named in the message. These steps are discussed in the sections that follow.

Nominal Dereferencing

In technical material such as equipment casualty reports, complex names containing long sequences of premodifiers and postmodifiers are common [7]:

- *ship's turbine start air system*
- *muffler assembly body end flange*
- *discharge hose from surge air valve assembly*
- *low-speed coupling from diesel to SAC lube oil pump*
- *number 4 SAC 21 inch woven cotton covered rubber surge air hose*

It is also clear from the message data that writers often do not refer to pieces of equipment by their proper names as given in the Navy manuals, but instead they use a variety of descriptive naming conventions. For example, we have the following references to the part named **drive shaft**:

- *shaft*
- *drive shaft*
- *SAC shaft*
- *compressor shaft*
- *input shaft*
- *spline shaft*
- *SAC drive shaft*
- *SAC input shaft*
- *input spline shaft*
- *input drive shaft*
- *spline drive shaft*
- *SAC input drive shaft*
- *splined input drive shaft*
- *SAC spline input drive shaft*
- *connecting shaft*
- *581732-1*

Clearly, we could not simply provide the **drive shaft** unit with a synonym list of all possible names that could be used for it, since new data will likely present unanticipated new names. Our solution was to recognize that left modifiers in equipment names fill only a limited set of descriptive roles. These roles represent either attributes of the equipment or relations of the equipment to other components. In an object-oriented representation, the attributes can be represented as slots attached to the equipment classes in the domain model. The following slot-filler representation of the **drive-shaft** unit, for example, captures all the modifier roles exhibited by the previous list of references:

```
Unit <drive-shaft> =
  :name "drive shaft"
  :partno 581732-1
  :isa shaft
  :function drive
  :supercomponent <starting-air-compressor>
  :components <drive-shaft-spline>
  :input-from <drive-adapter-hub>
  :output-to <ring-gear-hub>
```

In *spline shaft*, the shaft is described by a distinctive *component* (e.g., *wheel chair*). In *SAC shaft*, it is described by its *supercomponent* (e.g., *automobile engine*). Since the shaft is a component of the starting air compressor but the equipment the shaft gets input from (drive adapter hub) is not, the shaft can be described as a *SAC input shaft*. And since the shaft connects the drive adapter hub and ring gear hub, it can be referred to as a *connecting shaft*. In a more complex nominal such as *starting air compressor shaft*, the match must be recursive. To determine if *starting air compressor* matches the unit in the shaft's *:supercomponent* slot, it must first determine that *starting air* matches the unit in *<starting-air-compressor>*'s *:substance* slot. This in turn requires that *starting* match some attribute of the substance. The head noun of an equipment name must match either the object's class (*:isa*), or a value in either its *:partno* or *:aka* slots. *:Aka* is used primarily for acronyms, such as *SAC*, *GTRB* (gas turbine) and *SSDG* (ship's service diesel generator).

Slots can contain either atomic values (*drive*) or pointers to other units (<**drive-shaft-spline**>). The links to other units form the model into a directed multigraph. Methods attached to the equipment units in the model perform link traversals to determine if two units have a part-of or functional-connectivity relationship. For example, the method *is-directly-connected-to* can determine that <**starting-air-compressor**> and <**drive-adapter**> have a direct functional connection to each other, because (1) neither is a subcomponent of the other, and (2) a subcomponent of the first (<**drive-shaft**>) has an *:input-from* link to a subcomponent of the second (<**drive-adapter-hub**>).

The following are the attributes we have found necessary to capture the primary left-modifier roles in CASREP equipment names:

- partof (*pump shaft*)
- parts (*spline shaft*)
- subclass (*turbine engine*)
- input-from (*diesel hub*)
- output-to (*ring gear hub*)
- function (*regulating valve*)
- substance (*oil pump*)
- behavior (*lift valve*)
- shape (*ring gear*)
- fuel (*gas turbine*)

Matching

Since the syntactic analyzer uses only broad classes of semantic information, for the most part it is unable to structurally analyze complex nominals in a semantically significant manner. As a result, PART entries in the information formats consist simply of the head noun accompanied by the flat text string of the entire nominal. The matching of these nominal strings against model objects is done in the postsyntactic phase using another parser, the chart parser PROTEUS [8]. PROTEUS provides a convenient and efficient computational mechanism for generating host-modifier bindings in the string, testing each of these against the equipment model, and returning pointers to those model objects that allowed an analysis to succeed. We are currently transitioning to PROTEUS as our syntactic analyzer as well, so this approach may make it possible to interleave the model-based nominal dereferencing with the syntactic analysis, greatly reducing the problems of noun phrase isolation and sentential ambiguity.

PROTEUS grammars consist of context-free BNF (Backus-Naur Form) productions augmented by arbitrary test conditions called *restrictions*. In our application, the BNF component generates all the linguistically feasible host-modifier analyses of the string, and the restrictions then test each candidate analysis for semantic validity against the equipment model. The BNF grammar and dictionary used to drive the parse have as their preterminal symbols the semantic categories PART, FUNC, PROP, STATUS, and ALARM, as follows:

1. <UNIT> ::= <EQUIP> | <PROPERTY>.
2. <EQUIP> ::= <PART> | <ATOM> <EQUIP> | <UNIT> <EQUIP> | <EQUIP> "assembly" | <ALARM>.
3. <PROPERTY> ::= <PROP> | <EQUIP> <PROPERTY> | <STATUS> <PROPERTY>.
4. <ATOM> ::= <FUNC> | <STATUS> | <SHAPE>.
5. <SHAPE> ::= <PART>.

Production 1, for example, states that the nominals to be matched against the model are either names of pieces of equipment or names of equipment properties (such as *oil pressure*). Production 2 describes the various ways that an equipment name might be analyzed, such as PART (*pump* or *oil*), ATOM EQUIP (*lube oil*), or UNIT EQUIP (*oil pump*, *lube oil pump*). The recursive definition of EQUIP allows *lube oil pump* to be analyzed as both

```
(UNIT
 (EQUIP
  (ATOM (FUNC lube))
  (EQUIP (UNIT (EQUIP (PART oil))))
  (EQUIP (PART pump))))))
```

(i.e. a “lubricating oil-pump”), or

```
(UNIT
 (EQUIP
  (UNIT
   (EQUIP
    (ATOM (FUNC lube))
    (EQUIP (PART oil))))
  (EQUIP (PART pump))))
```

(i.e., a “lubricating-oil pump”).

We want the first analysis to be considered semantically inconsistent by the equipment model. During each analysis, restrictions attached to the EQUIP production consult the model to see if an object can be located whose structure matches the analysis. For example, restriction VALID-EQUIP accepts *turbine*, *GTRB*, and *LM2500* as minimal equipment names because they are respectively the class, acronym, and part-number of the <gas-turbine> unit in the model. VALID-ATOM-MODIFIER accepts *lube oil* because one of the units matched by *oil* has *lubrication* as a value of its *:function* slot. Also, VALID-UNIT-MODIFIER accepts *pump shaft* because at least one of the units that matches *shaft* is a subcomponent of one of the units that matches *pump*. If a restriction succeeds, it adorns the root node of the parse with a pointer to the matched model object(s), and signals success. The algorithms of these restrictions on EQUIP are as follows:

```
VALID-EQUIP (in production EQUIP → PART):
  loop for model-object in model do
    if either model-object is in class PART
    or the part-number of model-object is PART
    or an acronym of model-object is PART
    then add model-object to the MATCHING-UNITS attribute of the present node
  if MATCHING-UNITS return true else false
```

```
VALID-ATOM-MODIFIER (in production EQUIP → ATOM EQUIP):
  loop for model-object in MATCHING-UNITS of element EQUIP do
    if some slot of model-object contains ATOM
    then add model-object to the MATCHING-UNITS attribute of the present node
  if MATCHING-UNITS return true else false
```

```
VALID-UNIT-MODIFIER (in production EQUIP → UNIT EQUIP):
  loop for model-object-1 in MATCHING-UNITS of element UNIT do
```

```

loop for model-object-2 in MATCHING-UNITS of element EQUIP do
if either
  model-object-2 is a subcomponent of model-object-1
  or model-object-2 is-directly-connected-to model-object-1
  or some other slot of model-object-2 contains model-object-1
  then add model-object-2 to the MATCHING-UNITS attribute of the present node
if MATCHING-UNITS return true else false

```

As an example of how the matching proceeds, we consider the nominal *lube oil pump shaft* and the dictionary entries

```

(lube (FUNC))
(oil (PART))
(pump (PART)(FUNC))
(shaft (PART))

```

1. The parser begins by entering (FUNC *lube*) in the chart, which it then analyzes as ATOM using the production $\langle \text{ATOM} \rangle ::= \langle \text{FUNC} \rangle$.
2. On entering (PART *oil*) in the chart, the parser must first verify the constraint VALID-EQUIP before it can analyze this word as an EQUIP using the production $\langle \text{EQUIP} \rangle ::= \langle \text{PART} \rangle$. Since the model contains at least one object that *isa oil* — $\langle \text{diesel-oil} \rangle$ and $\langle \text{lube-oil} \rangle$ — the restriction succeeds. The EQUIP node gets a pointer to those two objects.
3. The parser then attempts to conjoin nodes 1 and 2 into a single constituent, using option 2 of production 2. This requires that the restriction VALID-ATOM-MODIFIER succeed. It does, because the $\langle \text{lube-oil} \rangle$ object pointed to by node 2 has “lubrication” as the filler of one of its attribute slots (*function*). This node now contains a pointer to $\langle \text{lube-oil} \rangle$ only.
4. *pump* is entered in the chart in its FUNC sense, and analyzed as an ATOM.
5. When *pump* enters the chart in its PART sense, it is analyzed as EQUIP with a pointer to the only pump in the model, $\langle \text{oil-pump-assembly} \rangle$. This node is then permitted to conjoin with both node 2 (i.e. “oil pump”) and node 3 (“lube-oil pump”) by restriction VALID-EQUIP-MODIFIER, because the pump has a slot (*substance*) containing a pointer to $\langle \text{lube-oil} \rangle$. However, it cannot conjoin with node 1 (“lube pump”) because the pump does not have “lubrication” as an attribute.
6. On processing the final word *shaft*, a large ambiguity set of objects is found to match that word. None of these objects can conjoin with nodes 1, 2, 3 or 4, indicating that there is no “lube shaft,” “oil shaft,” or “pumping shaft” in the model. This node conjoins with node 5, however, since one object — the $\langle \text{pump-drive-shaft} \rangle$ — has a *supercomponent* link to $\langle \text{oil-pump-assembly} \rangle$. The root node of the completed parse now contains a pointer to that set of model objects that the entire string successfully matched (the single object $\langle \text{pump-drive-shaft} \rangle$).

Based on the BNF alone, there are 14 possible interpretations of *lube oil pump shaft*. For example, it might mean a “shaft that pumps lube oil,” just as *start air check valve* is a valve that checks start air. Analyses such as this are rejected by the model, since no such shaft exists. The only analysis that matches a domain object is “shaft that is a component of a pump such that the pump

processes lube oil,” since that type of oil exists, there is a pump that processes it, and there is a shaft that is a component of that pump.

Also the BNF tells the matcher whether a modifier can represent a structured unit in the model (PART, PROP) or an atomic slot value (FUNC, STATUS, SHAPE). Since at the present time we do not have a semantic category representing the concept of shape, we allow PART to serve in both roles to handle such nominals as *ring gear*, where *ring* is not referring to a piece of equipment (e.g., an o-ring), but to the *:shape* attribute of the gear.

Contextual Disambiguation

Frequently the matching described in the previous section does not identify a single domain object, but it returns an ambiguity set of several possible referents, because of underspecification by the message writer. Underspecification can have several sources:

- The writer has fully specified a piece of equipment earlier in the message and is now referring to it in an abbreviated manner.
- The message class can be used to understand the reference (in SAC CASREPs the name *unit* always refers to the starting air compressor).
- The discourse context can be used to disambiguate the reference (*alarm sounded* can be disambiguated by using the previous message clause *lube oil pressure was dropping*).

The maintenance of a *focus list* [9] is a common method for dealing with such referential ambiguities. This is a list of entities that have already been referred to in the message and are used for resolving subsequent ambiguous references. Techniques such as spreading activation or marker-passing [10] can determine semantic relationships between the prior and subsequent referents. Our disambiguation algorithm most closely resembles spreading activation, in that it takes all the candidate referents and finds the subset of these that forms the most tightly connected cluster of linked nodes in the equipment model graph. The algorithm operates as follows.

After matching, each PART/PROP/ALARM nominal in the message has an associated *matching-units* set of possible referents. First, unambiguous (singleton) matching-units sets are deleted, and their content is transferred to the *referent* attribute of the matched nominal. Each matching-units set that remains represents an ambiguity set (unresolved referent). The members of these ambiguity sets are then scored by the following procedure.

Each resolved referent is paired with each ambiguity set member that follows it in the message. Each ambiguity set member is scored by how closely it is linked to the referent, both by model links (such as *:component* and *:input-from*) and by the distance separating the two referencing assertions. The algorithm then tries to find connections between each matching-units set member and each resolved referent and ambiguity set member in subsequent formats. After all scoring is finished, the highest scoring item in each ambiguity set is chosen as the referent of the corresponding nominal. Thus, even if every part reference in a message were to be underspecified, the most tightly connected subgraph of the model that matches the set of nominals will be returned, with a high likelihood of representing the components in question. The algorithm itself is as follows:

For each format *fl* do:

fl-matches ← merge all matching-units sets of entries in *fl*

fl-referents ← collect all referents of entries in *fl*

```

For each i in f1-referents do:
  For each format f2 in subsequent formats do:
    f2-matches ← merge all matching-units sets of entries in f2
    For each j in f2-matches do
      score ← SCORE (i, j)
      Add score to j's score
  For each i in f1-matches do:
    For each format f2 in subsequent formats do:
      f2-matches ← merge all matching-units sets of entries in f2
      f2-referents ← collect all referents of entries in f2
      For each j in f2-referents do
        score ← SCORE (i, j)
        Add score to i's score
      For each j in f2-matches do
        score ← SCORE (i, j)
        Add score to both i's and j's score

```

Function SCORE (*i*, *j*):

```

link-distance ← either
  0 if i and j are the same object;
  the number of :input-from or :output-to links separating i and j;
  the number of :supercomponent or :components links separating i and j;
  0 if some other slot (e.g. :substance) of one item points to the other item;
  otherwise, nil.
If link-distance = nil, return 0
else
  format-distance ← |format.no (f1) - format.no (f2)|
  return 1/(1 + link-distance) + 1/(1 + format-distance)

```

As an illustration, in the message fragment shown in Fig. 8, all part references are unambiguous except for *spline shaft*, which matches four objects in the model. This is the first (and only) mention of this piece of equipment in the message, so we cannot count upon the first reference to an entity to be unambiguous. The disambiguation routine generates the score matrix shown in Fig. 9, where each row is a match and each column is another object referred to in the message. The <**drive-shaft**> unit gets the highest score, primarily because it is more closely linked in the model to the <**drive-adapter-hub-spline**> and <**ships-service-diesel-generator**> (SSDG) than are the other shaft units. For instance, the <**drive-shaft**> is separated from the SSDG by only two functional-connectivity links (score = 0.33), whereas the <**gear-shaft**> is separated from the SSDG by six such links (score = 0.14). Since the format mentioning the SSDG immediately follows the one mentioning the shaft, a format-distance score of 0.5 is added to obtain the matrix entry for each.

*While conducting operational checks, experienced engagement malfunction.
 Discovered that the engine drive adapter hub internal splines had sheared,
 causing the SAC to fail to engage.
 Spline shaft rcvd w/assist from Desron 8.
 Installed on #3 SSDG.
 Tested sat on 25 Feb.*

Fig. 8 — Message 2

	SPLINE	SAC	SSDG	TOTAL
DRIVE SHAFT	0.83	0.83	0.83	2.50
GEAR SHAFT	0.50	0.83	0.64	1.98
COUPLING DRIVE SHAFT	0.47	0.83	0.62	1.93
WHEEL SHAFT	0.40	0.75	0.57	1.72

Fig. 9 — Disambiguation score matrix

The MODEL Rule Base

The MODEL knowledge base also contains a set of equipment-general rules that, in conjunction with the equipment model, can infer equipment state and causal relations from the content of a casualty report. Several rules also take part in the nominal dereferencing process.

The root rule class *Model-rules* is divided into four rule sets: *Preprocess*, *Postmatching*, *Find-referents*, and *Inferencing*.

The *Preprocess* rule set is run before nominal matching against the equipment model is undertaken. It contains one rule at present.

generic-component-head: for a PART entry that ends in the word *part* or *component* (such as *valve parts*), create a new PART unit representing the left modifier (in this case, *valve*) for the matcher to dereference, and assert that the new PART entry has a *supercomponent-part* relationship to the current entry (e.g. *valve* is the supercomponent of *valve parts*).

The *Postmatching* rule set is run after matching and prepares the system to perform nominal disambiguation.*

is-property-of-same-format: if a PROP entry co-occurs with a PART entry, and an equipment unit matching the PROP is the same as the *:output-property* of the referent of the PART, then assert that the unit is the referent of the PROP. Example: *pump pressure* is formatted into the two entries PART (*pump*) and PROP (*pressure*). Assert that *pressure* refers to **<oil-pressure>** because that is the output property of the oil pump.

has-property-same-format: if a PROP entry co-occurs with a PART or ALARM entry, and an equipment unit matching the PART/ALARM has as either its *:trigger-condition* or *:output-property* the referent of the PROP, then assert that the equipment unit is the referent of the PART/ALARM. Example: in *oil pressure alarm*, assert that the referent of *alarm* is **<lube-oil-pressure-alarm>** because **<oil-pressure>** is the trigger condition of the alarm.

has-substance-same-format: if a PART entry co-occurs with a PROP or ALARM entry, and the referent of the PART is the *:substance* of an equipment unit matching the PROP/ALARM, then assert that the equipment unit is the referent of the PROP/ALARM. Example: in *oil pressure*, assert that the referent of *pressure* is **<oil-pressure>** because its *:substance* is **<lube-oil>**.

*Normally a noun phrase like *pump pressure* would be handled entirely by the matcher. However, the formatting component splits certain PART/PROP/ALARM nominals into more than one format entry, making it appear to the matcher that the head noun (i.e., *pressure*) is an underspecified nominal. The first three rules in this rule set can prevent an unnecessary call to the disambiguation routine by placing the head noun back in its original context.

name-matches-exactly: if an equipment unit that matches a PART entry has exactly the same name as the PART entry text string, increase the score of this match by 1 point.

only-match: if a PART/PROP/ALARM entry only matched a single unit in the equipment model, then assert that the unit is the referent of the entry (disambiguation is unnecessary).

delete-matches: if the referent of an entry has been found, then delete all other candidate matches of the entry.

After the disambiguation/focusing routine is run, the *Find-referents* rule set selects the highest scoring match for each nominal, and also explicitly asserts the existence of any assembly/component or input/output links that exist between named units.

highest-score: if an entry has more than one matching equipment unit, select as the referent of the entry the match having the highest score.

supercomponent-part: if the referent of one PART entry is the *:supercomponent* of another PART entry, assert that the first entry has a *supercomponent-part* relation to the second entry.

input-from-part: if the referent of one PART entry is functionally downstream from the referent of another PART entry, assert that the first entry has an *input-from-part* relation to the second entry.

Finally, the *Inferencing* rule set performs model-based causal and state inferencing on the contents of the message.

infer-bad-supercomponent: if a STATUS entry cannot be classified as either BAD or normal, and co-occurs with a PART entry that has a *supercomponent-part* relation to another PART entry whose state is MALFUNCTIONING, then infer that the classification of the STATUS entry is BAD.

cause-bad-supercomponent: if a PART entry in one assertion is the *supercomponent-part* of a PART entry in another assertion, and both PARTs are MALFUNCTIONING, then infer that the condition in the first assertion was *caused* by the condition in the second assertion.

cause-bad-downstream: if a PART entry in one assertion has an *input-from-part* relation with a PART entry in another assertion, and both PARTs are MALFUNCTIONING, then infer that the condition in the second assertion *caused* the condition in the first assertion.

infer-alarm-trigger-property: if an assertion mentions that an ALARM went off, and it has not been asserted elsewhere in the message that the *:trigger-condition* of the ALARM is in a BAD state, then make just such an assertion.

direct-cause-bad-property: if an assertion states that a PART is MALFUNCTIONING, and another assertion states that the *:output-property* of the PART's referent is BAD, then assert that the condition in the first assertion *caused* the condition in the second assertion.

indirect-cause-bad-property: if an assertion states that the *:output-property* of some equipment unit is BAD, and another assertion states that a PART is MALFUNCTIONING, and the PART's referent is either functionally upstream from or is a supercomponent of the

equipment unit, then assert that the condition in the second assertion *caused* the condition in the first assertion.

causal-chain: if condition A *caused* condition B, and condition B *caused* condition C, then assert that condition A also *caused* condition C.

In addition to these rules, the MODEL knowledge base also contains four LISP methods for doing link traversals of the equipment model, and one method to run the pattern matcher.

Model-Based Inferencing

Just as the part-whole and functional-connectivity links in the model were used heuristically to dereference equipment names, these links can also be used to perform heuristic inferring of equipment state and causality. In the message shown in Fig. 10, the dereferencing procedures first determine that *valve* in the final sentence refers to the same object as *starting air regulating valve* in the first sentence. Since both the valve and its parts are in a negative state, this leads us to suspect that the writer is implying a causal relationship between the two predications. Either the latter is providing an explanation for the former (in which case we infer that *parts corroded* caused *valve failed*), or the latter is describing the result of the former (*valve failed* caused *parts corroded*). We use four heuristics in making causal/state inferences such as this.

<p><i>Starting air regulating valve failed.</i> <i>Unable to consistently start nr 1b gas turbine.</i> <i>Valve parts excessively corroded.</i></p>

Fig. 10 — Message 3

1. Heuristic 1 (still experimental) involves the subclassification of state-predicating words into three groups. The DAMAGE group includes such words as *fail* (when applied to low-level components like gears and bearings), *fault*, *corrode*, *shear*, *damage*, *wear*, *erode*, and *strip*. FAILURE connotes decreased functional behavior and includes such words as *fail* (when applied to higher level components), *low*, *slip*, *drop*, and *seize*, as well as any FUNCTION words that have been negated. LOSS connotes loss of equipment usage by ship personnel and includes words like *loss*, *inoperative*, and *unable*. Our test data indicate that inferrable causality always flows from the more specific classes of state-predicators to the same or more general classes. FAILURE can be inferred to cause either FAILURE or LOSS, but LOSS cannot be inferred to cause FAILURE or DAMAGE, for example. In the message of Fig. 10, this heuristic confirms the inference that *valve parts corroded* (DAMAGE) probably caused *valve failed* (FAILURE). Had the failure caused the corrosion (for example allowing salt water to enter the valve), the inference would be more difficult to make, and the writer would probably have expressed the causal relationship explicitly. Since DAMAGE is the lowest level class of state-predicating words, therefore, we do not attempt to infer causes of damage. Message writers tend to be explicit about causes of physical damage, since the inferences involved are often not straightforward.
2. The second heuristic prefers flow of causality from component to assembly, rather than from assembly to component. Applied to the message of Fig. 10, it leads to the inference that *valve parts corroded* caused *valve failed* rather than vice versa. In our test data we have found no clear instances where an inference of flow of causality from assembly to

component can be made. We have found some instances where the malfunctioning of a piece of equipment resulted in damage to itself or its components, but the writer clearly was aware that the inference would be difficult to make and so described the causality explicitly: *Suspect drive shaft slipped backwards and ground down the teeth [on the shaft], due to installation defect.* Had the writer simply said *Drive shaft slipped backwards. Teeth ground down*, the simpler inference would have been that the worn teeth (DAMAGE) allowed the shaft to slip (FAILURE). So if an assembly and component are in negative states and no explicit causal relationship is mentioned, we infer flow of causality from the component to the assembly.

3. Besides parent-child links, we also use the model's functional-connectivity links (:input-from and :output-to) to infer causal relations. In the message of Fig. 10, the model unit referred to by *starting air regulating valve* has an :output-to link to the unit referred to by *gas turbine*. Since both parts have been described as malfunctioning, Heuristic 3 infers that the valve's malfunction was responsible for the turbine's malfunction, since the latter is functionally "downstream" from the former. This rule is heuristic in that we do not model the precise nature of the functional connectivity (that the valve passes pressurized air to the turbine, the valve is regulating the amount of this air, the turbine requires sufficient air to start, and failure of the valve means insufficient flow of air to the turbine). We simply know that the valve is not performing its function properly and that the turbine is functionally dependent on that valve in some way. It is possible for negative state to flow "upstream" through the system, as for example in the message fragment shown in Fig. 11. Here, the SAC's FAILURE (*seized*) caused its input component *drive shaft* to be in a FAILURE state (*remain stationary*), which in turn caused the shaft's input device *drive adapter hub* to acquire DAMAGE (*shear*). Since these would probably be difficult inferences for the reader to make, the message writer described the flow of malfunction causality explicitly.

*SAC apparently seized during clutch engagement causing input drive shaft to remain stationary while drive adapter hub on SSDG continued to rotate.
Drive shaft sheared all internal gear teeth from drive adapter hub.*

Fig. 11 — Message 4

4. A final use for the domain model is in the disambiguation of equipment status. This is seldom necessary, but does provide the system with additional robustness. Referring again to the message of Fig. 1, *freely* is a status word that cannot simply be classified as FAILURE, because in many contexts it could just as easily serve as a positive indicator. However, the context states that the shaft's splines are worn and that there are *loud noises from drive end*, both of which suggest that the shaft is malfunctioning. Since *freely* is an unclassified status word, the system then infers that *freely* probably indicates malfunction in this case. Using Heuristics 1 and 2, the system can then infer that this malfunction was caused by *splines worn*. Again, this heuristic does not require that the relationship be modeled in any more detail (that resistance between splines is how the shaft transmits torque, rotation is the "carrier" of torque, and *freely* connotes lack of resistance).

Usually, we do not model the functional behavior of each piece of equipment, but simply represent the flow of functionality from one piece to another by links connecting the two. In certain cases, however, it becomes necessary to represent what properties are being added to the functional

medium by a piece of equipment. This is because the CASREP writers frequently refer to certain properties explicitly. To make causal inferences between the property and a piece of equipment, it is necessary to know the property's origin and destination. In the message of Fig. 1, to infer that the drive shaft's rotating freely was the cause of the drop of oil pressure, it is necessary to know that the origin of oil pressure is the oil pump assembly. Since no other possible cause of drop of oil pressure was mentioned (such as an oil leak), the system infers that the pump was malfunctioning. It can then infer that the cause was the malfunctioning of a piece of equipment (the drive shaft) functionally upstream from the pump. As this example illustrates, the system can infer the states of pieces of equipment not mentioned at all in the message (the oil pump assembly), just by following the chain of connectivity links between two units that were mentioned, *drive shaft* and *LO pressure*.

Functional feedback loops within the equipment make causal inferencing more difficult. For example, the drive shaft provides power to the oil pump and enables it to function, but the same oil that the pump pressurizes also lubricates the drive shaft. So in the message of Fig. 1, it is certainly possible that the drop of oil pressure caused the splines to wear, rather than the other way around. In this particular instance, our first heuristic (preferring the progression of malfunction from more specific to more general cases) would resolve the matter, since the causation of DAMAGE (*worn*) by FAILURE (*drop*) is the less likely occurrence. Had the writer not mentioned *splines worn* in the message, however, this approach would not work. It is likely that an expert reader considers lubrication to be only a secondary or support functionality within the equipment, and so prefers an interpretation that involves the primary flow of mechanical functionality. At present, we simply do not represent secondary functionality such as lubrication, because transforming the model from an acyclic to a cyclic graph would greatly increase the complexity of doing network traversals. A solution would be to tag functional links as either primary or secondary. This would add one more dimension to the graph but would allow it to remain acyclic.

SAMPLE RUN

We now present a complete trace generated by the system as it processed the message of Fig. 1. Comments (lines beginning with semicolons) have been added manually for clarification.

The system first reads in the information format representation of the message from a text file. The contents of the file appear as follows.

```

;;; 9.1.1
;;; SAC RECEIVED HIGH USAGE DURING TWO BECCE PERIODS.

(CONN (OP (REL-CLAUSE (HEAD T-EXPAND-REFPT) (TEXT T-EXPAND-REFPT)))           ;; CONN1
  (ARG1
    (FMT (TEXT SAC RECEIVE PAST HIGH USAGE DURING PERIOD PLURAL)           ;; FORMAT1
      (PART (HEAD SAC) (TEXT SAC))
      (FUNC (HEAD USAGE)
        (MODS (EVID (HEAD RECEIVE) (TEXT RECEIVE _ PAST)))
        (TEXT HIGH USAGE))
      (EVNT-TIME (TPREP2 DURING)(REFPT PERIOD))
      (TIME (VTENSE PAST))))
    (ARG2 (FMT (TEXT TWO BECCE PERIOD PLURAL)                               ;; FORMAT2
      (STASK (HEAD PERIOD)(TEXT TWO BECCE PERIOD PLURAL))))

;;; 9.1.2
;;; CCS RECEIVED A REPORT THAT LO PRESSURE WAS DROPPING.

(CONN (OP (EMBEDDED (HEAD EMBEDDED-N) (TEXT EMBEDDED-N)))                 ;; CONN2
  (ARG1
    (FMT (TEXT CCS RECEIVE PAST A REPORT LUBE OIL PRESSURE DROP PAST PROG) ;; FORMAT3
      (ORG (HEAD CCS) (TEXT CCS))
      (MSG (HEAD REPORT))

```

```

(DET A)
(MODS (EVID (HEAD RECEIVE) (TEXT RECEIVE _ PAST)))
(TEXT REPORT LUBE OIL PRESSURE DROP PAST PROG)))
(ARG2
(FMT (TEXT LUBE OIL PRESSURE DROP PAST PROG)                ::: FORMAT4
(PROP (HEAD PRESSURE) (TEXT PRESSURE))
(PART (HEAD LUBE — OIL) (TEXT LUBE OIL))
(TIME (VTENSE PAST _ PROG))
(STATUS (HEAD DROP) (TEXT DROP _ PAST _ PROG))))

;;; 9.1.3
;;; ALARM SOUNDED.

(FMT (TEXT ALARM SOUND PAST)                                ::: FORMAT5
(ALARM (HEAD ALARM) (TEXT ALARM))
(FUNC (HEAD SOUND) (TEXT SOUND PAST))
(TIME (VTENSE PAST)))

;;; 9.1.4
;;; LOUD NOISES WERE COMING FROM THE DRIVE END DURING COAST DOWN.

(CONN (OP (REL-CLAUSE (HEAD T-EXPAND-REFPT) (TEXT T-EXPAND-REFPT)))  ::: CONN3
(ARG1
(FMT (TEXT LOUD NOISE PLURAL COME PAST PROG FROM THE        ::: FORMAT6
DRIVE END DURING COAST DOWN)
(STATUS (HEAD NOISE) (TEXT LOUD NOISE PLURAL))
(PARTLOC (HEAD COME) (TEXT COME PAST PROG))
(AREA (HEAD END) (DET THE) (TEXT DRIVE END))
(EVNT-TIME (TPREP2 DURING) (REFPT COAST _ DOWN))
(TIME (VTENSE PAST _ PROG))))
(ARG2
(FMT (TEXT COAST DOWN) (STASK (HEAD COAST — DOWN) (TEXT COAST DOWN))))  ::: FORMAT7

;;; 9.1.5
;;; DRIVE SHAFT WAS FOUND TO ROTATE FREELY AT THE SSDG END.

(CONN (OP (RELATION (HEAD FIND) (TEXT FIND PAST) (TIME (VTENSE PAST))))  ::: CONN4
(ARG1 (FMT)  ::: FORMAT8
(ARG2
(FMT (TEXT DRIVE SHAFT ROTATE FREELY AT THE SSDG END)  ::: FORMAT9
(PART (HEAD SHAFT) (TEXT DRIVE SHAFT))
(STATUS (HEAD FREELY)(TEXT FREELY))
(FUNC (HEAD ROTATE) (TEXT ROTATE))))))

;;; 9.1.6
;;; SPLINES WERE EXTENSIVELY WORN.

(FMT (TEXT WEAR PAST EXTENSIVELY SPLINE PLURAL)            ::: FORMAT10
(STATUS (HEAD WEAR) (TEXT EXTENSIVELY WEAR PAST))
(PART (HEAD SPLINE) (TEXT SPLINE — PLURAL))
(TIME (VTENSE PAST)))

```

Now the system performs matching of the TEXT strings of all the PART/PROP/ALARM entries in these formats against the equipment model, finding either singleton matches or ambiguity sets.

```

(SAC) matches SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR
(LUBE OIL) matches LUBE-OIL
(DRIVE SHAFT) matches COUPLING-DRIVE-SHAFT
(DRIVE SHAFT) matches DRIVE-SHAFT
(DRIVE SHAFT) matches OIL-PUMP-SHAFT
(SPLINE) matches DRIVE-SHAFT-SPLINE
(SPLINE) matches DRIVE-ADAPTER-HUB-SPLINE

```

(SPLINE) matches WHEEL-SHAFT-SPLINE
 (SPLINE) matches GEAR-SHAFT-SPLINE
 (SPLINE) matches COUPLING-DRIVE-SHAFT-SPLINE
 (ALARM) matches COMPRESSOR-FAIL-TO-ENGAGE-ALARM
 (ALARM) matches LUBE-OIL-PRESSURE-ALARM
 (PRESSURE) matches OIL-PRESSURE
 (PRESSURE) matches AIR-PRESSURE

Next, the system performs some simple predisambiguation routines. Although *SAC* only matches one model unit, each of the other nominals matches more than one unit and needs further disambiguation. (The nominal *lube oil pressure* had been decomposed by the formatting component into two entries, leaving the head noun *pressure* ambiguous even though the entire noun phrase is not; rule **has-substance-same-format** quickly “disambiguates” *pressure* by assuming that *lube oil* is part of its description.) Rule **name-matches-exactly** increases the weight of a match in which the name used in the message is the same as the actual name of the piece of equipment as given in the Navy manuals. Rule **only-match** asserts that a singleton match is the correct referent of a nominal, and once the referent is asserted, rule **delete-matches** deletes all the tentative candidates.

Running rule NAME-MATCHES-EXACTLY in world *BACKGROUND*
 (DRIVE SHAFT) matches the proper name of #[Unit: DRIVE-SHAFT MODEL]

Running rule NAME-MATCHES-EXACTLY in world *BACKGROUND*
 (LUBE OIL) matches the proper name of #[Unit: LUBE-OIL MODEL]

Running rule ONLY-MATCH in world *BACKGROUND*
 believe that the referent of (LUBE OIL) is LUBE-OIL
 Adding (THE REFERENT OF FORMAT4-PART IS LUBE-OIL) to *BACKGROUND*

Running rule DELETE-MATCHES in world *BACKGROUND*
 Retracting (A MATCH OF FORMAT4-PART IS MATCH108) from *BACKGROUND*

Running rule HAS-SUBSTANCE-SAME-FORMAT in world *BACKGROUND*
 believe that the referent of (PRESSURE) is OIL-PRESSURE
 Adding (THE REFERENT OF FORMAT4-PROP IS OIL-PRESSURE)
 to *BACKGROUND*

Running rule DELETE-MATCHES in world *BACKGROUND*
 Retracting (A MATCH OF FORMAT4-PROP IS MATCH119) from *BACKGROUND*

Running rule DELETE-MATCHES in world *BACKGROUND*
 Retracting (A MATCH OF FORMAT4-PROP IS MATCH120) from *BACKGROUND*

Running rule ONLY-MATCH in world *BACKGROUND*
 believe that the referent of (SAC) is SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR
 Adding (THE REFERENT OF FORMAT1-PART IS SHAFT-DRIVEN-AIR-
 CENTRIFUGAL-COMPRESSOR) to *BACKGROUND*

Running rule DELETE-MATCHES in world *BACKGROUND*
 Retracting (A MATCH OF FORMAT1-PART IS MATCH107) from *BACKGROUND*

The truly ambiguous nominals that remain are *alarm*, *drive shaft* and *spline*. Next, the system runs the focusing/disambiguation routine to choose referents for these nominals. Whenever a connection is found between a matched object and some other object referenced in the message, a score is generated and added to the overall score of the matched object. For brevity, we show only those pairings in which such a connection is found.

testing match #[Unit: LUBE-OIL-PRESSURE-ALARM MODEL] in format 5
 against referent #[Unit: SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR MODEL]
 in format 1 score = 0.25

testing match #[Unit: COMPRESSOR-FAIL-TO-ENGAGE-ALARM MODEL] in format 5
 against referent #[Unit: SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR MODEL]
 in format 1 score = 0.33333334

testing match #[Unit: OIL-PUMP-SHAFT MODEL] in format 9
 against referent #[Unit: SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR MODEL]
 in format 1 score = 0.33333334

testing match #[Unit: DRIVE-SHAFT MODEL] in format 9
 against referent #[Unit: SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR MODEL]
 in format 1 score = 0.33333334

testing match #[Unit: COUPLING-DRIVE-SHAFT MODEL] in format 9
 against referent #[Unit: SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR MODEL]
 in format 1 score = 0.33333334

testing match #[Unit: COUPLING-DRIVE-SHAFT-SPLINE MODEL] in format 10
 against referent #[Unit: SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR MODEL]
 in format 1 score = 0.25

testing match #[Unit: GEAR-SHAFT-SPLINE MODEL] in format 10
 against referent #[Unit: SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR MODEL]
 in format 1 score = 0.25

testing match #[Unit: WHEEL-SHAFT-SPLINE MODEL] in format 10
 against referent #[Unit: SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR MODEL]
 in format 1 score = 0.2

testing match #[Unit: DRIVE-ADAPTER-HUB-SPLINE MODEL] in format 10
 against referent #[Unit: SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR MODEL]
 in format 1 score = 0.5

testing match #[Unit: DRIVE-SHAFT-SPLINE MODEL] in format 10
 against referent #[Unit: SHAFT-DRIVEN-AIR-CENTRIFUGAL-COMPRESSOR MODEL]
 in format 1 score = 0.25

testing match #[Unit: LUBE-OIL-PRESSURE-ALARM MODEL] in format 5
 against referent #[Unit: LUBE-OIL MODEL] in format 4
 score = 1.0

testing match #[Unit: LUBE-OIL-PRESSURE-ALARM MODEL] in format 5
 against referent #[Unit: OIL-PRESSURE MODEL] in format 4
 score = 1.0

testing match #[Unit: LUBE-OIL-PRESSURE-ALARM MODEL] in format 5
 against match #[Unit: OIL-PUMP-SHAFT MODEL] in format 9
 score = 0.7

testing match #[Unit: LUBE-OIL-PRESSURE-ALARM MODEL] in format 5
 against match #[Unit: DRIVE-SHAFT MODEL] in format 9
 score = 0.4

testing match #[Unit: LUBE-OIL-PRESSURE-ALARM MODEL] in format 5
 against match #[Unit: DRIVE-ADAPTER-HUB-SPLINE MODEL] in format 10
 score = 0.30952382

testing match #[Unit: LUBE-OIL-PRESSURE-ALARM MODEL] in format 5
 against match #[Unit: DRIVE-SHAFT-SPLINE MODEL] in format 10
 score = 0.33333334

testing match #[Unit: COMPRESSOR-FAIL-TO-ENGAGE-ALARM MODEL] in format 5
 against match #[Unit: OIL-PUMP-SHAFT MODEL] in format 9
 score = 0.34285715

testing match #[Unit: COMPRESSOR-FAIL-TO-ENGAGE-ALARM MODEL] in format 5
 against match #[Unit: DRIVE-SHAFT MODEL] in format 9
 score = 0.3

testing match #[Unit: COMPRESSOR-FAIL-TO-ENGAGE-ALARM MODEL] in format 5
 against match #[Unit: COUPLING-DRIVE-SHAFT MODEL] in format 9
 score = 0.7

testing match #[Unit: COMPRESSOR-FAIL-TO-ENGAGE-ALARM MODEL] in format 5
 against match #[Unit: DRIVE-ADAPTER-HUB-SPLINE MODEL] in format 10
 score = 0.25

testing match #[Unit: COMPRESSOR-FAIL-TO-ENGAGE-ALARM MODEL] in format 5
 against match #[Unit: DRIVE-SHAFT-SPLINE MODEL] in format 10
 score = 0.25757575

testing match #[Unit: OIL-PUMP-SHAFT MODEL] in format 9
 against match #[Unit: DRIVE-ADAPTER-HUB-SPLINE MODEL] in format 10
 score = 0.7

testing match #[Unit: OIL-PUMP-SHAFT MODEL] in format 9
 against match #[Unit: DRIVE-SHAFT-SPLINE MODEL] in format 10
 score = 0.75

testing match #[Unit: DRIVE-SHAFT MODEL] in format 9
 against match #[Unit: COUPLING-DRIVE-SHAFT-SPLINE MODEL] in format 10
 score = 0.7

testing match #[Unit: DRIVE-SHAFT MODEL] in format 9
 against match #[Unit: GEAR-SHAFT-SPLINE MODEL] in format 10
 score = 0.75

testing match #[Unit: DRIVE-SHAFT MODEL] in format 9
 against match #[Unit: WHEEL-SHAFT-SPLINE MODEL] in format 10
 score = 0.5833333

testing match #[Unit: DRIVE-SHAFT MODEL] in format 9
 against match #[Unit: DRIVE-ADAPTER-HUB-SPLINE MODEL] in format 10
 score = 1.0

testing match #[Unit: DRIVE-SHAFT MODEL] in format 9
 against match #[Unit: DRIVE-SHAFT-SPLINE MODEL] in format 10
 score = 1.5

testing match #[Unit: COUPLING-DRIVE-SHAFT MODEL] in format 9
 against match #[Unit: COUPLING-DRIVE-SHAFT-SPLINE MODEL] in format 10
 score = 1.5

testing match #[Unit: COUPLING-DRIVE-SHAFT MODEL] in format 9
 against match #[Unit: GEAR-SHAFT-SPLINE MODEL] in format 10
 score = 1.0

testing match #[Unit: COUPLING-DRIVE-SHAFT MODEL] in format 9
 against match #[Unit: WHEEL-SHAFT-SPLINE MODEL] in format 10
 score = 0.64285713

testing match #[Unit: COUPLING-DRIVE-SHAFT MODEL] in format 9
 against match #[Unit: DRIVE-ADAPTER-HUB-SPLINE MODEL] in format 10
 score = 0.64285713

testing match #[Unit: COUPLING-DRIVE-SHAFT MODEL] in format 9
 against match #[Unit: DRIVE-SHAFT-SPLINE MODEL] in format 10
 score = 0.6666667

The matches that have the highest scores overall are now selected as the most likely referents for each nominal. Also, any input/output or assembly/component connections that exist between the named objects are noted explicitly, to aid in causal inferencing later.

Running rule HIGHEST-SCORE in world *BACKGROUND*
 believe that the referent of (SPLINE) is DRIVE-SHAFT-SPLINE
 Adding (THE REFERENT OF FORMAT10-PART IS DRIVE-SHAFT-SPLINE)
 to *BACKGROUND*

Running rule HIGHEST-SCORE in world *BACKGROUND*
 believe that the referent of (DRIVE SHAFT) is DRIVE-SHAFT
 Adding (THE REFERENT OF FORMAT9-PART IS DRIVE-SHAFT) to *BACKGROUND*

Running rule SUPERCOMPONENT-PART in world *BACKGROUND*
 Adding (A SUPERCOMPONENT-PART OF FORMAT10-PART IS FORMAT9-PART)
 to *BACKGROUND*

Running rule HIGHEST-SCORE in world *BACKGROUND*
believe that the referent of (ALARM) is LUBE-OIL-PRESSURE-ALARM
Adding (THE REFERENT OF FORMAT5-ALARM IS LUBE-OIL-PRESSURE-ALARM)
to *BACKGROUND*

Running rule INPUT-FROM-PART in world *BACKGROUND*
Adding (AN INPUT-FROM-PART OF FORMAT5-ALARM IS FORMAT10-PART)
to *BACKGROUND*

Running rule INPUT-FROM-PART in world *BACKGROUND*
Adding (AN INPUT-FROM-PART OF FORMAT5-ALARM IS FORMAT9-PART)
to *BACKGROUND*

Running rule INPUT-FROM-PART in world *BACKGROUND*
Adding (AN INPUT-FROM-PART OF FORMAT5-ALARM IS FORMAT1-PART)
to *BACKGROUND*

From the preceding we notice that the system has decided that *alarm* refers to the *lube oil pressure alarm*, and that *drive shaft* refers to the particular drive shaft that carries output from the SSDG to the SAC (that shaft happens to be named “drive shaft” as well, but this is only one factor that the system takes into account in its disambiguation). It can then decide that *splines* refers particularly to the splines of this shaft.

The system now leaves the domain-specific (MODEL) knowledge base and enters the more domain-general (TERSE-RULES) knowledge base. Rule **catgz.damage** asserts that the STATUS *worn* is a DAMAGE word, and then asserts that the state of the PART *splines* is DAMAGED, since the STATUS and PART occurred in the same format line. Rule **damage.is.bad** then asserts that *worn* is an undesirable (BAD) state, and rule **bad.part.malfunc** asserts that pieces of equipment in an undesirable state are not functioning properly. Rule **catgz.failure** now classifies the status *drop* as a FAILURE word that in turn is also asserted to be an undesirable state.

Running rule CATGZ.DAMAGE in world *BACKGROUND*
Adding (A HEADCAT OF FORMAT10-STATUS IS DAMAGED) to *BACKGROUND*

Running rule DAMAGED.PART in world *BACKGROUND*
Adding (A STATE OF FORMAT10-PART IS DAMAGED) to *BACKGROUND*

Running rule DAMAGE.IS.BAD in world *BACKGROUND*
Adding (A HEADCAT OF FORMAT10-STATUS IS BAD) to *BACKGROUND*

Running rule BAD.PART.MALFUNC in world *BACKGROUND*
Adding (A STATE OF FORMAT10-PART IS MALFUNCTIONING) to *BACKGROUND*

Running rule CATGZ.FAILURE in world *BACKGROUND*
Adding (A HEADCAT OF FORMAT4-STATUS IS FAILURE) to *BACKGROUND*

Running rule FAILURE.IS.BAD in world *BACKGROUND*
Adding (A HEADCAT OF FORMAT4-STATUS IS BAD) to *BACKGROUND*

Now that it knows the referents for all pieces of equipment and the states of most of those pieces of equipment, the system can perform model-based causal and state inferencing. First, it notes

that *freely* was not classified as either BAD or normal, so it attempts to determine if the drive shaft might be malfunctioning by seeing if either a component of or a functional input to the shaft is malfunctioning. The drive shaft splines are malfunctioning because they are *worn*, so the system decides that *rotate freely* is probably used to describe a malfunction. The system also asserts that the worn splines were the *cause* of this malfunction. Since the oil pressure is known to be BAD and the drive shaft is what provides power to the oil pump, the system then determines that the malfunctioning shaft was the cause of the drop of pressure. Finally, the worn splines are asserted to be the root cause of the drop of pressure, by rule **causal-chain**.

Running rule INFER-BAD-SUPERCOMPONENT in world *BACKGROUND*

Adding (A HEADCAT OF FORMAT9-STATUS IS FAILURE) to *BACKGROUND*

Adding (A HEADCAT OF FORMAT9-STATUS IS BAD) to *BACKGROUND*

Adding (A STATE OF FORMAT9-PART IS MALFUNCTIONING) to *BACKGROUND*

Running rule INDIRECT-CAUSE-BAD-PROPERTY in world *BACKGROUND*

inferring that DRIVE SHAFT ROTATE FREELY caused LUBE OIL PRESSURE DROP

Running rule CAUSE-BAD-SUPERCOMPONENT in world *BACKGROUND*

inferring that WEAR PAST EXTENSIVELY SPLINE caused DRIVE SHAFT ROTATE FREELY

Running rule CAUSAL-CHAIN in world *BACKGROUND*

inferring that WEAR PAST EXTENSIVELY SPLINE caused LUBE OIL PRESSURE DROP

The system finally returns to the domain-general knowledge base, and is ready to assign scores to each salient concept (such as PART, CAUSE, FINDING, and BAD-STATUS) discovered in the message.

Running rule MENTION.PART in world *BACKGROUND*

;;; Add 1 point to FORMAT1 for SAC

Running rule MENTION.PART in world *BACKGROUND*

;;; Add 1 point to FORMAT9 for DRIVE SHAFT

Running rule MENTION.PART in world *BACKGROUND*

;;; Add 1 point to FORMAT10 for SPLINES

Running rule CAUSE in world *BACKGROUND*

;;; Add 2 points to SPLINES WORN because it caused ROTATE FREELY

Running rule CAUSE in world *BACKGROUND*

;;; Add 2 points to ROTATE FREELY because it caused PRESSURE DROPPING

Running rule CAUSE in world *BACKGROUND*

;;; Add 2 points to SPLINES WORN because it caused PRESSURE DROPPING

Running rule FIND in world *BACKGROUND*

;;; Add 1 point to SHAFT ROTATE FREELY because it was a FINDING

Running rule BAD-STATUS in world *BACKGROUND*

;;; Add 10 points to FORMAT4 for DROP

Running rule BAD-STATUS in world *BACKGROUND*
 ;;; Add 10 points to FORMAT9 for FREELY

Running rule BAD-STATUS in world *BACKGROUND*
 ;;; Add 10 points to FORMAT10 for WORN

Now the display shows *SPLINES EXTENSIVELY WORN* as the highlighted clause, having accumulated 15 points (it was the CAUSE of two events mentioned in the message, it mentions a PART, and it mentions a BAD-STATUS). A close runner-up was *DRIVE SHAFT ROTATE FREELY* with 14 points (it only was the CAUSE of one event, but it was FOUND to be true, and also mentions a PART). The following KEE queries show the final state of the system's working memory.

> (query '(a match of ?x is ?y))

;;; potential referents found in initial matching phase

((A MATCH OF FORMAT5-ALARM IS MATCH118)
 (A MATCH OF FORMAT5-ALARM IS MATCH117)
 (A MATCH OF FORMAT9-PART IS MATCH111)
 (A MATCH OF FORMAT9-PART IS MATCH110)
 (A MATCH OF FORMAT9-PART IS MATCH109)
 (A MATCH OF FORMAT10-PART IS MATCH116)
 (A MATCH OF FORMAT10-PART IS MATCH115)
 (A MATCH OF FORMAT10-PART IS MATCH114)
 (A MATCH OF FORMAT10-PART IS MATCH113)
 (A MATCH OF FORMAT10-PART IS MATCH112))

> (query '(the object of ?x is ?y))

;;; the equipment object associated with each matching

((AN OBJECT OF MATCH109 IS COUPLING-DRIVE-SHAFT)
 (AN OBJECT OF MATCH110 IS DRIVE-SHAFT)
 (AN OBJECT OF MATCH111 IS OIL-PUMP-SHAFT)
 (AN OBJECT OF MATCH112 IS DRIVE-SHAFT-SPLINE)
 (AN OBJECT OF MATCH113 IS DRIVE-ADAPTER-HUB-SPLINE)
 (AN OBJECT OF MATCH114 IS WHEEL-SHAFT-SPLINE)
 (AN OBJECT OF MATCH115 IS GEAR-SHAFT-SPLINE)
 (AN OBJECT OF MATCH116 IS COUPLING-DRIVE-SHAFT-SPLINE)
 (AN OBJECT OF MATCH117 IS COMPRESSOR-FAIL-TO-ENGAGE-ALARM)
 (AN OBJECT OF MATCH118 IS LUBE-OIL-PRESSURE-ALARM))

> (query '(the score of ?x is ?y))

;;; the score associated with each matching, and the winner (referent) for each

((THE SCORE OF MATCH118 IS 5.1928573) ;;; LUBE-OIL-PRESSURE-ALARM
 (THE SCORE OF MATCH117 IS 2.3837664)

 (THE SCORE OF MATCH116 IS 2.55)
 (THE SCORE OF MATCH115 IS 2.1)

(THE SCORE OF MATCH114 IS 1.5261905)
 (THE SCORE OF MATCH113 IS 3.5023808)
 (THE SCORE OF MATCH112 IS 3.857576) ;;; DRIVE-SHAFT-SPLINE

(THE SCORE OF MATCH111 IS 2.9373016)
 (THE SCORE OF MATCH110 IS 6.677778) ;;; DRIVE-SHAFT
 (THE SCORE OF MATCH109 IS 5.596825)

;;; the score earned by each information format

(A SCORE OF FORMAT1 IS 1) ;;; SAC RECEIVED HIGH USAGE
 (A SCORE OF CONN1 IS 0) ;;; DURING
 (A SCORE OF FORMAT2 IS 0) ;;; TWO BECCE PERIODS

(A SCORE OF FORMAT3 IS 0) ;;; CCS RECEIVED A REPORT
 (A SCORE OF CONN4 IS 0) ;;; THAT
 (A SCORE OF FORMAT4 IS 10) ;;; LO PRESSURE WAS DROPPING
 (A SCORE OF FORMAT5 IS 0) ;;; ALARM SOUNDED
 (A SCORE OF FORMAT6 IS 0) ;;; LOUD NOISES WERE COMING FROM
 THE DRIVE END

(A SCORE OF CONN8 IS 0) ;;; DURING
 (A SCORE OF FORMAT7 IS 0) ;;; COAST DOWN
 (A SCORE OF FORMAT8 IS 0) ;;; SOMEONE
 (A SCORE OF CONN11 IS 0) ;;; FOUND
 (A SCORE OF FORMAT9 IS 14) ;;; DRIVE SHAFT ROTATE FREELY
 (A SCORE OF FORMAT10 IS 15)) ;;; SPLINES EXTENSIVELY WORN

> (query '(the referent of ?x is ?y))

;;; the referent found for each PART/PROP/ALARM noun phrase

((THE REFERENT OF FORMAT4-PROP IS OIL-PRESSURE)
 (THE REFERENT OF FORMAT1-PART IS SHAFT-DRIVEN-AIR
 CENTRIFUGAL-COMPRESSOR)
 (THE REFERENT OF FORMAT4-PART IS LUBE-OIL)
 (THE REFERENT OF FORMAT9-PART IS DRIVE-SHAFT)
 (THE REFERENT OF FORMAT10-PART IS DRIVE-SHAFT-SPLINE)
 (THE REFERENT OF FORMAT5-ALARM IS LUBE-OIL-PRESSURE-ALARM))

IMPLEMENTATION

The SUMMARY system was implemented in OPS5, which provides forward-chaining production rules, control clauses on the left-hand sides of rules (which can be used to segregate rules into rule sets), and the ability to pattern match simple record-like structures. The first implementation of the starting air compressor equipment model was in the object-oriented language **Flavors** [11], that is integrated into the production rule system YAPS [12] that was being used for further development of the text highlighting system. **Flavors** provides all the basic mechanisms for implementing a frame-like representation, and so it was well suited to the initial development of a highly structured model. The most recent implementation of the text highlighting system is in Intellicorp's KEE Knowledge Engineering Environment. Although the YAPS environment provides both production rules and a frame-like language (**Flavors**), the YAPS discrimination net (the pattern matcher for rule conditions)

is not sensitive to changes to individual fields of **Flavors** objects, so the two representations are not fully integrated for rule-based behavior. KEE rules are sensitive to changes in the fields of structured objects (units), and have an English-like syntax that gives them good readability.

An example of a KEE production rule is shown in Fig. 12. Semantic pattern information is implicitly represented in the left-hand sides of rules. For example, the subpattern PART STATUS can occur in either a **repair** pattern (*repaired worn assembly*) or a **part-state** pattern (*splines extensively worn*). Thus the example rule will search for all **part-state** predications in which the STATUS is bad, and assert that the co-occurring PART is MALFUNCTIONING. The first three clauses of the IF condition match the general semantic pattern, while the fourth clause is a test for a more specific piece of information (HEADCAT) that was generated by some other piece of knowledge.

```
(BAD.PART.MALFUNC
 (IF (THE PART OF ?FORMAT IS ?PART)
      (THE STATUS OF ?FORMAT IS ?STATUS)
      (CANT.FIND (THE REPAIR OF ?FORMAT IS ?REPAIR))
      (A HEADCAT OF ?STATUS IS BAD)
      THEN
      (THE STATE OF ?PART IS MALFUNCTIONING)))
```

Fig. 12 — Rule **bad.part.malfunc**

KEE provides powerful frame-based mechanisms, a variety of rule-based control strategies, a convenient user interface, and graphics capabilities. KEE also provides the ability to maintain several knowledge bases at one time. This allowed us to give the system a more modular and manageable organization. YAPS was available only in Franz Lisp, but KEE is based in Common Lisp. TERSE runs on both a Symbolics and a Sun workstation. All the accompanying code for the system is written in Common Lisp.

CONCLUDING REMARKS

We have not yet carried out empirical tests to determine to what extent the inclusion of the additional domain-specific knowledge improves the fit between the machine-generated and manual text extracts. In some cases, the ability to infer additional causal information seems to be the factor that brings the machine extract closer to the human extract. The sample run of the previous section shows that the inclusion of domain-specific inferencing brought the score of the sentence *drive shaft was found to rotate freely at the SSDG end* up to within one point of the previous winner *splines were extensively worn*, thus giving us an extract virtually identical to the human-generated one. In other cases, domain-specific expertise does not seem to have any bearing on the variance between the human and machine extracts, therefore some other value judgments must be involved.

We presently judge TERSE's success by comparing its output to the output of the human personnel trained for the extraction task. This only allows us to measure success by the amount of overlap between the machine and manual extracts. Having these experts evaluate the system's output would provide interesting and valuable feedback in determining how close TERSE's misses are, and why the experts chose the alternatives they did.

Nominal Dereferencing Limitations

Although the techniques described in the section on Nominal Dereferencing allow most equipment names to be successfully dereferenced, several unresolved problems still must be dealt with.

Head Nouns

Certain generic head nouns such as *assembly*, *module*, and *coupling* are used liberally by message writers to describe various parts or collections of parts but have proved problematic in dereferencing against the model. Many pieces of equipment are named *assembly* in the equipment manuals, and their names generally fall into three categories based on the role of their left modifiers: equipment class (*oil pump assembly*), functionality (*high speed rotating assembly*), and component (*impeller assembly*). Those in the first group can be referred to simply by their model equipment class (e.g. *oil pump*). Those in the remaining two groups are represented in the model as being of class **<assembly>**, since that head noun is required to reference them. Message writers, however, often use *assembly* to invent a name for an unspecified collection of parts that has no distinct identity in the equipment manuals, such as *spline assembly* (a collection of things having splines) and *pump drive assembly* (a collection of things that drive the pump). Such references can often be ambiguous: *shaft assembly* may refer to a shaft or to some collection of related parts that includes a shaft. (Note that a piece of equipment need not have components to be called an assembly in the equipment manuals, such as the **<ring-gear-hub-assembly>**.)

Our present method of dealing with this problem is to assume that *shaft assembly* refers only to the shaft itself. This may not be precisely what the writer meant, but does return a pointer to a piece of equipment prominently related to the reference. This is probably the best that can be hoped for, since such nominals are confusing and ambiguous to human readers as well.

Left Modifiers

Several left modifier roles are problematic. Although we have developed ideas on how to handle these, the system currently must ignore them.

Four distinct starting air compressor systems are aboard ship, but we have chosen to instantiate only a single one and consider it a generic model for all four. Thus the matcher ignores the left modifiers in expressions like *nr 4 SAC*. This decision was based on earlier test data, in which only one system at a time was ever mentioned within a single message. In later data, however, we have seen references to more than one system within a single message, so a mechanism for distinguishing between the different systems is needed. This will probably not require the duplication of units in the equipment model, but just the flagging of message PART entries with an integer ID.

The equipment model does not yet incorporate spatial relationships, and so the matcher currently ignores locative left modifiers, as in *end flange* and *front seal*. Message writers tend to use locatives to identify low-level components whose precise identity is not as crucial as that of higher level components. The inclusion of spatial relationships in the model is helpful, however, for the interpretation of locative prepositional phrases, therefore the inclusion of spatial relationships is being considered.

Currently the system cannot handle verbal modifiers that *explicitly* signal functional or connective relations to other pieces of equipment:

- *mating engine mounted* drive hub
- solenoid *operated* clutch valve
- *driven* gear

In all these cases, to ignore the verbal modifier causes no dereferencing problems because the implicit functional/connective relations are still understood. *Engine drive hub* can be interpreted as a hub connected to an engine, *solenoid clutch valve* can be interpreted as a valve containing a solenoid component, and all gears can be considered to be *driven* by something. Still, the explicit verb provides a

modifier role cue that could be used. An extension to the pattern-matching BNF can probably handle these verbs by treating the verb as a predicate and any enclosing items as arguments [13].

Another problem in the matching of equipment names against the model arises from our approach to information formatting. In extracting classes of information from the message, the formatting component frequently decomposes nominal expressions into several distinct format entries. This is often desirable; it is convenient for *impellor blade tip erosion* and *tip of impellor blade eroded* to be formatted as PART AREA STATUS, because this gives the model a distinct part name (*impellor blade*) to match, and because it separates the predicative status word. When equipment names are further decomposed, however, we get ambiguities and erroneous semantic patterns. For example, *Unit has low output air pressure* and *failure of SAC lube oil pressure transducer* both decompose into the pattern PART PROP STATUS that should be interpreted semantically as “the PROP of the PART is in state STATUS.” However, only the first sentence conforms to these semantics. Similarly, the noun phrase *low lube oil alarm set point* is decomposed into the pattern STATUS PART ALARM PROP. This makes it appear that something (either a PART, an ALARM, or a PROP) is being predicated as having a STATUS, whereas the STATUS is being used to name something and not to predicate it.

In our present approach, we do not allow partial matching. Once the left modifiers that the system knows can ignore have been stripped off, each of the remaining words of the nominal must correspond to an attribute of some model object for a match to be successful. Since we already use uncertainty factors in the disambiguation of underspecified nominals, as described in the next section, partial matching could easily be incorporated. However, the problem of deciding how close a partial match must be to be considered acceptable is a difficult one. Partial matching could make the system somewhat more robust, but it might also introduce nominal dereferencing errors, and so requires further thought.

Text Representation Limitations

The initial distributional analysis of equipment failure messages determined that there were eight semantic patterns in the sublanguage. Each pattern corresponds to a different *primitive assertion type*, by which we mean a predication (either nominal or verbal) that contains no further embedded predicate material. The six primitive assertion types relevant to air compressor messages were **repair** (*SAC was removed*), **part-state** (*power pack failed*), **investigation** (*inspection of lube oil filter*), **general-administration** (*matting to be procured locally*), **assistance** (*assistance from Desron 8*), and **ship-task** (*awaiting parts for nr 2 SAC*).

We defined a new **format** structure as a general data structure that could encode any of these primitive assertion types. The format structure has one slot for each semantic category in the sublanguage. Depending on which of the structure’s slots are filled in, one can determine which semantic pattern to use for interpreting the format. For example, *Splines were sheared* fills the PART and STATUS slots of the structure, indicating the **part-state** pattern; *Replaced damaged hub* fills the PART, STATUS and REPAIR slots, indicating the **repair** pattern. To represent compound sentences, a binary operator called a **connective** conjoins two formats. Multiple conjoinings are accomplished by allowing either or both of the arguments to the connective to be a conjoined structure as well.

Many verbs in the sublanguage can take other predicates as arguments, such as *cause*, *believe*, and *reveal*. Since format structures cannot contain embedded predicate material, we decided that these verbs be represented as a special type of connective, because that was the only available structure that could have other predicates as arguments. For some verbs, such as *cause* and *reveal*, both

arguments to the verb are often predicative in nature. For verbs like *believe* or *find*, however, only the second argument can be sentential. Since the connective structure can only conjoin predicates, the subject of *believe* must be represented as a fragmentary assertion, which is erroneous. The resulting representation scheme is also inconsistent: although the high-order verbs are represented in a kind of binary predicate-argument form, first-order verbs (those occurring in the primitive assertion types) are not marked as predicates but simply co-occur with their arguments (also unmarked) as format elements.

Since the format structure only contains one slot for each semantic category, a primitive assertion such as *drive shaft sheared all internal gear teeth* cannot be represented as a single format at all, because two pieces of equipment are mentioned but only one PART slot is in the format. Hence the verb *sheared* in this sentence has to be treated as a connective, with *drive shaft* and *internal gear teeth* formatted as fragmentary assertions. When used either intransitively or passively, however (e.g., *the teeth sheared*, *the teeth were sheared*), *sheared* is formatted as a STATUS entry and the sentence is represented by a single information format.

For distributional co-occurrence patterns to work as a consistent text representation, there must be a pattern corresponding to every type of assertion that can be made in the sublanguage, whether primitive or higher order. There would have to be a **reveal** pattern, for example, of the form **investigation REVEAL part-state**, in which the items co-occurring with the REVEAL verb are subpatterns. To accommodate *shaft sheared all gear teeth*, there would have to be a pattern CAUSATIVE-PART DAMAGE AFFECTED-PART. The sentence *gear teeth sheared* (whether interpreted in the active or passive sense) would map to the same pattern, with *gear teeth* filling the AFFECTED-PART slot in either case, and the CAUSATIVE-PART slot left empty. As a result, *sheared* would always be formatted into the same semantic class and role no matter how it had occurred syntactically.

Because of the syntactic and semantic complexity of the domain, however, it is not altogether certain that the subject of the verb *shear* will always be a PART entry. One can readily imagine the sentence *Continued rotation of drive shaft sheared all teeth from hub*, where the subject of *shear* is the nominalized predicate *continued rotation of drive shaft*. This would require that we generalize the "CAUSATIVE-PART" slot above to be simply CAUSATIVE, and that we place a type constraint on *shear* requiring that the item filling the CAUSATIVE slot be either of the semantic class PART, or of another predicate form. In turn, the possibility of the sentence *Debris sheared all teeth from hub* suggests that we create a new semantic category PHYSICAL-THING of which PART and MATERIAL (debris) are both subclasses, and loosen the PART class constraint to be PHYSICAL-THING.

These modifications are clearly taking us in the direction of *case-frame representation* [14], in which co-occurrence patterns are replaced by *predicate classes*. The slots of these predicates represent true *roles*, not just semantic classes; the semantic classes (reorganized into a hierarchy) serve as type constraints on the values each slot can hold. The **investigation** pattern ORG PART STATUS INVEST, for example, would be replaced by the predicate class **investigate**, which takes two arguments, an "agent" and an "object." The agent must be of semantic class ORG, and the object must be either of semantic class PART (*S/F inspected SAC*) or a nominalized predicate (*S/F investigated failure of SAC*). Note that this representation explicitly provides a certain amount of abstract role information (i.e., *predicate, agent, object*), represents all predicating elements consistently, and associates each entity with the predicate of which it is an argument. The co-occurrence pattern ORG PART STATUS INVEST does not indicate that the STATUS is a predicate on the PART, but simply that it is an optional co-occurring element. In the predicate-class representation of *S/F investigated failure of SAC*, however, *failure of SAC* is encoded as a separate (nominalized) predicate in its own right. The new parser that we have recently begun to use for syntactic analysis (see the next section) has predicate-argument structures as its default sentence normalization scheme, so moving to this new representation should be straightforward.

RECOMMENDATIONS

Transitioning to Fleet

The components of the present system are (1) the natural language parser LSP (written in Fortran) that runs only in batch mode, and (2) the TERSE software that runs in the interactive KEE software environment. To process a message from an input text file, first we run the LSP parser and formatting component on a VAX 11/750 machine. The information format output is written to text files. TERSE is then run separately on either a Symbolics or Sun workstation, and it reads these files as its input. The complete system is not fully integrated at this moment.

We are currently transitioning to the PROTEUS parser for our natural language analysis work. PROTEUS is written in Common Lisp, which allows it to run easily in the same machine environment as the TERSE system. PROTEUS has several other advantages over the LSP parser. It uses a faster and more efficient parsing algorithm, and it has a declarative normalization component that seems to be easier to program than LSP's procedural, tree-manipulating normalization routines. Also, PROTEUS encourages the use of a more consistent text normalization scheme than information formats, as discussed in the previous section. We recommend that the additional work necessary to bring PROTEUS into full use be done, so that it can serve as the natural language analysis software for TERSE. This work will be done in three phases. First, many grammatical restrictions in the LSP grammar must be rewritten to accommodate the PROTEUS tree construction methodology. Second, a selectional component (semantic constraints for selecting from multiple syntactic analyses) must be built. Finally, the TERSE system rulebases must be modified somewhat to accept predicate-argument forms instead of information formats as input.

Although the inferencing rules used in the equipment model were designed to apply to any other mechanical domain, the model itself (that portion of the knowledge base containing the component definitions) must be updated for each new CASREP equipment domain to be processed. Even as small a piece of equipment as the Starting Air Compressor required the modeling of over 80 components, having hundreds of interconnections and attributes, to enable nominal dereferencing and causal inferencing. As was discussed in the introduction to the section "THE EQUIPMENT MODEL," common-sense knowledge about equipment in general does not seem to provide enough expertise to perform any useful inferencing about equipment failure messages. Rather, detailed knowledge about the structure and configuration of the particular piece of equipment seems to be necessary. The task of constructing a domain model for the thousands of pieces of equipment on a Navy ship would be extensive and costly. Further progress must be made to solve the *knowledge acquisition bottleneck* problem to make this feasible.

Moving to New Message Domains

In general, what the TERSE rule system does is search a message for sentences of the form *We found that X*, *Investigation revealed that X*, *X caused Y*, *We suspect that X*, *The problem is X*, and *(Part) X is (in state) Y*, and attach arbitrary scores to each. We have not yet abstracted *why* these particular "sentence templates" should signal important information, or what the basis of their importance relative to each other is. For example, the goal of ship personnel is that all equipment work properly, and equipment failure violates or challenges that goal. To correct the goal's violation, the equipment must be repaired or replaced. This requires that investigations be made to determine which equipment has failed and what caused it. By abstracting a "script" representing the hierarchy of human goals and subgoals, we are able not only to provide psychological justification for these rules, but recast them in a form that could be applied to other message domains. At present, the system requires that the sublanguage style of the new message domain use the same types of explicit

phrasal markers. In a new domain, such as tactical messages, the same types of information might occur — symptoms of a possible problem are detected (receiving intelligence reports), investigations are made (dropping sonobuoys) that reveal the exact nature of the problem (enemy platforms are detected), and actions are taken to correct the bad conditions (firing across the enemy's bow) — but without any of the CASREP-sublanguage phrasal markers listed above being used. With a more abstract goal/subgoal script, it might be possible to *infer* the contribution of each message sentence to the goal hierarchy, in effect finding the abstract “story” behind the message.

Reevaluating the Task

In the prior discussion we assumed that a Text Reduction System such as TERSE provides a facility useful to the Navy. In this section we examine the topic of Navy message processing in more detail and propose some alternative approaches.

The Navy is faced with several problems in using messages such as CASREPs. One is the sheer volume of messages coming in on a daily basis; another is the need to sort out the time-critical from the non-time-critical information. Some of these tasks can be handled by using simple human or computerized approaches, especially when the important data are contained in the formatted portions of the reports. However, some of the important information is contained in sections of the message that are written in free text. This information is much more difficult to process by machine than the information contained in the formatted sections.

The approach currently taken by NAVSEA is for contracted personnel to read the narrative portions of the messages and to extract up to 60 characters of the *most important* information from the paragraph. Their goal is to reduce the text to a manageable size while still retaining information important to perform failure trend analysis. After this reduction is made, the extracts can be entered as raw texts into a computer database, or collected into a report. This information is then used by other personnel to do failure trend analysis. The goal of the TERSE system was to accomplish the same task as the NAVSEA personnel, that is, the generation of short extracts containing equipment, state, and causal information suitable for failure trend analysis.

We feel that there were restrictions that caused NAVSEA to approach the problem in the manner they did, and that these restrictions no longer apply to our automated message processing approach. NAVSEA's approach was to extract a 60 character text from the narrative, and discard the remainder. When this operation was implemented, computer technology was only in its *second generation*. This generation was characterized by limited and expensive primary storage (main memory) and limited, expensive, and slow secondary storage, compared to what is available today. Database retrieval systems were inefficient at handling the variable-length records that would be required to hold the paragraph information in these reports. Even if the text could have been formatted into these databases, no means existed for retrieving the contents, because these fields could not be “keyed” like traditional fields in a database record. The system would have had to resort to exhaustive character-by-character examinations of all text records contained in the database just to retrieve a record containing a particular word. All in all, technology in both software and hardware made it prohibitive to do anything useful on the computer with this portion of the messages. It seems that NAVSEA's answer to this problem was to reduce the text to a more manageable and predictable size. However, the problems of accessing the reduced natural language text in these traditional database systems still remained, and we doubt that any automated procedure uses these fields at all.

The alternative to this approach is the kind of natural language technology we are now exploring, which drastically changes what is possible in automated message processing. Rather than duplicate a process that had its origins in an earlier computer technology, with its inability to process

natural language inputs, it now seems more worthwhile to attempt to automate some of the actual end-user information processing tasks, such as failure trend analysis and ship readiness assessment. Rather than take a text, discard some of its information, and enter the remainder of the information (still in text form) into a database to be used for failure trend analysis by human personnel, we recommend that all the message information be retained in machine-accessible form (such as the information format database records employed by TERSE). Several automated end-user applications (including failure trend analysis) can then make use of this database, without the unnecessary intermediate stage of text reduction. Although text reduction is a useful application, it must be given an appropriate target domain. The text in the CASREP reports tends to be highly condensed and telegraphic, thus suggesting that the writers have already attempted to be as concise as possible and include only important information.

ACKNOWLEDGMENTS

The Office of Naval Technology under A.I. Project RS34C74 and the Office of Naval Research under Natural Language Research project RR015-08-01 supported this research. We also thank Dennis Perzanowski for his work in parsing the message narratives and for reviewing this report.

REFERENCES

1. E. Marsh, H. Hamburger, and R. Grishman, "A Production Rule System for Message Summarization," Proc. of the AAI, Austin, TX, Aug. 6-10, 1984, pp. 243-246.
2. E. Marsh, J. Froscher, R. Grishman, H. Hamburger, and J. Bachenko, "Automatic Processing of Navy Message Narrative," NRL Report 8893, Aug. 1985.
3. C.L. Forgy, "OPS5 User's Manual," Technical Report CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, July 1981.
4. N. Sager, *Natural Language Information Processing: A Computer Grammar of English and its Applications* (Addison-Wesley, Reading, MA, 1981).
5. E. Marsh, "Transporting the Linguistic String Project System from a Medical to a Navy Domain," *ACM Trans. Office Inf. Syst.* 3(2), 1985.
6. Z.S. Harris, "Distributional Structure," *Word* 10(2-3) (1954).
7. E. Marsh, "A Computational Analysis of Complex Noun Phrases in Navy Messages," COLING 84, July 2-6, 1984, pp. 505-508.
8. R. Grishman, "PROTEUS Parser Reference Manual," PROTEUS Project Memorandum #4, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, July 1986.
9. D.A. Dahl, "Focusing and Reference Resolution in PUNDIT," Proc. AAI, Philadelphia, PA, Aug. 11-15, 1986, pp. 1083-1088.
10. G. Hirst, *Semantic Interpretation and the Resolution of Ambiguity* (Cambridge University Press, Great Britain, 1987).
11. E.M. Allen, R.H. Trigg, and R.J. Wood, "The Maryland Artificial Intelligence Group Franz Lisp Environment," Technical Report TR-1226, Department of Computer Science, University of Maryland, Dec. 1984.

12. E.M. Allen, "YAPS: Yet Another Production System," Technical Report TR-1146, Department of Computer Science, University of Maryland, Feb. 1982.
13. T. Ksiezyk, R. Grishman, and J. Sterling, "An Equipment Model and its Role in the Interpretation of Noun Phrases", Proc. Tenth Int. Joint Conf. on Artificial Intelligence, Milano, Italy, 1987.
14. C.J. Fillmore, "The Case for Case," in *Universals in Linguistic Theory*, E. Bach and R.T. Harms, eds. (Holt Rinehart and Winston, New York, 1968), pp. 1-88.

