Inferring Heuristic Classification Hierarchies from Natural Language Input*

Richard Hull, Fernando Gomez
Department of Computer Science
University of Central Florida
Orlando, FL 32816
hull@cs.ucf.edu (407) 823-2366

Abstract

A methodology for inferring hierarchies representing heuristic knowledge about the check out, control, and monitoring sub-system (CCMS) of the space shuttle launch processing system from natural language input is explained. Our method identifies failures explicitly and implicitly described in natural language by domain experts and uses those descriptions to recommend classifications for inclusion in the experts' heuristic hierarchies.

1 Introduction

It is becoming generally accepted that most experts organize their problem-solving knowledge into a hierarchy of concepts [Gomez and Chandrasekaran, 1984; Clancey, 1985]. This hierarchical organization of knowledge is not explicitly used by the experts during the solution of problems, but rather is used in an implicit form. The task of the knowledge acquisition programs is to extract this hierarchical organization from the experts by making explicit to them the steps they need to visit in arriving to solutions. In other words, the goal of the knowledge acquisition interface is to make explicit the hierarchy of concepts. A well known knowledge acquisition methodology to acquire hierarchical knowledge from experts is that of repertory grids [Boose and Bradshaw, 1988; Boose, et al., 1989; Gaines and Shaw, 1988]. The repertory grid methodology elicits categorizations, called *constructs*, from the expert by asking him/her to rank numerically elements of the domain according to how well they satisfy a given construct.

Although this methodology has achieved considerable success, the problem of construct selection remains one of the most serious bottlenecks in the repertory grid methodology. If the constructs are provided to the domain expert by the knowledge engineer, the method works reasonably well because the task of the domain expert consists of filling in the cells of the grid with the appropriate values. However, in most cases the key aspect of the knowledge acquisition task is the acquisition of the constructs themselves from the domain expert. In this regard, elicitation techniques face strong limitations due to the fact that the linguistic aspect and contextual knowledge associated with the constructs are difficult to handle by elicitation techniques alone.

Our own research has been addressing this problem by studying the automatic construction of *constructs* or categorizations from natural language input. In [Gomez and Segami, 1991], the reader may find a description of linguistic constructions whose underlying structures are hierarchical categorizations. In this paper, however, we study the problem of *inferring* classifications from natural language sen-

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

tences, rather than that of directly mapping into hierarchical structures. In order to provide some motivation for the problem we are facing, Figure 1 contains a portion of the heuristic hierarchy acquired from domain experts using our present interface. The problem we have experienced with our present interface is similar to the acquisition of *constructs* in the repertory grid methodology. If a good portion of the heuristic hierarchy is provided to the domain expert by the knowledge engineer, he/she can continue from there without considerable difficulty. However, building the hierarchy from scratch by the domain expert is a different matter altogether. Then, the main idea is to ask the expert to describe a given problem (a CCMS computer error in our application), infer some categorizations from the natural language description, and ask the expert to select the relevant one(s). This is basically the main idea that we explore in this paper in the context of the CCMS space shuttle network.

The remainder of this paper is organized into 6 sections. Section 2 describes the problem domain and our original knowledge acquisition interface. Section 3 describes the relationship between the interface, the Natural Language Component (NLC), and the Classification Suggestion Module (CSM). Section 4 explains the structures passed from the NLC to the CSM. Section 5 describes how the CSM infers classifications. Section 6 provides an overview of the NLC. Section 7 gives the authors' conclusions and lists future work to be done.

2 Automatic Knowledge Acquisition Interface (AKAI)

OPERA (Expert System Analyst) is an expert system whose task is to improve the operations support of the computer network in the space shuttle launch processing system at Kennedy Space Center[Adler, et al., 1989]. OPERA functions as a consultant to systems engineers by suggesting probable causes and recommending diagnostic and operational advisories regarding network error messages generated by the check out, control, and monitor subsystem (CCMS). Because OPERA only has information on approximately 10% of the 1500 error messages generated by the CCMS network, some type of knowledge acquisition tool is needed. During the past several years we have worked to develop a knowledge acquisition interface for OPERA. The result of this effort has been the creation of the Automatic Knowledge Acquisition Interface or simply AKAI.

It became apparent to us as we worked on the interface that while OPERA is not based on classification problem-solving, AKAI could make use of classification hierarchies [Gomez, et al., 1992a]. Two distinct types of classification hierarchies were identified and are now used by the interface: heuristic hierarchies and factual hierarchies. Heuristic hierarchies represent heuristic problem-solving knowledge of the domain. Each expert has his/her own ideas about how this knowledge is organized depending on their personal experience. Factual hierarchies represent hard or factual knowledge about the physical structure of physical objects. A factual hierarchy for the CCMS network was constructed and is currently being used by the interface. Because of the static nature of the CCMS network, the factual hierarchy is rarely modified. Of primary concern to us is the acquisition of the heuristic knowledge possessed by CCMS experts. Therefore, the focus of our research now is acquiring and constructing heuristic hierarchies, with the goal of AKAI being to acquire probable causes and advisories from systems engineers as efficiently as possible.

Towards this goal, user friendly features such as pull-down menus, mouse selectable text, and a wealth of functions to reorganize the hierarchy were incorporated in AKAI. Beta testing revealed, however, that naive users still had difficulty during the initial stages of heuristic hierarchy construction for the reasons stated above. In an effort to address this problem,

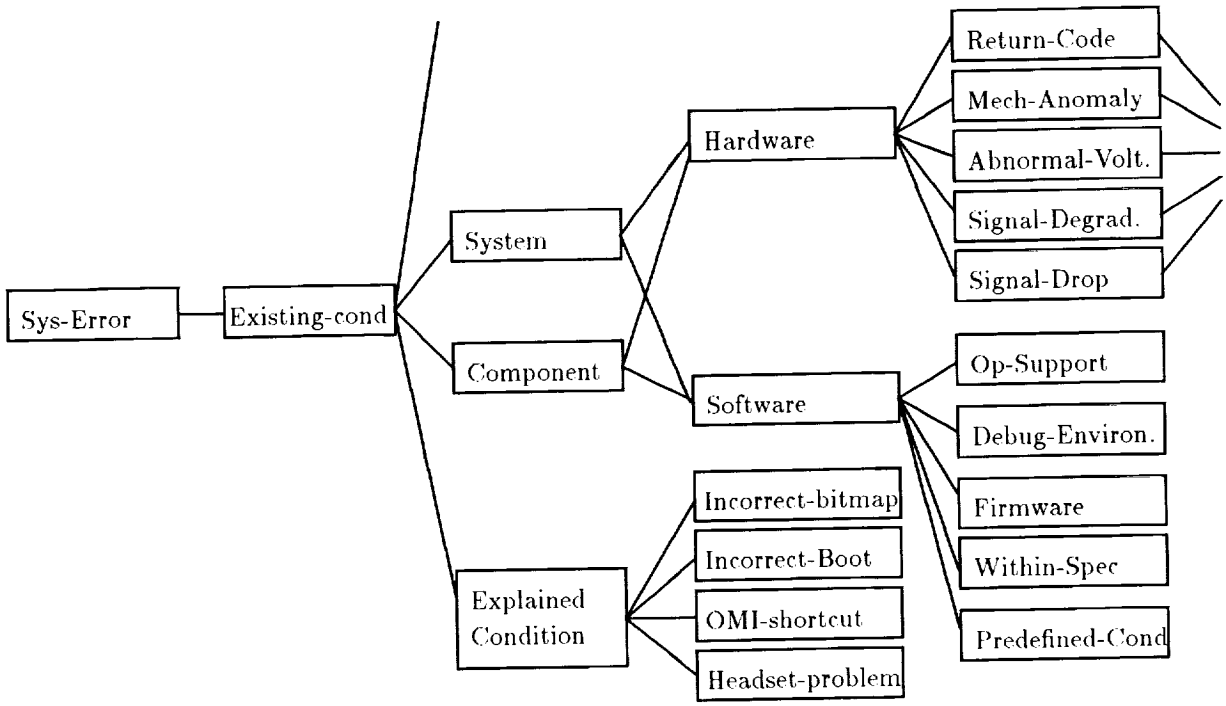

Figure 1: A Portion of a Heuristic Hierarchy for the CCMS Domain

we have added a natural language component (NLC) and a classification suggestion module (CSM).

3 The Improved Knowledge Acquisition Interface

The operation of the interface, graphically displayed in Figure 2, has changed only slightly due to the addition of the NLC and the CSM. The NLC is constructed around SNOWY [Gomez & Segami, 1989, 1990, 1991]. SNOWY is responsible for parsing (determining the syntactic constituents of the sentence) and interpreting (constructing the logical form of the sentence), and then forming (mapping the logical form of the sentence into SNOWY's representation language). The NLC is called by the interface during error categorization. At this time, the expert is asked to place the error message he/she has chosen to describe in his/her heuristic hierarchy. During the first stages of hierarchy construction there is a good chance that the appropriate category for the error message currently being described is not already in the heuristic hierarchy. In the original interface, the expert was expected to know, and was asked for, the name of an appropriate category. This was often a problem in the initial stages, and the experts caught in these situations tended to provide unsound categorizations.

The interface has since been enhanced to help unsophisticated users add new error categories to their heuristic hierarchies. If a user is unsure of how to classify an error, he/she is asked to provide a short description of what he/she knows about the error. This description typically consists of two or three sentences detailing relevant information about the message. The text is saved and passed to the NLC. The NLC enlists SNOWY to parse, interpret, and form the sentences. If SNOWY can make sense of the expert's description, the output of the formation phase is then passed to the CSM. The CSM uses the formation output to recommend categories to the expert. If one or more of these recommendations are selected by the

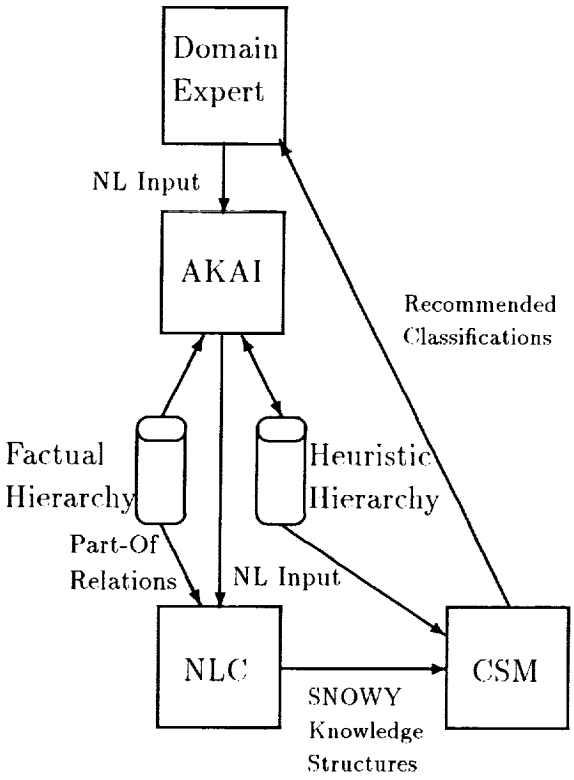

Figure 2: System Design

expert as acceptable, the problem of classifying the error is solved and the suggested error categories as well as the error message are placed in the expert's heuristic hierarchy. The interface then prompts the user for the probable causes of the error message and its operational and diagnostic advisories (this function of AKAI was not changed by the addition of the NLC and CSM).

On the other hand, if the expert is not satisfied with the CSM's recommendations or if SNOWY is unable to understand the expert's description, we may still be able to make a reasonable suggestion by postponing the classification of the error message until the probable causes have been entered by the domain expert and examined by the interface. We strongly believe that the probable causes represent an excellent source of text that is understandable by SNOWY and will provide classifications worth recommending. Most of the probable cause data that has been collected so far is of the form "$\mathcal{X}$ has failed," where $\mathcal{X}$ is a component of the CCMS network. SNOWY is quite capable of understanding sentences in this form. The classifications suggested by the CSM for these sentences are usually relevant because experts commonly use failed component names as category names within their heuristic hierarchies. If this process fails, however, the NLC and CSM are deactivated and the user falls back on the features of the original interface.

One may then question why the interface bothers to ask the user for a textual description of the error when analysis of the probable causes appears to provide suitable suggestions. We have found that additional text is needed if we are to make suggestions other than failed component suggestions. The system would not be able to make suggestions like "initialization failures" or "on line failures" if we only called the NLC with probable cause text. Classifications of this type are present in the heuristic hierarchies of the Grumman personnel first consulted to test the interface. Therefore, we must provide the interface with additional texts which could lead to recommendations other than failed components.

The operation of the enhanced interface is identical to the original after the error message has been placed within the heuristic hierarchy. The code of the original interface, therefore, was disturbed only slightly, and the users of the interface did not need to re-learn how to operate the system.

4 Input to the CSM

Before addressing the details of the CSM we must describe the structures which it takes as input. The formation phase of the NLC maps the logical form constructed by the interpreter into the knowledge representation structures of the representation language KL-SNOWY through the use of formation rules. The formation algorithm is called to form clauses as they finish the interpretation phase. The most

embedded clause of a sentence is formed first, the second most embedded is formed second, and so on until the main clause is formed. The structures, called object structures and relation structures, are used by the CSM to make recommendations. Together these two types of structures form the kernel of KL-SNOWY. There is a significant advantage to having SNOWY apply its formation phase to the logical form produced by the interpreter. This will become apparent during our discussion of the Classification Suggestion Module in section 5, if one understands the structures of the representation language. Therefore, it is important that the semantics of object and relation structures is clear.

4.1 Object Structures

Object structures represent knowledge about physical and abstract objects. Some physical objects are trains, tools, mountains, geese, etc., and some abstract objects or ideas are sets, states, properties, and relations. Conceptual relations representing knowledge about the object are represented as slots in the object structure frame (see the box surrounding the object structure for *CPU* in Figure 3).

These relations will either describe the object in some way or attribute actions to it. In the CPU object structure example, the slot "(process (data ($more (@a3))))" represents an action attributed to the concept CPU, and "(made-of (silicon ($more (@a2))))" represents a description. The relation structure names, *@a3* and *@a2*, point to relation structures that contain additional information which is not stored directly under CPU but elsewhere in SNOWY's long-term memory (LTM). In general, concept relations are represented in object structures as:

```
relation (@a1)                  monadic
relation (concept1 (@a1))       diadic
```

All concepts must have a unique name in memory so that the knowledge about them can

Object Structure

CPU
(is-a (electrical-component))
(part-of (computer ($more (@a1))))
(made-of (silicon ($more (@a2))))
(process (data ($more (@a3))))

@a3
(instance-of (action))
(args (CPU) (data))
(pr (process))
(actor (CPU (q (?))))
(theme (data (q (?))))

Figure 3: A Portion of the Concept *CPU* Acquired by SNOWY from Natural Language Input.

be integrated in a single place. Therefore, we need a method for dealing with concepts which are not explicitly named in the sentence. An example of such a sentence is "The adapter in the FEP returned an invalid status." The subject of the sentence, "the adapter in the FEP," is a complex concept which must be given a dummy name (a gensym) to uniquely identify it in LTM. The structure is called an *x-structure*. We use a *characteristic-features* slot to specify the necessary and sufficient conditions describing this new concept. For this complex concept, the representation would be:

```
(x1 (cf (is-a (adapter))
        (part-of (FEP))))
```

The meaning of this is that the x-structure *x1* is a sub-class of adapter, whose members all have the feature of being a part of a front-end processor (FEP). This feature is "characteristic" because it is shared by every member of the class *x1*. Complex concepts can arise from natural language constructs such as existentially quantified sentences, complex noun phrases, and restrictive qualifiers (relative clauses and prepositional phrases).

4.2 Relation Structures

Relation structures represent knowledge about instances of conceptual relations. Each structure contains a verbal concept, its cases and their fillers, the quantification of each filler, an *instance-of* slot indicating whether the relation is a description, action, proposition (embedded relation) or cf-structure, and an optional *truth-value* slot which indicates whether the relation is believed to be true or false by SNOWY. In the absence of a *truth-value* slot the statement is taken as true by default. For example, the relation structure, *@a3*, that represents "CPUs process data" is shown at the bottom of Figure 3.

The first slot, *instance-of*, indicates that *@a3* is an instance of an action relation. The *args* slot lists the arguments of the relation. If the relation is monadic, the *args* slot will contain a single concept. If the relation is diadic, as is the case in this example, the *args* slot contains two concepts, and so on. The *pr* slot contains the verbal concept or primitive. Following the verbal concept are its thematic cases. Each case is filled by a "quantified" concept from the argument list. The quantifier of an argument is the filler of its *q* sub-slot. In *@a3*, both the quantifiers for CPU and data are unknown, represented by a question mark. This reflects the fact that from the statement "CPUs process data" it is not clear if all CPUs process all data or only some CPUs process some data. Other possible fillers of the *q* slot are: most, many, all, cardinal adjectives, and numerals.

Creation of relation structures is normally handled by the formation algorithm. This algorithm constructs structures from the logical form by collecting the thematic cases identified by the interpreter for sentence clauses. In certain sentences, however, the formation algorithm must be overridden or postponed because the verbal concept requires an unusual construction to be formed. To handle these special cases, we use formation rules which are briefly discussed in section 6.

5 The Classification Suggestion Module (CSM)

The task of the Classification Suggestion Module (CSM) is to take the output from the formation phase of SNOWY and produce a list of error message classifications that can be suggested to the user. To accomplish this task, the CSM scans the output of the formation phase of SNOWY looking for certain constructions that are likely to lead to plausible suggestions. The CSM looks for the following constructions: negated relations and relations that indicate failures, descriptive relations which explicitly or implicitly indicate failed components, and complex noun phrases describing failed components. After a set of suggestions is identified, the CSM attempts to prioritize them based upon an analysis of the expert's heuristic hierarchy. This prioritized list of suggestions is then presented to the expert. Additionally, if the expert selects one or more of the suggestions, the CSM will attempt to engage the expert in a dialog whose purpose is to elicit more information. The sections below discuss each of the constructions relevant in identifying possible suggestions, the prioritization task, and the elicitation of additional information.

5.1 Relation Structures

The CSM identifies relation structures containing negated verbal concepts or with verbal concepts that indicate failures. Consider for example the formation of the sentence "The FEP failed to detect an acknowledgement from the i/o adapter," which contains a negated verbal concept.

The CSM scans the formation output for relation structures, such as @a27 below, and examines their truth-value slots. If the truth-value slot indicates that a verbal concept is explicitly negated, as **become-aware** is in the example below, we save the relation structure. The system can then use the cases of these structures to generate plausible classification suggestions (see the following section).

```
@a27
   (truth-value (f))
   (args (fep) (acknowledgement))
   (pr (become-aware))
   (actor (fep (q (constant))))
   (theme
      (acknowledgement (q (?))))
   (instance-of (proposition))
```

Example 1

Verbal concepts that implicitly indicate failures are also identified. In the sentence, "The option plane microcode crashed," the verb *crashed* indicates a failure. This is immediately obvious to the CSM because of the verbal concept that *crash* is mapped to during formation.

```
option-plane-microcode
   (is-a (microcode))

@a30
   (args (option-plane-microcode))
   (pr (fail))
   (actor
      (option-plane-microcode
         (q (constant))))
```

Example 2

The verb rules for the verb *crash* map it to the verbal concept **fail**. Other verbs which are mapped to the verbal concept **fail** are *break*, *collapse*, and *fail*. Because SNOWY is able to determine the underlying meaning of these verbs, the CSM has an easy time selecting negated relations and relations indicating failures.

5.2 Case Roles as Plausible Classifications

Some of the cases of these relation structures, such as actor, theme, at-loc, and at-time, lead to plausible classifications. In Example 1 above, the relation structure @a27 has two case slots: the actor case, filled by *fep*, and the theme case, filled by *acknowledgment*. These two cases suggest two possible error message classifications. One possible classification is the class of error messages generated by "fep failures". Because all the relation structures selected by the CSM denote failures, the actor of each relation represents a component that has failed to accomplish some task.[1] That failed component may also be responsible for generating other error messages. Therefore, it makes sense to recommend a class of error messages caused by the failed component. For this example, the CSM would save the classification "fep failure" as a possible classification to be recommended to the expert.

Another possible classification is "acknowledgement failures".[2] This supports the notion that the theme case of failure relations may lead to plausible classifications, when the original sentence is a "fail to" construction. In the sentence "The common data buffer failed to update the system configuration table," the theme case, filled by "the system configuration table," may potentially represent a category of errors. While the actor case represents "what" failed, the theme case describes the component that failed to be acted upon. Consequently, one might think that the theme case is not as likely a source of classifications as the actor case. We can, however, conceptualize a class of error messages which indicates the failure of some component to update the system configuration table. Each member of the class would have similar operational advisories instructing systems engineers in how to handle the failed update. Therefore, the CSM saves the theme case fillers of negated relations as possible classifications.

At-time cases can also lead to plausible classifications. These cases indicate *when* a failure occurred, which may be very signif-

[1] We must recognize that if the expert describes failures of irrelevant components, the system will make necessarily irrelevant recommendations which the expert may ignore.

[2] These failures are so common they are referred to as NOACKs.

icant. For example, consider the sentence "The FEP failed to respond during initialization." The prepositional phrase "during initialization" tells us that the failure occurred during the process of initialization. In general, if the filler of the at-time case is a process, we recommend that filler as a possible classification. For the example above this gives "initialization failures". It is our belief that the fillers of the at-time case should almost always be processes. This is because it makes little sense to use a time NP (a noun phrase specifying a time) except in certain situations.[3]

At-loc cases can lead to plausible classifications. For example, "The transmitter/receiver failed in the HIM" is a sentence in which the at-loc case, filled by "the HIM," represents a possible category of error messages. Because the failure occurred within the HIM, we can infer that the transmitter/receiver is located within the HIM and therefore may be a sub-part of the HIM. The HIM, which is the larger object, is likely to have other sub-parts which may fail. This means that the class of "HIM failures" is likely to be a good category of error messages. One should note that the object and its sub-part(s) form a part-of hierarchy. Discussion of how part-of hierarchies can be used to help prioritize suggested classifications can be found in section 5.5, *Part-Of Hierarchies.*

[3]In most cases, we would not expect to see a sentence with an at-time case filled by a time NP, such as, "The FEP failed to respond to the HIM at 10 pm." Obviously the expert giving such a description does not realize that he/she has described a specific error event, while what we are after is a more general description of the error. However, it may make sense to write, "The FEP fails to respond to the HIM during the winter". This sentence would lead to the classification, "winter failures," which seems plausible. In the cases where the at-time filler is a time NP, the CSM asks the expert, "Is this the only time that this error occurs?" If the expert responds with an affirmative answer, the system retains the filler as a possible classification.

5.3 Descriptive Relations and Noun Phrases

Concepts that have negative properties may lead to plausible classifications. If the expert mentions a defective component within his/her error message description, that component is likely to contribute to the error. The CSM identifies descriptive relations that indicate faulty components, as in "the i/o adapter is not operational," "the HIM may be down," or, "the HIM is unable to reset the status register." In these cases, the predicate adjective is examined to see if a failure is present. Predicates that explicitly or implicitly describe negative properties of network components provide strong indications that the components they modify have failed. Explicitly negated predicates are those that clearly indicate a negation, either by inclusion of the adverbs *not* and *no*, or through the use of negative prefixes. Some examples of explicitly negated predicates are *abnormal*, *unable*, *disabled*, *uninhibited*, and *incapable*. Important features, such as negative prefixes, are stored in a lexicon for each word. For example, the word *abnormal* has the following feature:

```
abnormal
   (neg-prefix (normal))
```

The neg-prefix slot tells us that *abnormal* contains a negative prefix affixed to the root word *normal.*

The representation of descriptive relations that denote negated properties is exactly the same as the representation of negated actions discussed in an earlier section. For example, the output from the formation phase for the sentence "the i/o adapter is abnormal" is

```
@a39
   (truth-value (f))
   (args (i/o-adapter) (normal))
   (pr (has-property))
   (descr-subj (i/o-adapter
      (q (constant))))
   (descr-obj (normal (q (?))))
   (instance-of (description))
```

Notice that the descriptive relation **has-property** is negated. The meaning of the relation structure, @a29, is "the i/o adapter does not have the property of being normal." The CSM can determine that this structure denotes a negative property by examining the truth-value slot in search of an "f". A more difficult sentence to handle would be "the i/o adapter is not abnormal." In this case, the formation phase realizes that there is a double negation. The final structure, therefore, will not have a truth-value slot filled by "f", and we will not recommend "i/o-adapter failures" as a category of error messages.

Some predicates may indicate a failure or negation but are not explicitly negated. Examples of this type of predicate are *defective*, *down*, and *broken*. In these cases, the meaning of the predicate adjective is needed if we are to determine that a failure has occurred. Currently, a sub-hierarchy within SNOWY's LTM maintains knowledge of properties.

The CSM also identifies complex noun phrases that indicate faulty components, as in "the defective HIM..." or "the failed data bus...." This is accomplished by examining the x-structures of complex noun phrases for negative properties. If the x-structure of a complex noun phrase has a negative property, the CSM will save the super-concept of the x-structure as a possible classification. From the sentence "All further polling will cease pending component fault isolation of the failed HIM," we would like to recommend "HIM failures" as a possible classification. The relevant portion of the representation provided to the CSM by the formation phase is

```
x1
   (cf (is-a (HIM)) (@a41))

@a41
   (args (x1) (defective))
   (pr (has-property))
   (descr-subj (HIM (q (constant))))
   (descr-obj (defective))
```

Defective indicates a failure so the CSM saves the super-concept of x1, HIM, as a possible classification.

5.4 Prioritizing Recommendations

Once a set of candidate classifications has been determined from a sequence of text, the CSM orders the candidates from highly recommended to least recommended. Several orderings are possible.

- If it can be determined that the user's heuristic hierarchy is structured based upon component/sub-component relationships, then failed components should be highly recommended.
- If it can be determined that the user's heuristic hierarchy is structured based upon process/sub-process relationships, then verbal concepts that represent processes or at-time slot fillers which are processes should be highly recommended, e.g., "the microcode fails during initialization," or, "the microcode failed to initialize."
- If nothing about the user's hierarchy can be determined, then fall back on the structure of the factual hierarchy which is a structural one, i.e., failed components should be highly recommended.

By prioritizing the classifications, the most relevant classifications (determined heuristically using the rules above) can be presented to the expert as such. This helps when the set of possible classifications is large.

5.5 Part-Of Hierarchies

There may also be a hierarchical relationship between several of the candidate classifications, especially when the candidates are selected from text describing probable causes. For example, the probable causes for error 141 are:

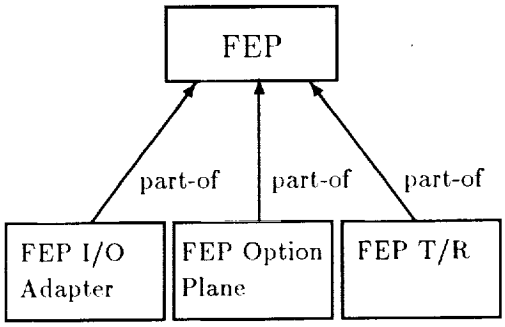

Figure 4: FEP Part-Of Hierarchy

1. `FEP i/o adapter failed.`
2. `FEP option plane failed.`
3. `I/O adapter port on the 4-port controller failed.`
4. `FEP transmitter/receiver failed.`

This leads to the failed component hierarchy shown in Figure 4. Probable causes 1, 2, and 4 describe the failures of sub-parts of a FEP. Determining that the FEP is related to i/o adapter, option plane, and transmitter/receiver by the has-part relation is the job of the noun-noun interpretation algorithms contained within SNOWY. That is, SNOWY is responsible for determining the meaning of complex noun phrases such as "FEP option plane," which is, "an option plane that is part of a FEP." The CSM simply has to look for a *part-of* relation under each of the sub-parts to recognize that a hierarchy exists. The existence of a part-of hierarchy provides strong evidence that the root of the hierarchy should be an error message category. In fact, it makes sense for the system to recommend the entire hierarchy to the expert.

5.6 Eliciting Additional Information

Up to this point, we have discussed how the NLC understands natural language input and how the CSM uses that understanding to identify and prioritize relevant categories of errors for presentation to the expert. The knowledge acquisition task does not end, however, when the expert selects a suggested classification. When experts accept suggested classifications, the CSM will "keep them talking" by prompting them with questions designed to trigger their recall of additional error message classifications. These questions prompt the expert for the names of similar messages that they feel would fall under the suggested category. The CSM also asks the user for other categories of errors that may be similar to the suggested category.

6 An Overview of the Natural Language Component

The NLC is an application of SNOWY. SNOWY is a system which integrates problem solving, knowledge acquisition, and information retrieval. In [Gomez & Segami, 1989] it was shown that, "in order for SNOWY to understand text, it needs to start with a minimum set of concepts which categorizes the world into states, actions, collections, etc." This *a priori* set of concepts, or ontology, is organized into a hierarchy based upon *is-a* relationships. The hierarchy is part of SNOWY's LTM. This LTM maintains the information that SNOWY has gathered from natural language input.

Each sentence presented to SNOWY undergoes three phases: a parsing and interpretation phase, a formation phase, and a recognition and integration phase. Because the recognition and integration phase is primarily concerned with updating SNOWY's LTM, which is unnecessary for our task, we only call upon SNOWY to parse, interpret, and form the expert's natural language input. These three processes are described below.

Parsing a sentence involves identifying its syntactic structures. The parser used by SNOWY is called WUP, which stands for word usage parser [Gomez 1989]. The underlying philosophy of WUP is that the syntactic usages of words play a greater role in parsing than is generally admitted. Discussion of how the usages are implemented and the details of the

operation of the parser will not be conducted in this paper, however. For our purposes, it is enough to know that the parser identifies the syntactic categories of the sentence. The syntactic categories used by WUP are: subject, verb, object, indirect object, prepositional phrase (PP), predicate, subordinate clause, and conjoined clause. The parser is not responsible for determining the attachment of prepositions, the verbal concept underlying the main verb, or the meaning of complex noun phrases. That is the duty of the interpreter.

6.1 Interpretation

The interpretation process is responsible for constructing the logical form from the syntactic constituents identified by the parser. This logical form represents the semantics of the sentence independent of any context. As each constituent of a sentence is identified, it is sent to the interpreter. It is important to point out that a constituent need not be interpreted the first time that it is seen by the interpreter. In fact, there are many cases where the interpretation of a particular constituent must be postponed until all the constituents of the sentence have been read. The constituent could be a noun phrase representing the subject or object of the sentence, in which case the interpreter must determine the meaning of the noun phrase. If the constituent is a verb, the interpreter must determine the underlying verbal concept that the verb represents. If the constituent is a prepositional phrase, the interpreter must determine its attachment and its meaning. Each of these three types of interpretation has its own set of interpretation rules. We will discuss each of the three types of interpretation and then culminate the interpretation section with a discussion of how this fits in with the domain at hand.

6.1.1 Noun Phrases

Interpreting noun phrases requires a great deal of knowledge of the meanings of nouns and adjectives. This is evident in the noun phrase "arthropod legs," which is the subject of the simple sentence "Arthropod legs are jointed." We can make sense of this phrase only because we know very well that arthropods, such as spiders and crustaceans, have legs. This knowledge allows us to determine that the NP above means "the legs that are part of arthropods." Without any knowledge of *arthropod* or *leg* we would be unable to determine a relationship between these two nouns. Similarly, knowledge of the adjective "wooden" is necessary to determine the meaning of the NP, "wooden legs," which is "legs made of wood." SNOWY stores knowledge of nouns as relations under their corresponding LTM concepts in SNOWY's concept hierarchy. Knowledge of adjectives is stored as interpretation rules. The noun phrase interpretation algorithm uses this knowledge when considering each pair of items in a given NP.

6.1.2 Verb Rules and Verbal Concepts

The interpreter algorithm makes use of verb rules to establish the underlying verbal concepts of sentences. These verbal concepts represent the meaning of the verb in the sentence. Below are the verb rules for the verb *dump*:

```
A Portion of the Verb Rules for the
            Verb Dump

(dump
  (((dump) (dumps) (dumped) (dumping)
    (has dumped) (had dumped))
      ((obj
         ((if part-of obj computer)
            (primitive-is transfer-data)
            (semantic-role-of-is obj
                from-loc)))))))
```

The "obj" slot contains a verb rule which will be tried when the parser passes the object constituent to the interpreter. This rule chooses the verbal concept **transfer-data** and marks the object as filling the from-loc case of this verbal concept in the event that the object of

the sentence is a part of a computer. Therefore, this rule would be used when SNOWY is interpreting a sentence like "Dumping the CPU registers would help isolate..." The interpretation of the main clause of this sentence would be "somebody transfer-data from the CPU registers (from-loc) to some unknown location (to-loc)."

While verbs like *dump* have very clear meanings, other verbs can be quite ambiguous. The verb *go* is reported to have 63 different meanings [Hirst, 1992]. We can side step this problem in most cases, however, because the domain of the incoming natural language is restricted to CCMS network error message descriptions.

Once the verbal concept has been determined, the interpreter attempts to fill the thematic cases of the verbal concept. Interpretation is now said to be driven by the verbal concept in the sense that we will attempt to place each of the other constituents within its framework. Thematic cases or roles show how noun phrases are related to the verbal concepts of sentences. Some of the most common thematic cases used by KL-SNOWY are: actor, theme, instrument, at-loc, from.loc, to-loc, at-time, init-time, end-time, descriptive-subject, and descriptive-object.

6.1.3 Prepositional Phrases

Interpreting prepositional phrases involves selecting the proper attachment (what sentence constituent is modified by the prepositional phrase) and establishing the meaning of the modification. Meanings and attachments are established by the verbal concept and interpretation rules under the given preposition [Gomez et al., 1992b]. Verbal concepts claim prepositional phrases through preposition rules (P-rules) stored under them. Noun phrases claim prepositional phrases through P-rules stored under the preposition.

6.1.4 Interpretation in the CCMS Network Domain

While interpretation of arbitrary text is currently an open problem, we can use the fact that we know the domain of the incoming discourse and the task of the CSM to limit the scope of the interpretation so that it is manageable. For instance, we have found that a significant percentage of the noun phrases used in error message descriptions and in the probable causes indicate specific components of the CCMS network.[4] The following is a table of some of the most common noun phrases in this domain:

Table 1: Common Noun Phrases in the CCMS Domain

active cpu	i/o card
common data buffer	data bus
error message	GSE FEP
ground data bus	LDB FEP
FEP option plane	PCM FEP
GSE data bus	standby cpu
GSE microcode	i/o adapter
HIM status register	option plane
system config table	

The semantics of these noun phrases can be captured by a few noun phrase interpretation rules. For instance, the phrase "data bus" is taken to mean a "bus for transporting data," where in this case *bus* is not a vehicle which makes frequent stops, but is a physical structure for transporting data and control information. Because we know the domain of the natural language input we will simply ignore the vehicle meaning of *bus*. A rule stored under the concept "data" will build the following interpretation when the noun phrase is interpreted:

```
(bus (transport (data)))
```

Another rule will look for part-of relationships between the nouns in noun phrases. This

[4]The components may be hardware components or software programs and data structures.

rule captures the meaning of phrases like "GSE data bus," "GSE FEP," "GSE microcode," and "FEP option plane." The interpretations of these four phrases are:

```
((bus (transport (data)))
    (part-of (GSE)))
(FEP (part-of (GSE)))
(microcode (part-of (GSE)))
(option-plane (part-of (FEP)))
```

Of course, to determine these part-of relations we must know *a priori* the physical structure of the CCMS network. This *a priori* information has been assembled by a knowledge engineer and is stored in AKAI's factual hierarchy. Therefore, we can determine these relations simply by consulting the factual hierarchy.

Verb rules need to be provided for the verbs commonly used in error descriptions and probable causes. A list of the verbs, for which verb rules were added, is given in Table 2. Each of these verb rules must specify a verbal concept.

Table 2: Verbs needing New Verb Rules

activate	fail	poll
command	generate	reset
detect	initialize	respond
dump	isolate	

Table 3 lists the new verbal concepts created for this domain.

Table 3: New Verbal Concepts

activate	initialize
command	isolate
become-aware	poll
fail	reset
fail-negation	respond
generate	transfer-data

Because the verb rules and verbal concepts added to SNOWY are dependent on knowing the LTM categories for nouns commonly used in the CCMS domain, it was necessary to add them to SNOWY's *a priori* hierarchy. Table 4 is a partial list of the concepts that were added to SNOWY's LTM.

Table 4: New Concepts added to LTM

acknowledgement	LDB
adapter	microcode
board	option-plane
buffer	PCM
bus	register
card	signal
computer	switch
cpu	transceiver
FEP	transmission
HIM	uplink
i/o	

6.1.5 Formation Rules

Formation rules are stored under verbal concepts. When the formation algorithm is activated, it looks to see if the verbal concept selected by the interpretation process has any special formation rules stored under it.[5] If formation rules are found, the normal formation algorithm is overridden and the system attempts to fire them. If a rule fires successfully, its consequent list is evaluated, effectively taking over the task of formation. Let us now discuss an example of a formation rule written by the authors to handle a special construction used in the CCMS domain.

Negated relations may come from sentences which use the "fail to" construction, or from sentences with explicitly negated verbs. The "fail to" construction is one in which the main verb of the main clause is *fail* and *fail* is followed by an embedded clause beginning with the word *to*. The representation of sentences using this type of construction is a relation structure representing an embedded clause whose verbal

[5]This discussion assumes that the interpreter was able to determine a verbal concept. In the event that no verbal concept was selected, the formation phase will be unable to construct a relation structure and is aborted.

concept has been negated. The formation rule responsible for creating this structure, shown below, is stored under the verbal concept **fail-negation** in the *f-rules* slot.

```
(fail-negation
    (is-a (description))
    (subj (thing (descr-subj)))
    (obj  (proposition (descr-obj)))
    (f-rules
        (fire-all
            ( t (negate-relation)))))
```

This rule calls a LISP function, called *negate-relation*, to negate the embedded clause. Take for example the sentence "The FEP failed to detect a response from the i/o adapter." We would like to end up with KL-SNOWY structures that represent that the FEP did *not* become aware of a response from the i/o adapter. Therefore, the task of the negate-relation function is to place an *f* in a *truth-value* slot of the relation structure associated with the embedded relation "[FEP] detect a response from the i/o adapter."

7 Conclusions and Future Work

We have shown how natural language input can be used to infer classifications suitable for inclusion into the heuristic hierarchies of AKAI, in a real world environment. We are currently in the early stages of the implementation of these ideas. Very little work needs to be done on the NLC, per se, because SNOWY is a working system. The bulk of our effort is, therefore, focused on implementing the CSM. Nevertheless, there are several data files used by SNOWY that must be scaled up if the enhancements of AKAI are to work "outside the lab."

One such data file is SNOWY's lexicon. To address this problem, a machine-readable dictionary created by the Summer Institute for Linguistics, called Englex, is being adapted for use by SNOWY. Specifically, entries in Englex are being converted into a format assimilable by SNOWY and added to SNOWY's lexicon. Englex contains morphological data for approximately 11,000 nouns, 4000 verbs, and 3400 adjectives, as well as adverbs, acronyms and abbreviations, proper nouns, prepositions, determiners, conjunctions, quantifiers, etc. Especially useful are markers indicating negative prefixes and nominalizations for nouns. By incorporating these words into SNOWY's lexicon, we hope to minimize the problem of encountering unknown words during the parsing of an expert's description.

Other data that will need to be expanded are SNOWY's verb rules and verbal concepts, interpretation rules for interpreting complex noun and prepositional phrases, and new formation rules for handling special sentence constructions. At first glance this task may seem quite daunting, but because we are receiving natural language input constrained to the domain of CCMS network messages, we can expect a limit to the diversity and complexity of the incoming text. This claim is supported by an analysis of the text that makes up the probable causes and advisory data currently stored in OPERA.

While extension of the NLC involves data, work on the CSM requires coding changes. It is important to note that the complexity of implementing the CSM is significantly reduced by the robustness of SNOWY's representation. Determining failures and their related cases is a simple task, assuming that SNOWY has been able to create the appropriate structures. This underscores the importance of an adequate representation for the purpose of acquiring knowledge.

Acknowledgements

We would like to thank R. Bruce Hosken and William Verhagen of Grumman Technical Services, KSC, for providing us with a set of typical error message descriptions, and the referees who made valuable comments.

References

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

Boose, J. and Bradshaw, J. (1988). Expertise Transfer and Complex Problems: using Aquinas as a knowledge acquisition workbench for knowledge-based systems, in J. Boose and B. Gaines *Knowledge-Based Systems,* vol. 2, Academic Press: London.

Boose, J., Shema, D. & Bradshaw, J. (1989). Recent Progress in Aquinas: a knowledge acquisition workbench. Journal of Knowledge Acquisition, 1, 185-214.

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

Gaines, B.R., and Shaw, M. L.G. (1981). New Directions in the analysis and interactive elicitation of personal construct systems, in M.L.G. Shaw (ed). *Recent Advances in Personal Construct Technology*, Academic Press: New York.

Gomez, F. (1989) WUP: A Parser based on Word Usage. UCF-Tech-89-2, Dept. of Computer Science, University of Central Florida, Orlando, FL, 32816.

Gomez, F. & Chandrasekaran, B. (1984). Knowledge organization, and distribution for medical diagnosis. In W. Clancey & E. Shortliffe, Eds. *Readings in Medical Artificial Intelligence.* Reading, MA: Addison-Wesley.

Gomez, F., Hull, R., Karr, C., Hosken, R.B., & Verhagen, W. (1992a) Combining Factual and Heuristic Knowledge in Knowledge Acquisition, *Telematics and Informatics*, 9(6), Dec., 1992.

Gomez, F. & Segami, C. (1989) The Recognition and Integration of Concepts in Understanding Scientific Texts. *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 1, 51-77.

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

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

Gomez, F., Segami, C., & Hull, R. (1992b) Prepositional Attachment, Prepositional Meaning, Verb Meaning, and Thematic Roles. UCF-Tech-92, Dept. of Computer Science, University of Central Florida, Orlando, FL, 32816. (Submitted for Publication)

Hirst, Graeme. (1992) Semantic interpretation and the resolution of ambiguity. Cambridge University Press.