

Naval Research Laboratory

Washington, DC 20375-5000



NRL Report 9299

**InterFIS: A Natural Language Interface
to the Fault Isolation Shell**

D. PERZANOWSKI AND B. POTTER

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

December 31, 1990

InterFIS

A NATURAL LANGUAGE INTERFACE TO THE FAULT ISOLATION SHELL

INTRODUCTION

InterFIS is a natural language interface used to interface with an electronics troubleshooting expert system that is part of an expert system development tool called the Fault Isolation Shell (FIS). In this report we present a brief overview of one module of the expert system and then discuss the natural language interface with this system. Our focus here is on how InterFIS maps natural language input into an appropriate representation for the expert system. We also discuss our research approaches, their implications, and the questions raised by our work. We conclude with a brief discussion of what work is required in the future.

FIS

FIS runs on SUN workstations under a UNIX environment and is written in Lucid Common Lisp. The domain of FIS is electronics troubleshooting of analog equipment [1]. The principal functions of FIS are

- to aid the knowledge engineer in producing concise descriptions of electronic equipment in a particular domain;
- to compute the probability that a particular fault hypothesis is correct after one or more tests have been performed on a particular piece of electronic equipment;
- to recommend the next best test to make based on the information supplied by a diagnostician during a testing session.

The first function of FIS is part of the knowledge acquisition component of the expert system, and the other two functions are part of the troubleshooting module. Here we are not concerned with the knowledge acquisition module. A technician who wishes to perform certain electronic troubleshooting tests on a particular piece of electronic equipment accesses the troubleshooting module through one of three interfaces (Fig. 1).

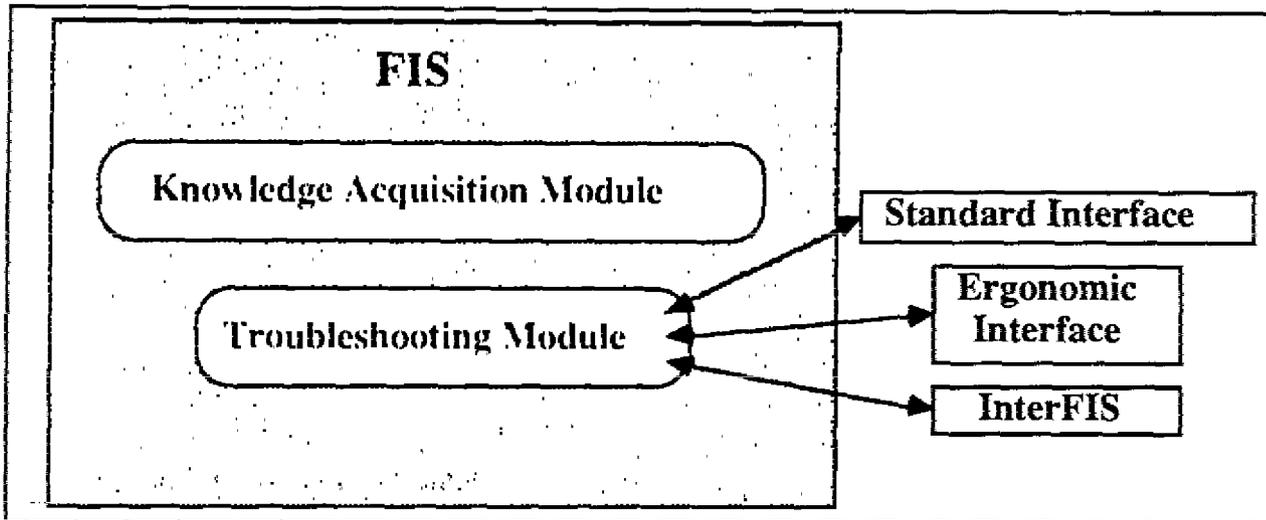


Fig. 1 — Schematic of FIS and the various interfaces with the expert system shell

Fig. 1 shows the overall architecture of FIS, the independent knowledge acquisition and troubleshooting modules, the standard interface, and the ergonomic interface. This report is concerned with the natural language interface, InterFIS. The technician interacting with the troubleshooting module of FIS can interface with the expert system through mnemonic commands arranged in a hierarchical series, hereafter referred to as the standard interface [2]. The technician can also access the same hierarchical sequence of commands through a mouse-driven interface, hereafter referred to as the ergonomic interface¹ [3]. We have added the natural language interface. Since we have limited our research to interactions with the troubleshooting module, the natural language interactions that we are concerned with here do not encompass the kinds of interactions that knowledge engineers would experience. Specifically, the natural language interactions are those of a technician interacting with the expert system to determine unit fault or optimal test diagnostics.

Interactions with any mnemonic series of commands in a menu format can be obscure at times, and a naive user can frequently be intimidated by the menus and their cryptic commands. Natural language communication is, on the other hand, by definition natural. The user seldom needs additional training to interact with a system that has a natural language interface. Natural language interfaces, however, are not designed to replace existing interfaces. Multimodal interaction is desirable, and a natural language interface should be only one of several means of interaction with expert systems. Therefore, we envision InterFIS not only to encompass natural language interaction but also to complement the standard and ergonomic interfaces of this expert system.

At the present time, InterFIS is keyboard driven. Thus, the diagnostician must type in all the interactions as a sequence of words followed by a period or a question mark, just like normal typing

¹In this study, no comparisons are made between either the standard interface or the natural language interface (InterFIS) and the ergonomic interface. The latter interface is no longer supported at this research site.

at a keyboard. We assume here that the natural language interface, although faster to learn than the other interfaces, is slower because it is a keyboard entry device. It takes longer to type some of the commands than to type in a two-character mnemonic command or to click on an icon. Performance tests would undoubtedly corroborate this belief.

Regardless of which interface is used, interactions with all three interfaces change the unit under test (UUT) as it is displayed on the monitor. Fig. 2 is a schematic drawing that shows what a typical monitor looks like when a diagnostician interacts with FIS through the natural language interface, InterFIS.

The bottom SUN window in Fig. 2 is the interactive window that communicates with the expert system. The user initializes the system from this window and interactively loads a unit for testing. A colorized graphic representation of the topology of the unit appears in the SUN window at the upperpart of the monitor. At various stages during the troubleshooting process, different modules (the darkened boxes in the upper window) and test points of the UUT (the shaded triangles) are suspected as possible places to be tested for fault. On the display monitor, the representations of the various suspected modules change color according to the degree of their probability [2] of causing the fault. Various numerical fault probabilities are also displayed in the darkened boxes or the modules in this upper graphics window.

In the lower interactive window, the user requests or commands FIS to perform the diagnostic tests or to display other lists of information, which also appear in the interactive window. In Fig. 2, the user has requested through InterFIS that the system display or show the fault probabilities of the last failed test. The system responded with a list of fault probabilities and certification rates. The user then requested that the system test the frequency at one of the terminal or test points, namely MFET.

The other two interfaces to FIS display their results similarly.

OVERVIEW OF INTERFIS

To produce representations suitable for the expert system to process, the natural language interface processes the natural language input in several distinct steps. InterFIS first determines the syntactic constituents of the input and normalizes grammatical English sentences in the syntactic component. The input sentence or string is then regularized into canonical forms in an operator/operand notation. These representations are then checked against domain-specific verb models by using predefined noun classes for the corresponding arguments and transforming them into predicate-argument notation. This latter representation then undergoes semantic interpretation in the Command-Oriented INterpreter (COIN) component. This component maps predicate-argument forms to FIS-commands that the expert system can then process (Fig. 3).

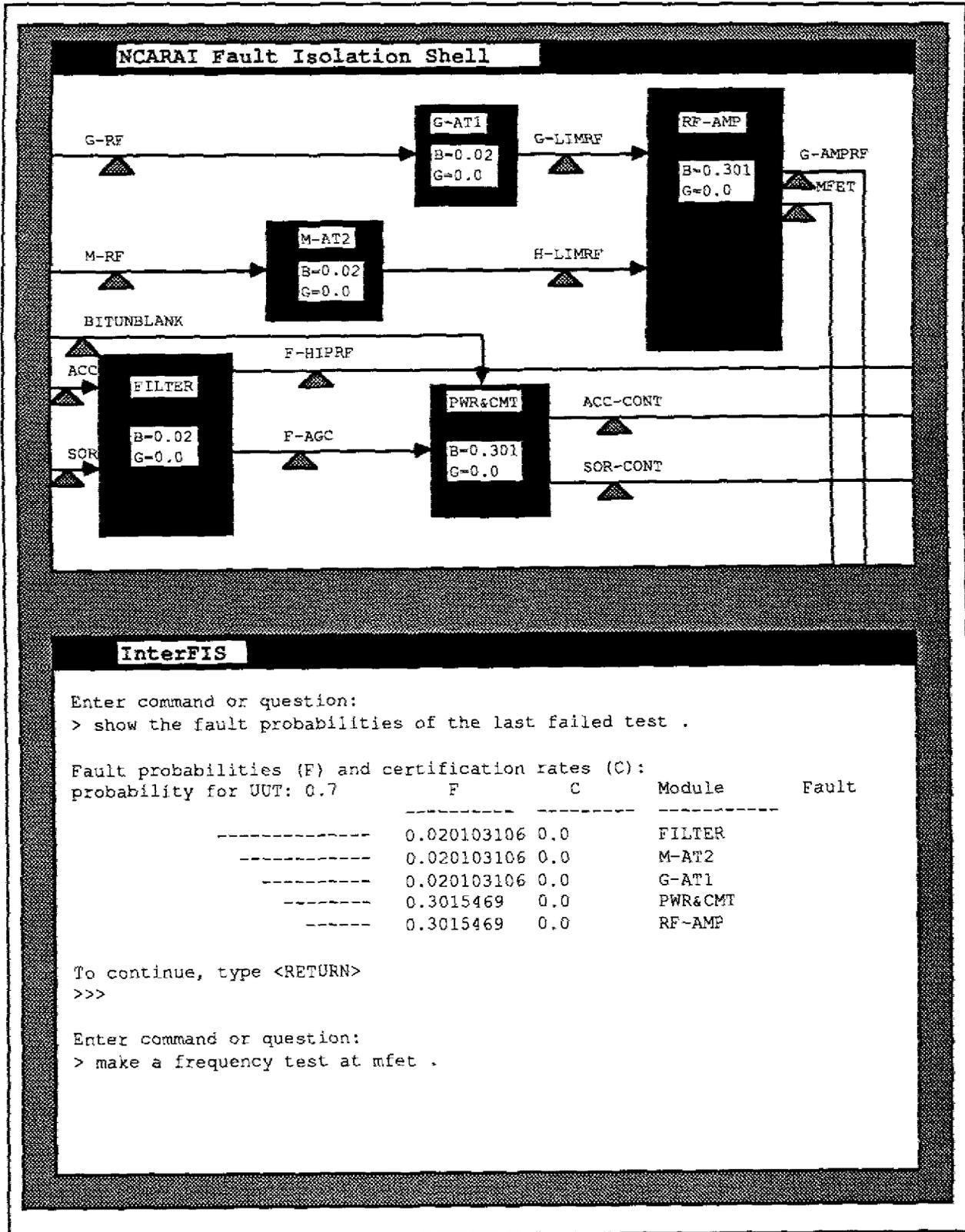


Fig. 2 — FIS interactive diagnostic session with InterFIS

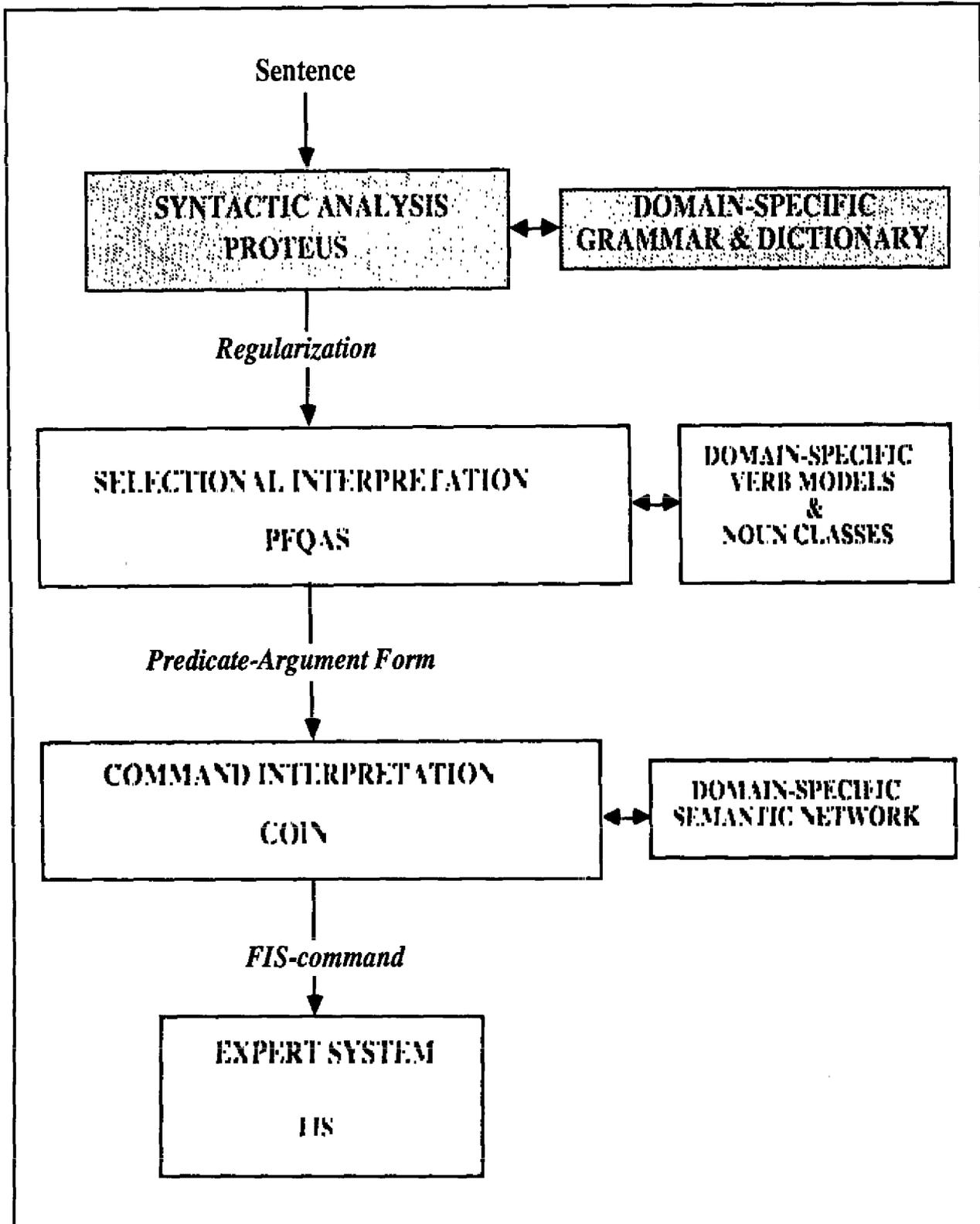


Fig. 3 — Overview of input processing in InterFIS

We discuss in this report the natural language processing of the input to the expert system: how InterFIS parses the input in its syntactic component, normalizes the output of the syntactic component; and then translates the output into a predicate-argument form that is matched with an appropriate FIS-command in the expert system by COIN.

In this report, we limit ourselves to approximately 30 FIS-commands taken from the standard interface. These commands comprise the topmost level and several items from a sublevel of the expert system. We used menu items such as SHOW-HISTORY, SHOW-ACTIVE-LIST, and SIM-FAULT (SIMULATE-FAULT). Time did not permit us to look at the entire set of FIS-commands. Furthermore, handling some of these additional FIS-commands would have entailed the incorporation of a discourse component that is not a part of this study. By using the 30 FIS-commands mentioned above, we formulated a set of English sentences and their possible paraphrases. We tried to translate each FIS-command into at least three or four possible paraphrases that were syntactically general enough to allow for rewording, elimination or incorporation of syntactic elements, rephrasing through nominal compounding, and the like. For example, the FIS-command SHOW-PROBS (SHOW-PROBABILITIES) has at least seven paraphrases, as shown in "(1) Data".

(1) — Data
<ul style="list-style-type: none"> • Show the current fault probabilities of the unit. • Show the fault probabilities of the unit's modules. • Show the fault probabilities of the unit modules. • Show the fault probabilities on the console. • Show the probabilities list. • Show the list of probabilities. • Show the probabilities.

We also included the synonyms for the verbs in this domain. Possible synonyms for SHOW are DISPLAY, LIST, and GIVE, to name but a few. All of these verbs are mapped into the same domain predicate DISPLAY.

With a core set of paraphrases consisting of approximately 120 sentences, we developed a grammar and domain model. We assume that by using this grammar and domain model, more paraphrases can be mapped into appropriate representations. As a result, the total number of interpretable sentences is much larger; it is dependent solely upon the completeness of our linguistic analysis.

SYNTACTIC ANALYSIS

Sentences are scanned by using a word dictionary or lexicon that consists of approximately 425 words. The sentences are grammatically analyzed by the PROTEUS parser, a context-free chart

parser [4]. The syntactic parser uses two inputs: an input string that contains lexical information for each word in the sentence, and a grammar written in PROTEUS Restriction Language, which is a variant of Backus-Naur Form (BNF). The lexical scanning in PROTEUS ensures that all lexical items of the input string are spelled correctly and that each of them has a dictionary entry. The lexicon contains entries like that shown in "(2) Lexical Representation". Each entry identifies the lexical category of a word and specifies some subcategorization features that may be either syntactic or semantic.

(2) — Lexical Representation
(word FIS (N (NONHUMAN PROPER Xn (FIS SINGULAR))))

In "(2) Lexical Representation", for example, the word FIS is a Noun (N), and its syntactic-semantic features are that it is a NONHUMAN and PROPER noun. The remainder of the list, the Xn element, is a base form or lexical regularization of the noun. When the lexical item FIS is entered into the regularization, the input appears in the list structures in its regularized, or Xn, form as FIS SINGULAR.

Each syntactic rule has three parts: the syntactic category, its syntactic structure and a syntactic regularization. In "(3) BNF Definition for Imperatives", we use the BNF to expand English imperative sentences into syntactic categories and a syntactic regularization.

(3) — BNF Definition for Imperatives
<pre><IMPERATIVE> ::= <SA> <IVERB> <N-NOT> <SA> <OBJECT> <SA> : {(tell IVERB you !OBJECT !SA* !N-NOT)}.</pre>

As the structural description for IMPERATIVES on the right-hand side of the rule is met, the various constituents, such as <SA> or <Sentence Adjunct>, <IVERB> or <Imperative Form of the VERB>, are further expanded until each word of the input sentence matches each of the constituents of the major syntactic category IMPERATIVE. PROTEUS then regularizes [5] the syntactic parse into a list structure, indicated in "(3) BNF Definition for Imperatives" as the list enclosed in brackets. As "(3) BNF Definition for Imperatives" shows, all imperative verbs (IVERB) are prefixed by the performative operator TELL, followed by the lexical item (IVERB) and the understood subject YOU of English imperatives. The remainder of the regularization for imperatives contains the OBJECT string and Sentence Adjuncts such as adverbial modifiers, which are similarly regularized by following rules¹.

¹Iterative occurrences of a node are indicated by *, and OBJECT and NOT-strings are spliced (as indicated by the !-notation) into the remainder of the list.

A sub-component of the grammar is the Restriction Component, which syntactically constrains and limits the number of acceptable parses. For example, in "(4) Restriction Language Definition", the restriction WIO only permits HUMAN objects of verbs that are subcategorized¹ for NN in their OBJLISTs², such as SHOW and GIVE.

(4) — Restriction Language Definition
WIO = IN NN: IF FIRST-ELEMENT IS NOT EMPTY THEN CORE OF FIRST-ELEMENT HAS ATTRIBUTE NHUMAN.

Thus, SHOW (ME) THE LIST and GIVE (ME) THE PROBABILITIES are grammatical, but SHOW THE HISTORY LIST is not, where HISTORY is parsed as the indirect object; i.e., as in the ungrammatical sentence SHOW (TO) THE HISTORY (THE) LIST.

Both the BNFs and the Restriction Component are a direct descendent of the Linguistic String Project grammar [6], but they have been adapted and augmented for the FIS domain by using procedures outlined in Ref. 7 for developing sublanguage grammars.

The final output of the syntactic processing is a normalization as described in "(5) Sample Sentence With a Sample Representation" of the input sentence in an operator/operand notation, similar to the "semantic" representations found in a Generalized Phrase-Structure Grammar [8,9].

(5) — Sample Sentence With a Sample Representation
Show the fault probabilities of the last failed test . = (SHOW (you) (the N2 probability plural (AN-STG ((fault singular))) (OF (the #N6 test singular (AN-STG ((AN-STG (last) fail)))))) (me))

¹Verb subcategorization identifies the restrictions on the types of syntactic objects that a verb can take. These restrictions are characterized in the OBJLIST for each verb in the lexicon.

²The OBJLIST, or OBJECT LIST of a verb, is a list of the syntactic complements to the verb. In other words, a verb like SHOW can have in its OBJLIST NSTGO (Noun STRing Objects) (ia), THATS (that + Sentence), (ib), or even NULLOBJ (NULL or no OBJECT) (ic), to name but a few.

- (i) (a) The test showed the results of the experiment.
- (b) The test showed that the results were positive.
- (c) His embarrassment showed.

Next, strings are regularized so that the predicates of those strings precede their grammatical arguments. For example, English declarative sentences do not have their verbs first, as in THE SYSTEM SHOWED THE LIST, but they do regularize to something like "(6) A Regularization".

(6) — A Regularization
(ASSERT (PAST SHOW (SUBJECT THE SYSTEM) (OBJECT THE LIST))

Some interrogatives, such as WHAT IS THE CURRENT LIST OF TESTS? have a different syntactic shape. However, this interrogative will regularize to something like what is shown in "(7) A Regularization".

(7) — A Regularization
(ASKWH (WHICH THING NIL) (PRESENT BE (SUBJECT VAR) (OBJECT THE LIST SINGULAR (AN-STG (CURRENT)) (OF (NULL-DET TEST PLURAL))))))

Except for the initial elements in "(6) A Regularization" and in "(7) A Regularization", namely ASSERT in "(6) A Regularization" and ASKWH (WHICH THING NIL) in "(7) A Regularization" respectively, the remainder of the regularizations are structurally similar. Some initial performative and tense information precede the predicates. ASSERT in "(6) A Regularization" is a place holder for the performative information of declarative information, while ASKWH in "(7) A Regularization" holds the performative information. The predicate is, in turn, followed by its logical arguments, namely SUBJECT and OBJECT, if one exists. These representations or regularizations capture the similar predicate-argument structure of the two syntactically different input strings. Such regularized representations of the input facilitate further processing, since the inherent information of a sentence, regardless of sentence type, is presented in the same format.

This representation is checked against certain domain-specific information. The resulting predicate-argument form is then analyzed by an application-specific interpreter that maps the syntactic-semantic representation into an appropriate representation or FIS-command for the expert system.

A slightly different procedure is used to process conjoined constructions or sentences. When a conjoined construction is found in the input sentence, the PROTEUS parser modifies the input before passing it on to the selectional interpreter, PFQAS (Fig. 3). In particular, conjoined input is broken into two or more individual structures. Conjoined Imperatives, such as those in "(8) Decomposition of Conjoined Imperatives", are simply split into separate but complete commands.

(8) — Decomposition of Conjoined Imperatives
Show the history and make a test. ::= (and (show the history) (make a test))

When an NP object involves conjunction, as in "(9) Decomposition of Conjoined Objects", certain linguistic items must be duplicated before processing the sentence.

(9) — Decomposition of Conjoined Objects
Make a test and a best test . ::= (and (make a test) (make a best test))

This duplication of elements is done before semantic processing during the syntactic analysis of the input.

Note that the conjunction functions act recursively, so that any amount of conjoining is possible and conjunction at different levels of embedding will be accounted for. Thus, although the expert system requires that the input be a single FIS-command, if the input is an extremely complex English sentence, such as "(10) Sentence Exhibiting Various Conjoined Elements", InterFIS will analyze the input into its constituent commands and present them serially to the expert system.

(10) — Sentence Exhibiting Various Conjoined Elements
Load and draw the unit on the screen and show the ambiguity and history lists.

Each of the separate commands in "(10) Sentence Exhibiting Various Conjoined Elements" is sent to the selection component, interpreted individually, and passed separately to the expert system.

SELECTION

The PFQAS Component

The PROTEUS-derived syntactic parse of the input sentence, or normalization, is passed to the semantic interpreter, PFQAS, which is an adaptation of the selection component of the Question-Answering System [4]. PFQAS tests the semantic integrity of the parse, prunes the number of possible parses and determines that the appropriate verb models with corresponding noun classes exist. PFQAS maps the parse to what we call a predicate-argument form. A predicate-argument form is like a predicate-calculus representation of a sentence to which subject and object labels have been appropriately inserted.

Semantic selection is coded in PFQAS through verb models and noun classes. The verb LEAVE might be defined in the PFQAS domain-specific verb models as in "(11) Verb Model For The Verb Leave".

(11) — Verb Model For The Verb Leave
(leave (subject you) (object nfisnoun) (to nsystemnoun nil optional))

An input sentence containing LEAVE as its main verb is deemed semantically acceptable by PFQAS if it fits the structure dictated by this model. In other words, the input sentence must contain an object argument recognized as a member of the noun class NFISNOUN and a TO-argument of the class NSYSTEMNOUN. OPTIONAL in "(11) Verb Model For The Verb Leave" also indicates that the TO-argument is optional, with NIL being required as the place-holder for the processing of the optional operand. Noun classes are defined as sets in PFQAS "(12) Sample Noun Classes And Members Of Their Sets".

(12) — Sample Noun Classes And Members Of Their Sets
(nfisnoun (fis system environment)) (nsystemnoun (unix lisp system))

As an example of how subcategorization in the lexicon and selection in the verb models operates, consider "(13) Sample Sentence".

(13) — Sample Sentence
Exit fis to unix .

In the verb models, EXIT is identified as a verb of the type LEAVE; FIS is an NFISNOUN and UNIX is an NSYSTEMNOUN. Thus, "(13) Sample Sentence" is an acceptable utterance in this domain, based on the verb model that it maps to, as seen in "(11) Verb Model For The Verb Leave". Since the TO-argument in "(11) Verb Model For The Verb Leave" is optional, a paraphrase of "(13) Sample Sentence" EXIT FIS, will also pass selection.

Note that we have written PFQAS to allow complex noun phrases, such as UNIX SYSTEM in "(14) Sample Sentence".

(14) — Sample Sentence
Exit the current environment to the unix system.

"(14) Sample Sentence" is a semantically valid sentence, just as "(13) Sample Sentence" is.

Wh-questions are mapped somewhat differently in InterFIS. InterFIS maps wh-questions to imperatives, like the one in "(15) Example of Mapping".

(15) — Example of Mapping
What tests have been performed? ::= Show the performed tests .

In PFQAS, we have assumed that wh-questions beginning with the interrogative pronouns, such as WHO, WHAT, and WHICH¹ map to the performative DO and the verb SHOW. By allowing this mapping, an interrogative sentence like the one in "(15) Example of Mapping" will eventually map to an appropriate FIS-command. Our assumption is that the underlying intent of "(15) Example of Mapping" is for the expert system to perform some action. Thus, mapping an interrogative to a performative verb is justified.

The COIN Component

Having achieved a valid predicate-argument form of the input sentence, control is then passed to a series of functions that together are called COIN. The role of this module is to map the predicate-argument form to its real-world meaning or representation. In expert systems, real-world meanings are function calls that correspond to commands that users input to the environment either through mnemonics, menus, or, in this case, natural language.

¹We have not looked at WHERE, HOW, and WHY for principled reasons discussed later in this report.

COIN is made up of three major components: the semantic network; the net-traversal component; and the move-stack building component. The semantic network of domain-relevant lexical items contains all the information on expert system function calls. Functions are picked from this network by the net-traversal component that follows a list of commands for movement through the network. This list of commands is formed by the move-stack building component.

COIN's semantic network is the only application-specific element of InterFIS. In brief, the network is a representation of the expert system's knowledge about objects (nouns), their attributes (adjectives and complements), and commands (verbs) associated with those objects. In other words, encoded in the network are the possible relationships among relevant nouns and adjectives and their corresponding interactions with verbs. At the core of the network lie the expert system function calls. All natural language is ultimately translated to these function calls.

The main unit in the network is the lexical node. Any significant adjective or noun in the expert system is represented by a node. Nodes are connected to each other through paths that correspond to different means of nominal modification. For example, there are adjectival paths that denote adjectival modification and relational paths that denote of-complementation (Fig. 4).

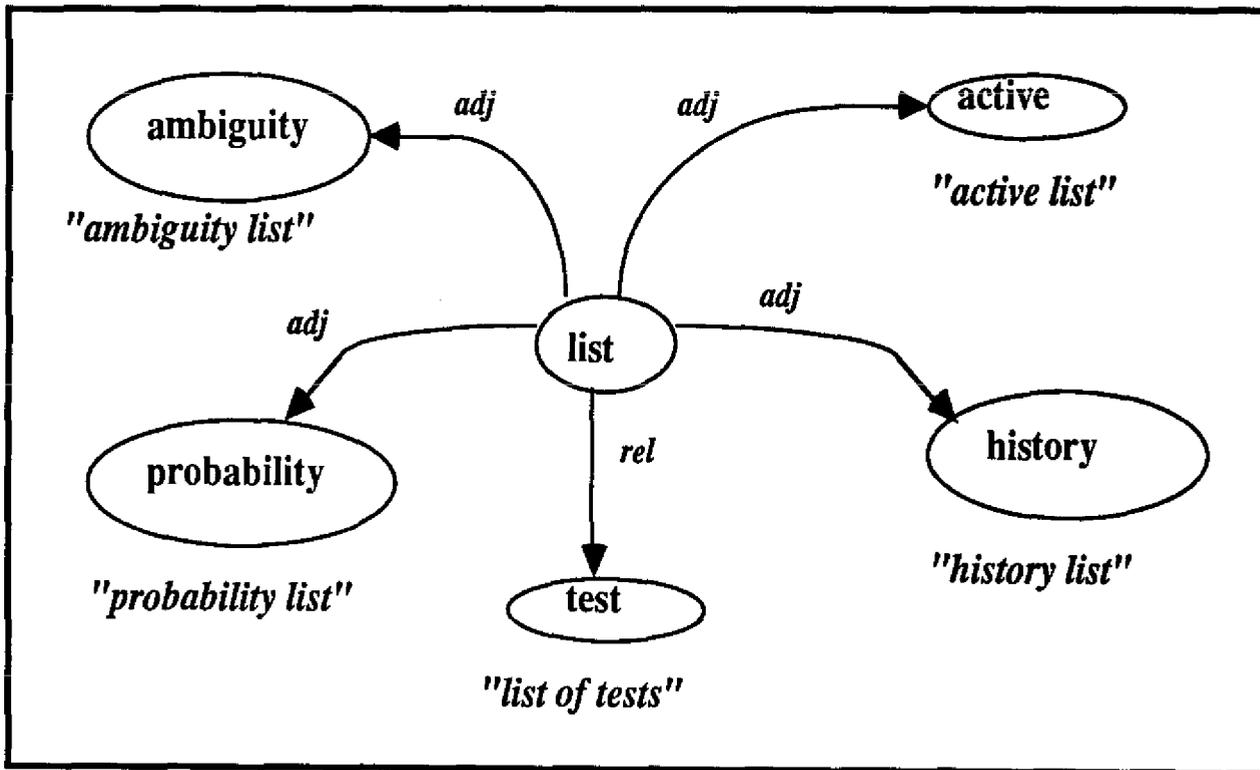


Fig. 4 — Simplified example from InterFIS semantic network

Fig. 4 shows that nominals, such as PROBABILITY LIST and LIST OF TESTS, are also accounted for by this net-traversal function.

Within each node information exists that maps node-relevant verbs to expert system function calls. Fig. 5 shows the example of the lexical node TEST that contains information that the verb MAKE refers to in order to call the function CMD-MAKE-TEST.

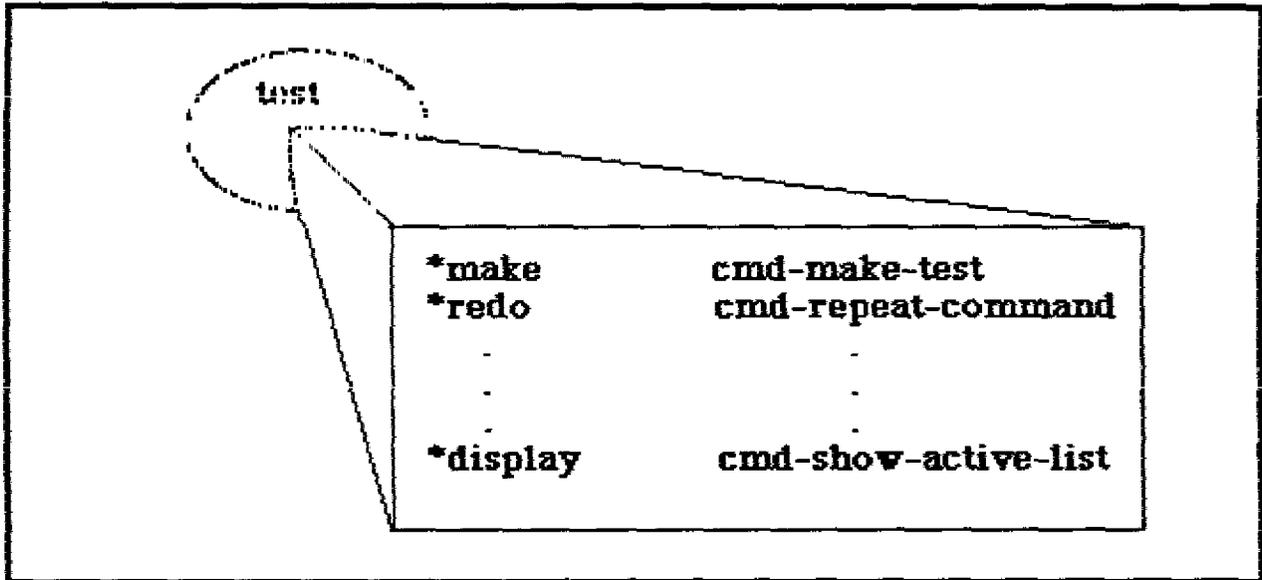
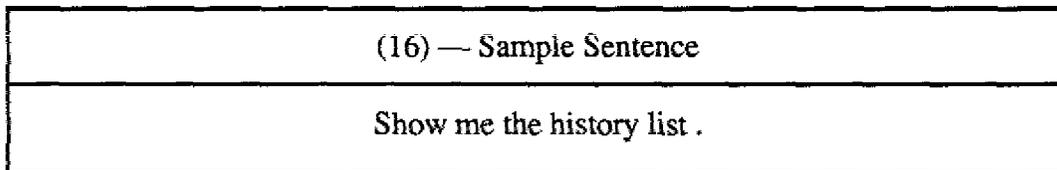


Fig. 5 — Partial information for Test node

In determining the expert system function call relevant to the natural language input, the basic strategy of the COIN net-traversal component is to select the verb-relevant function call in the last lexical node encountered. Traversal of the network begins at the lexical node corresponding to the head or main noun of the syntactic object, such as LIST in "(16) Sample Sentence", and proceeds according to the list of moves stacked for traversal through the network by COIN's move-stack building component.



The move-stack building functions operate on the PFQAS predicate-argument form, decomposing the PFQAS structure into ordered units of linguistic modifiers. As traversal through the semantic network progresses, COIN picks up all relevant function calls encountered. When traversal of the network is complete, the final function call collected is executed.

Several circumstances may alter the basic traversal strategy. One important case is quantification. When the move-stack building functions of COIN encounter a universal quantifier in the input sentence, a list of move-commands is developed. Each set of commands in this list corresponds to the move-command relevant to a particular instantiation of the quantified NP. In "(17) Sample of Quantification", four move-stacks will be created, because four kinds of lists are in the FIS domain.

(17) — Sample of Quantification
Show all the lists . ::= <ul style="list-style-type: none"> • Show the active list . • Show the history list . • Show the probabilities list . • Show the ambiguity list .

Another case that can alter the basic net-traversal strategy is the inclusion of parameters or variables in the input sentence. Some expert system function calls require one or more arguments for proper execution. Thus, nodes that represent words potentially used as parameters are flagged as special. During traversal of the network, any such nodes encountered are noted and are used as arguments in the execution of the final function call. For example, in "(18) Sample Sentence", FREQUENCY and G-IF are parameters relevant to the MAKE-TEST function hierarchy.

(18) — Sample Sentence
Make a frequency test at g-if .

As such they will be passed as arguments to a function called by MAKE-TEST.

APPROACHES, IMPLICATIONS AND QUESTIONS

Grammar

Although InterFIS currently can process imperative sentences in English in the FIS domain, the present grammar does not handle certain types of interrogative and declarative sentences. These limitations of the natural language system correspond to the limitations of the expert system. InterFIS does grammatically process the commands for which there is a corresponding FIS function at the highest level of interaction.

Interrogative questions, such as WHAT IS THE HISTORY LIST? and WHICH UNIT IS LOADED? will parse and an appropriate FIS-command will be returned. However, other interrogatives, for example HOW, WHO(M), WHERE, and WHY, do not have corresponding interpretations or functions and are therefore not in the grammar.

Furthermore, we did not pursue yes/no type questions such as **WAS A TEST RUN?** Yes/no questions require yes/no answers. Although the information may be available in the FIS domain, FIS functions do not exist to access this information.

Finally, we did not have the time to incorporate declarative structures, such as **I WOULD LIKE TO SEE THE HISTORY LIST**. However, as Ken Wauchope (personal communication) suggested, such declaratives could be interpreted as **DO** performatives, thus mapping into the domain-predicate **SHOW**.

Interfacing

We decided not to modify the existing expert system code or add to it but only to use available system functions and variables. This preserves the integrity of the expert system. However, within this framework we took advantage of the productivity of natural language, wherever possible. Although our underlying philosophy was not to modify any existing FIS code or add to it, the natural language interface did extend some of the capabilities of the expert system.

First, natural language has the capability of paraphrasing locutions. When InterFIS was interfaced to the expert system, FIS acquired this functionality as well. For example, the FIS mnemonic function **SH** in the menu-driven interface corresponds to the FIS command **SHOW-HISTORY**. The natural language interface, on the other hand, maps any equivalent paraphrase — **SHOW THE HISTORY LIST**; **WHAT TESTS CAN BE REMADE?**; **WHAT IS THE LIST OF TESTS RUN?**; and **WHAT TESTS CAN YOU REMAKE?**, etc. — to **SHOW-HISTORY**.

Similarly, the natural language capabilities of conjunction and quantification are passed to the expert system. For example, sentences like **MAKE A TEST AND BEST TEST** and **MAKE A TEST AT EVERY TERMINAL** can be processed regardless of how the expert system is organized. The latter sentence also permits the user to input a request once, rather than having to type a series of mnemonic commands.

By using the natural language interface, it becomes possible to access the expert system in ways not initially available. In other words, through natural language, a user can request actions and pieces of information that the expert system is capable of supplying, whether a specific mnemonic command existed to correspond to that request or not. For example, FIS has a function **ENHANCED-AMB-SET** that is accessed only through another FIS function and is not directly available to the user of the mnemonic interface. The user of InterFIS, however, can type in **SHOW ME THE ENHANCED AMBIGUITY SET** at any time, and the expert system will respond by displaying the enhanced ambiguity set on the screen.

Likewise, this ability to call any function in the expert system code allows the user to compress information that normally requires several interactions into one input sentence. For example, in the mnemonic interface the command **MT** corresponds to the function **MAKE-TEST**.

This function requests information on the location of the test and in turn calls other functions requesting additional information. With natural language the user may type in the command MAKE A FREQUENCY TEST AT G-IF, thereby supplying some of the needed information ahead of time and skipping to a function lower in the MAKE-TEST hierarchy.

In addition to accessing functions directly, we discovered that the language interface permits the users to access variable values of the system directly. If the user wishes to know WHAT UNIT IS LOADED?, InterFIS returns the value of the corresponding variable, even though no mnemonic command or FIS function exist to access that information. This consequence of similarities between Lisp's treatment of variables and InterFIS's treatment of functions was most desirable.

Extending the capabilities of an expert system as discussed above raises the obvious questions of how much of the expert system should be accessed that wasn't originally accessed and how much the existing system should be modified. All domain-relevant information is stored somewhere in the expert system. Natural language interfaces supply the user with a virtually unrestricted medium for requesting this information. Where is the line drawn?

We have answered this question by not extending the natural language capabilities of the interface beyond those commands the expert system was already capable of processing. Thus, yes/no questions and certain types of interrogatives are not processed, as noted above. On the other hand, we permitted the natural language interface to access information if it was readily accessible from the expert-system knowledge base, even in the absence of a specific expert-system function. For example, a user can now ask the expert system what unit is being tested. We did not believe that we were altering the system's functionality by so doing.

Researchers involved in natural language processing and interfacing to expert systems will be confronted with the problem of modifying or adding to the expert-system code. We believe that the writer of the natural language interface should preserve the integrity of the system and should not alter the code of the system to which he or she is interfacing. However, the question of enhancements to the expert system will arise. The authors of the expert system may not have incorporated some function that accesses information that is easily requested through the richness of natural language. InterFIS enhances the interface whenever a natural language sentence has a corresponding representation in the expert system, and the expert system does not have a function to present that information. This is not in any way an alteration of the expert system code. It simply allows the system to present what is already represented in the expert system's knowledge base.

We also capitalized on the natural language interface's ability to stack commands when information is compressed into one input sentence "(18) Sample Sentence". Such stacking simply permits the user of the natural language interface to step through the hierarchy of FIS-commands in a more natural manner. These additions, therefore, do not alter the expert system's functionality but merely extend the system's capabilities by incorporating natural language communication.

Representation

Throughout this report we have referred to COIN, the Command-Oriented INterpreter. The name of this module introduces one final issue that must be discussed. In surveying natural language interfaces to database management and expert systems [10 - 12], we note that there are essentially two types of system organization. Generally, database search systems are organized in a function/argument schema [13]. Verbs or actions in interfaces to these systems correspond almost directly to function calls, while data objects correspond to arguments of those functions. Systems so organized permit a one-to-one mapping between natural language verbs and objects to database functions and arguments.

Many other expert systems are organized around production rules [14]. FIS, a variant of the production rule system, was organized around "whole" predicates, making it similar to frame-based representation expert systems [14]. To illustrate, in FIS, the verb SHOW and the object noun phrase ACTIVE LIST do not correspond directly to individual functions and data objects. The predicate SHOW THE ACTIVE LIST, however, does directly correspond to the function SHOW-ACTIVE-LIST. Because of this, our interface had to map natural language input not to actions and objects that correspond to predicates and their logical arguments, but to specific FIS function calls or commands (thus the name: COIN or Command-Oriented INterpreter).

FUTURE WORK

We have not decided which expert-system organization is preferable for incorporation into a natural language interface that uses predicate-argument representations. Nor have we decided how intrinsic the differences are to the database/expert-system paradigm in general. In the future, we plan to consider alternate methods of meaning representation that may make any assumed distinctions debatable. Such representation would involve mapping all language input to a metalanguage (mentalese or the language of thought). System-function calls and corresponding arguments would also be represented in this language, and an alternative mapping strategy would be employed.

In the future we will incorporate a discourse component to the interface and investigate a user model for this expert system. Other elements not handled by sentence grammars, such as pronominal reference across sentences, need to be included to process all the information. These additions would greatly expand the capability of the system. In a totally natural language interchange, communication is not only a function of syntactic and semantic sentential processing. For example, in the following discourse "(19) Sample Discourse", sentential processing cannot resolve the pronoun reference across sentences.

(19) — Sample Discourse

user: Show the history list .

expert system: [an appropriate list is displayed on the monitor]

user: Show it again .

Currently, InterFIS cannot process the discourse in "(19) Sample Discourse" because it does not have a discourse component that can resolve pronominal reference. Future work, therefore, should incorporate such a component.

Interaction with an expert system can be greatly facilitated by multimodal interaction as well as by incorporating speech input. Our research with InterFIS has been in adding natural language capabilities to an expert system that already had a mnemonic menu-driven interface and a mouse-oriented interface. InterFIS is currently limited; the input must be typed at a keyboard. Therefore, the user of the interface is manually limited in doing other tasks. Voice input will free the hands of the user and let the user interact more efficiently with the system and with the work environment in general.

Furthermore, keyboard entry requires more time, since most people have different typing skills. A voice activated system could reduce the additional time required for input. Internal processing of the speech signal may, of course, be time consuming, but we assume that research in this technology will address that issue separately. It is simply our desire to provide a voice activated system that provides greater flexibility for the user in enhanced multimodal interactions with the expert system.

Some additional linguistic coverage is also required in future work with InterFIS. As noted earlier, yes/no questions are currently not processed or interpreted in InterFIS. Future work should address these constructions and process them. Relative clauses, such as WHICH WERE RUN in "(20) Sample Sentence", are processed in the grammar, and the PFQAS interpreter maps the embedded verb into an adjectival modifier.

(20) — Sample Sentence

Show the tests that were run.

Since a network connection is in COIN between TESTS and RUN, "(20) Sample Sentence" is interpreted. However, this avoids the linguistic issue of whether or not all relative constructions should be interpreted in this way. Although this method works in our limited domain, it is not obvious that it will be applicable in other domains. Development of a more robust, transportable system will force us to reconsider our strategy. As part of this effort, we will incorporate a new selectional interpreter [15] as one of the modules in InterFIS.

CONCLUSION

In this report, we have described the function of InterFIS, a natural language interface to an expert system, the Fault Isolation Shell. We have described how natural language input to the expert system is first parsed syntactically to obtain an intermediate, domain-independent representation. This representation is then mapped to verb-models within the specific domain of electronics troubleshooting, and the representation is matched to the existing commands within that expert system. As a result, typed input to InterFIS results in a series of electronic tests in FIS. We have discussed several research issues about natural language processing and interfacing to an expert system, as for example altering a system's functionality. We have explained what motivated our grammatical coverage. Finally, we have outlined our future research goals for InterFIS, mainly how we would extend the grammatical coverage by adding speech to the interface and by incorporating a different selectional interpreter.

ACKNOWLEDGMENTS

This research was supported by the Office of the Chief of Naval Research under Natural Language Research project N00014-90-WX4B738. The authors extend their thanks to Ken Wauchope for providing the PROTEUS Workstation Interface to adapt and develop grammars. We also thank William Spears for setting up and explaining the Fault Isolation Shell to us, Stephanie Everett, Elaine Marsh, Astrid Schmidt-Nielson, and Ken Wauchope for their critical comments in the preparation of this report. The authors, however, assume all responsibility for its contents.

REFERENCES

1. F. J. Pipitone, K. Dejong, W. Spears, and M. Marrone, "The FIS Electronics Troubleshooting Project," in *Expert Systems Applications to Telecommunications*, J. Liebowitz, ed. (Wiley and Sons, New York, 1988), pp. 73-101.
2. F.J. Pipitone, K.A. Dejong, and W.M. Spears, "An Artificial Intelligence Approach to Analog Systems Diagnosis," NRL Report 9219, 1989.
3. R. Schoeffel, "FIS Ergonomic Interface Users Guide," manual, Naval Research Laboratory, Washington, D.C., 1988.
4. R. Grishman, "PROTEUS Parser Reference Manual," PROTEUS Project Memorandum #4, New York University, New York, 1986.
5. J.M. Gawron, "Syntactic Regularization in PROTEUS," PROTEUS Project Memorandum #5, New York University, New York, 1986.

6. N. Sager, *Natural Language Information Processing: A Computer Grammar of English and Its Applications* (Addison-Wesley, Reading, MA, 1981).
7. D. Perzanowski and E. Marsh, "Preparing a Sublanguage Grammar," NRL Report (forthcoming).
8. G. Gazdar, "Phrase Structure Grammars," in *The Nature of Syntactic Representation*, P. Jacobson and G.K. Pullum, eds. (D. Reidel, New York, 1983), pp. 131-186.
9. G. Gazdar and G. Pullum, "Generalized Phrase Structure Grammar: A Theoretical Synopsis," Indiana University Linguistics Club, Bloomington, IN, 1982.
10. W. C. Ogden, "Using Natural Language Interfaces," in *Handbook of Human-Computer Interaction*, M. Helander, ed. (North-Holland, New York, 1988), Ch. 13, pp. 281-299.
11. R. J. H. Scha, "Natural Language Interface Systems," in *Handbook of Human-Computer Interaction*, M. Helander, ed. (North-Holland, New York, 1988), Ch. 44, pp. 941-956.
12. M. H. Chignell and P. A. Hancock, "Intelligent Interface Design," in *Handbook of Human-Computer Interaction*, M. Helander, ed. (North-Holland, New York, 1988), Ch. 46, pp. 969-995.
13. G. Widerhold, *Database Design* (McGraw-Hill, New York, 1977).
14. D.A. Waterman, *A Guide to Expert Systems* (Addison-Wesley, Reading, MA, 1986).
15. K. Wauchope, "A Tandem Semantic Interpreter for Incremental Parse Selection," NRL Report 9288, September 28, 1990.