A NATURAL LANGUAGE INTERFACE FOR EXPERT SYSTEM HELP DESKS

Candy F. Drew
IBM Corporation
Room 9042, MS 2091
6903 Rockledge Drive
Bethesda, MD 20817

ABSTRACT.
Many people today are coding menu-driven systems using expert system technology. This is particularly true in the expert system help desk[1] arena. There is nothing wrong with using expert systems technology for this purpose; expert system shells are merely one more step up the ladder of tools enabling more people to write application programs. However, using expert system objects, it is possible to create a much richer user interface. This paper will address how a natural language interface might be written entirely within IBM's Expert System Environment Shell (ESE) [2]. The interface is designed around the help desk application, a diagnostic knowledge base. The implementation uses techniques I've named FASTPATH, INHERIT, and CHGPATH[3].

FASTPATH --WHY IT'S NEEDED.
In the help desk environment, many callers want and expect immediate answers to their problems. Time is critical. Applications for the help desk which fail to take this into consideration may only further complicate the environment.

Time is spent in a variety of ways which may or may not be obvious. Each time the ENTER key or a Function key is pressed, the operator deals with a system response delay. Each time a screen appears before the operator, there is a delay while the operator reads and interprets the screen. Each keystroke the operator makes results in a delay. These delays must be minimized.

Menu-Driven Systems.
Menu-Driven Systems are incorporated into most applications today. Those menus may consume full screens, action bars, pull-down windows or hypertext. Menus have proven to be powerful means for navigating through information. For that reason, it will be some time before menu-driven systems are ever fully replaced. However, menu-driven systems have a few drawbacks which can become intensified in the time-driven help desk environment. Three of the most serious drawbacks are identified below:

1. Too Many Levels of Menus. How many levels can you traverse before frustrating or losing your user? In attempting to limit the number of menu levels, designers may create overly general, wordy menu items or lengthy menus with several choices per menu.

2. Lengthy Menus. If the user's selection is at the bottom of a long menu, the user is delayed by the keystrokes necessary to reach that choice. If the menu is so long that it is carried over to another page, the user is also delayed by the system response time to scroll to that page. Using a mouse makes this process somewhat faster than using arrow keys. Using hypertext enables you to embed lengthy menus in streams of text. However, unless the user is so familiar with the menus that it is unnecessary to read the text, another type of delay occurs.

3. Read/Interpret/Respond Delays. How do you word menu choices for the novice or infrequent user? Designers have found this to be a no-win situation. No matter how you word the choices, someone will fail to understand the menu. Inevitably, a user will select a wrong choice and end up on the wrong path (and you must design around this inevitability). But suppose your user is eventually able to understand the menu items; you should consider how much time was taken to reach that understanding. There is going to be a delay based upon the user's reading speed and a delay based on the time it takes the user to determine which menu item is most suitable.

Besides ensuring that the menu's options can be understood by its intended audience, the designer must also consider how long it may take someone to figure out how to respond to the menu. The absence of standards has lead to a wide range of ways to respond to menus. Some menus permit only one selection per menu. Some menus permit several selections per menu. Menu responses may also be made in a variety of places: selections may be made by entering mnemonics on command lines, selections may be made by positioning the cursor on a menu option and pressing the ENTER key, and selections may be made by entering characters or numbers beside the desired options. There are even menus which permit you to press the ENTER key without responding (this is supposed to signify that you don't know and are therefore willing to accept a default choice). In the applications, you can choose every one of these methods! It takes time for the...
Consider this scenario: A person calls the help desk and describes a problem in its entirety. The operator, using an automated help desk application, reads a number of menus and due to system response time, the depth of menus the operator had to search, the amount of reading and interpreting, and the variety of responses required, the operator forgets some or all of the problem description. The caller has been waiting on the phone all this time hoping a solution to the problem was being found. The operator's voice comes back on the phone, "I'm sorry. Could you please repeat your problem?". This is one reason why you'll see notepads beside help desk terminals.

With FASTPATH, the scenario changes. Once again, a person calls the help desk and describes a problem. This time, before having a chance to forget what the person said, the operator types the problem in on the first screen of the help desk application. The next screen which appears describes the solution to the problem. Now, when the operator's voice comes back over the phone, the caller gets a solution.

If a problem description must be recorded in a problem tracking system such as The Information Management[4] database, the operator has already provided the system with a description which the system may automatically enter into a database or problem tracking system. Not only does this alleviate the need for operators to use notepads, it also reduces the need for operators to be skilled at recording problems in databases or problem tracking systems.

**A Sample MENU-Driven Session**

Other natural language interfaces may have you design a backup system as an afterthought -- and usually only in piece-meal, when ambiguity is detected. FASTPATH requires you to design the backup system first. The menu-structure is FASTPATH's built-in backup system. This backup is needed not only to handle ambiguity but also to handle cases when the user enters incomplete problem descriptions or words which, though critical to understanding the problem, may not be in the system's vocabulary.

In this sample menu-driven session, a caller has asked the help desk operator how to transfer a call. The application begins with a menu of SYSTEMS which corresponds to an E.S.E. parameter called SYSTEMS:

```
What system is the caller using?
    _MVS
    _Phone
    _PC/DOS
    _VM/CMS
```

The operator selects 'Phone'. The next screen is a menu of PHONE_ISSUES:

```
What phone issue does this concern?
    _Phone Problems
    _Change in Service
    _Functions
```

The operator selects the 'Functions' option. The next screen which appears is a menu, PHONE_FUNCTION:

```
What function does this concern?
    _TRANSFER
    _Hold
```

The operator selects 'TRANSFER'. The next screen displays the ANSWER:

```
+--------------------------------------------------+
| Press the TRANSFER button then dial the four-     |
| digit extension. Let the person who answers at   |
| that extension know you are transferring a call, |
| then hang up.                                   |
+--------------------------------------------------+
```

**A Sample FASTPATH Session**

The FASTPATH version of that same session begins by asking the operator to enter a PROBLEM_DESCRIPTION:

```
Please describe the problem:
```

The operator might type:

```
Candy Drew wants to know how to use the phone's xfer funct.
```

The next screen would be the ANSWER:

```
+--------------------------------------------------+
| Press the TRANSFER button then dial the four-     |
| digit extension. Let the person who answers at   |
| that extension know you are transferring a call, |
| then hang up.                                   |
+--------------------------------------------------+
```

**THE FASTPATH IMPLEMENTATION**

FASTPATH is implemented in E.S.E. using, primarily, six knowledge objects:

- One menu parameter per menu
- A synonym parameter per menu option
- A parsing parameter per synonym parameter
- One problem description string parameter
- A parsing rule
- A rule driving the menu stream

**MENU PARAMETER DEFINITION**

Menus are designed in ESE by creating parameters with TAKEN FROM constraints. The menu parameters in the phone transfer example are defined as below:

```
*** the SYSTEMS menu parameter ***
taken from ('MVS', 'Phone', 'PC/DOS', 'VM/CMS')

*** the PHONE_ISSUE menu parameter ***
taken from ('Phone Problems', 'Change in Service', 'Functions')

* the PHONE_FUNCTION menu parameter *
taken from ('TRANSFER', 'CAMP', 'HOLD')
```
SIMPLE SYNONYM DEFINITION

One of the synonyms the knowledge engineer would create for this example would be a synonym for TRANSFER. The synonym parameter for TRANSFER could be defined like this:

```plaintext
*** TRANSFER SYNONYM ***
= ('transfer', 'xfer', 'send')
```

Unfortunately defining synonym parameters in this manner sets them up as constants, and no inferencing is required to determine the value of constants. The net effect is that you end up carrying a bulky thesaurus in memory throughout each consultation. To make the situation even worse, the expert system will then proceed to scan the entire "thesaurus" for each potential menu in the consultation. If the menu rule, currently being reviewed by the system, has six menu clauses in the IF-portion of the rule, the system may potentially scan the entire thesaurus from 'A' to 'Z', six times. This can have a drastic impact on performance.

COMPLEX SYNONYM DEFINITION

There is an alternative way to define the synonym parameter within E.S.E. which yields better system performance. This methodology has the effect of creating a menu-specific thesaurus for each menu, reading into memory only those synonyms relevant to the menu currently being investigated. It's important to understand and try to implement the prior methodology first, though, because this methodology is a lot less straightforward and requires that your expert system shell have something similar to E.S.E.'s sourcing sequence and E.S.E.'s mechanism for grouping rules.

You begin by defining the synonym parameter like this:

```plaintext
*** TRANSFER SYNONYM ***
= (string; multivalued)
```

Defining this parameter as a multivalued string is like saying that the word 'transfer' has more than one synonym. In a third generation language, this is the equivalent of declaring TRANSFER SYNONYM as an array of unspecified length.

Next you define the Default Constraint property of TRANSFER SYNONYM as follows:

```plaintext
*** Default Constraint ***
= ('transfer', 'xfer', 'send')
```

Then you must edit the Sourcing Sequence property of the TRANSFER SYNONYM by deleting all sources except the Default Constraint. In doing this, you trick the system into believing it doesn't know the multiple values of TRANSFER SYNONYM until the moment when the inference engine needs to determine if TRANSFER SYNONYM has any known values. At that time, the system will look at the sourcing sequence and discover that it must accept the Default Constraint.

Next you would place the rules which reference TRANSFER SYNONYM within a Group along with the rules which reference the synonym parameters for CAMP and HOLD (the other options on that particular menu). You would also modify the sourcing sequence of the menu parameter PHONE_FUNCTION so that it knows to check this Group of synonym rules before asking the user to select from the menu.

Both the simple synonym definition and the complex synonym definition enable you to maintain your entire application within the expert system shell. This was the aim established for this paper. However, unless your user has a small vocabulary, your system will perform a lot better if you maintain your synonyms outside of the shell in data files, databases, or third generation language structures such as arrays.

PARSING PARAMETER DEFINITION

Continuing with the FASTPATH implementation, you would next define a parsing parameter:

```plaintext
*** FOUNDTRANSFER ***
= location of (TRANSFER_SYNONYM within PROBLEM_DESCRIPTION)
```

PROBLEM DESCRIPTION is defined as a string:

```plaintext
*** PROBLEM DESCRIPTION ***
= a string
```

which you ask the user to enter on the first screen.

PARSING RULE DEFINITION

The parsing rule is defined as:

```plaintext
*** RULE: FOUNDTRANSFER ***
If FOUNDTRANSFER > 0
Then PHONE_FUNCTION = 'TRANSFER'
```

The parsing rule can actually be defined like this:

```plaintext
*** RULE: FOUNDTRANSFER ***
If location of (TRANSFER SYNONYM within PROBLEM DESCRIPTION) > 0 Then PHONE_FUNCTION = 'TRANSFER'
```

Coding the parsing rule in this manner means you don't have to have a parsing parameter. The less parameters you have in your system the less storage your system takes and that also has an effect on performance. My reason for having parsing parameters becomes obvious when I have compound words requiring more than one clause in the IF portion of my rule in order to clarify the meaning:

```plaintext
*** RULE: FOUNDMAILMAN ***
If location of (MAIL_SYNONYM within PROBLEM_DESCRIPTION) > 0 and location of (MAIL_SYNONYM within PROBLEM_DESCRIPTION) < location of (MAN_SYNONYM within PROBLEM_DESCRIPTION) Then Word = "Mailman NOT man mail"
```

You would typically have lots of rules in your system which use two or more synonyms in a variety of combinations with a variety of meanings. The example above helps distinguish between "The mail
man" and "The man [who may] mail". The rule definition above entails a lot more typing than:

*** RULE: FOUND MAILMAN2 ***
If FOUND MAIL > 0 and FOUND MAIL < FOUND MAN
Then Word = "Mailman NOT man mail"

MENU STREAM RULE DEFINITION
The rule driving the menu stream is the same for both the menu-driven system and for FASTPATH. In the sample session, the rule is defined like this:

*** RULE: MENU STREAM TRANSFER ***
If SYSTEMS='Phone' & PHONE ISSUE='Functions' & PHONE FUNCTION='TRANSFER'
Then Answer =

+--------------------------------------------------+
| Press the TRANSFER button then dial the four- | |
| digit extension. Let the person who answers at | |
| that extension know you are transferring a call,| |
| then hang up.                                |
+--------------------------------------------------+

Now suppose FASTPATH fails? It would be nice if we could rely upon the operators and callers to provide the system with consistently good (usable) PROBLEM DESCRIPTIONS, using the words in the synonym parameters, and supplying us with all the words necessary to eliminate all of the menus. Unfortunately, this is not always possible. In those cases, when the FASTPATH system finds the PROBLEM DESCRIPTION is lacking, it falls back on the menu-driven system. In our prior example, the operator might have entered this:

PROBLEM DESCRIPTION:
Caller wants to know how to use TRANSFER function.

This time the operator failed to type in which system the caller was trying to use. FASTPATH requires you to give it a synonym for each menu in the path to your solution, otherwise FASTPATH shows you the menus for which you failed to give it synonyms. Given the PROBLEM DESCRIPTION above, the next screen in FASTPATH would be the SYSTEMS menu:

What system is the caller using?
- MVS
- Phone
- PC/DOS
- VM/CMS

If the operator selects the 'Phone' option, the next screen is then the ANSWER screen:

+--------------------------------------------------+
| Press the TRANSFER button then dial the four- | |
| digit extension. Let the person who answers at | |
| that extension know you are transferring a call,| |
| then hang up.                                |
+--------------------------------------------------+

In the worst case, when there are no synonyms in the problem description (which are known to the system) FASTPATH becomes a menu-driven system. If there is just one synonym, you eliminate just one menu in your path. If there are two synonyms, you eliminate two menus and so on. If a path has five menus and you eliminate your first and third menu, you'll be given the second menu and then the fourth menu and then the fifth menu. If you give synonyms for the third and fourth menus, you'll be given the first menu and then the second menu and then the fifth menu. Your menu path preserves its order; FASTPATH simply skips over certain menus if it finds synonyms for them in the problem description.

FASTPATH Addresses Menu-Driven System Challenges
- FASTPATH can keep long paths of menus invisible to the user.
- The wording of menu options effects less users because they may never have to see certain menus.
- The Read/Interpret/Respond delay may be greatly reduced. In the best case, the operator never has to read, interpret or respond to a menu.
- Tabbing and scrolling may be greatly reduced or eliminated. In some cases it may even take less keystrokes to enter a description than it would take to select options using the arrow keys.
- FASTPATH provides operators with a mechanism for eliminating notepads at the helpdesk. And it can use that same mechanism to eliminate the need for operators to record problems in problem tracking databases. That is, the application (instead of the operator) can record problems in the database.

Considerations/Tradeoffs:
Synonyms must be well thought out. FASTPATH designers must consider the impact of a user adding affixes to a synonym. Designers must disambiguate between the roles a word may serve in a domain. For instance, "mail man" and "man mailed" would be the same thing to FASTPATH if your design only checks for 'existence' of synonyms. You would therefore have to check for word order or other syntactic clues to help narrow down the meaning.

It is also a good idea to plan for common misspellings and include them in the synonyms. There is little need, however, for the developer to include all forms of a word in a synonym list. If the inflections of a word (-s, -ed, -ing, etc.) are regular, the base form is enough unless there are other words (in the domain of describing your system's problems) which share the base form but have entirely different meanings. In such cases, the word (or base form) could be given two synonym parameters, one for each sense of the word (just like in a dictionary). The parsing rules referencing such words would then probably need more than one clause in the IF portion to disambiguate between the senses.

Consider maintaining synonyms in external files or databases. The synonyms which occur to the knowledge engineer will almost certainly be different from the synonyms which occur to the operators. There is also the possibility that the knowledge base might require national language support. If synonyms are designed to be acquired from external thesauruses,
it might make it easier for others to update the synonyms or translate them into other languages. It would certainly be easier on the expert system load.

The FASTPATH operator rarely needs to memorize each menu path. This is one of the biggest advantages of implementing FASTPATH. It is hard to forget the help desk operator who had created a very large menu-driven help desk system. All potential callers were supposed to use the system prior to calling the help desk. The operator complained that most of the calls to the help desk were unnecessary because the answers were in the automated system. The operator was asked to give an example of the most frequent call and to show the proper menu path and its answer in the system. After five minutes of searching through menus, the operator (who, as you may recall, developed this system) became flustered and asked us just to "take it on faith"; the answer was in the system but not even the developer could find it. This is an extremely common menu design problem.

Another consideration to be made is what to call this enhancement once you've implemented it. Because FASTPATH contains synonym lists and rules enabling it to semantically parse problem descriptions, such as the one above, it might be considered a "natural language" query system. Many computational linguists would find fault with such nomenclature. They would argue that to have a "natural language" system, you must first 'syntactically' parse sentences into parts of speech (nouns, verbs, etc.) and sentence components (subject, complement, etc.) in order to have true machine understanding. They would prefer that we classify systems which don't have syntactic parsers as keyword or subset computer language systems.

The reality is that all machine-based 'natural' language systems are doomed to be subset languages, particularly when the language to be captured is English. English is so vast and creative that there are an infinite number of words and sentence structures.

Because FASTPATH imposes no limits on sentence structure, the only thing limiting sentence structure might be the amount of storage required to house the structure while it is being parsed. Yet many of the "natural" language systems confine users to Standard-English sentence structure obeying Standard-English grammar rules. Even with those restrictions, there are still an infinite number of sentence structures possible which designers do not have an infinite amount of time to encode. So systems relying upon syntactic parsers are further restricted to understanding only those structures the designers have time to program. An argument could be made that systems such as FASTPATH, which impose no limit on sentence structure, enable users to communicate more "naturally" than systems with syntactic parsers.

Even though FASTPATH does not pre-program and thus limit sentence structure, the FASTPATH system is able to use word order and other aspects of syntax when additional clues are needed to determine meaning. Unlike pure keyword searches, FASTPATH can go beyond simply recognizing the existence of a word in an input stream; it can also determine the location of the word within the input. This additional information enables the system designer to create rules to narrow down the possible meanings of an ambiguous input stream.1

FASTPATH is a method which attempts to extract meaning from sentences (semantically parsing a sentence) first. It only resorts to syntactic parsing when syntax may help to resolve ambiguity.

In other words, FASTPATH parses in reverse of 'linguistic' natural language parsers; this can be a far more efficient mechanism for natural language understanding. In an input stream such as "Candy Drew wants to know how to use the phone's xfer funct.", many natural language systems might have begun by trying to classify my name as a NOUN followed by a VERB. In the absence of common sense or capitalization rules, they might settle on a meaning which would suggest that a "sweet confection wrote something". Next they might parse "wants" as a verb and realize that a verb does not follow a verb unless you're handling auxiliary verbs. So they would have to go back to the beginning of the sentence and parse for different senses of "Candy" or "Drew". Eventually, "Drew" might be re-parsed to another sense of the VERB, draw, meaning "pulled or attracted". The word "wants" could then parse into a deverbalized plural NOUN, as in "He made his wants known."

Even armed with an encyclopedia of knowledge and a dictionary, such systems might (after a whole lot of processing) most likely end up with the idea that "a sweet confection attracts desires in order to learn how to use a phone's transfer function". A few natural language parsers might take capitalization into consideration and might recognize my name as two proper nouns, but most such systems have a pre-set database of names they recognize; neither my first nor last name is usually on such databases. Therefore these systems might either give up parsing the sentence after the first two words, assume that proper nouns don't require further parsing and continue to burn cycles trying to parse through every word in the sentence, or treat the capitalizations as a possible typo and end up parsing the sentence just like systems which don't take capitalization into consideration. Eventually such systems might yield an answer; much more likely is that they will yield the canned response, "I don't know. I don't understand."

In this particular help desk domain, FASTPATH would have viewed "Candy Drew wants to know how to use the" as noise, and would not have parsed any of it, saving the system a substantial amount of processing time. FASTPATH would focus on "phone", "xfer" and "funct." because first the expert system is searching for a SYSTEM. If the SYSTEM turns out to

---

1 The method for handling this was shown earlier in a rule which distinguished between "mail man" and "man mail" using word order.
be the "phone" SYSTEM, then it's searching for a PHONE ISSUE. If the PHONE ISSUE turns out to be "functions", then it's searching for a FUNCTION. Consequently, FASTPATH would take the user down a path which contains everything the system knows about the phone's transfer function. In this case, all the system knows is how to use the function; so FASTPATH would immediately display an answer screen.

INHERIT -- A Faster FASTPATH

FASTPATH allows the user to take better advantage of expert systems technology than menu-driven systems, but fails sometimes to take full advantage of artificial intelligence. Given one synonym in a problem description, such as "xfer", FASTPATH may infer that PHONE_FUNCTION = 'TRANSFER'. If this is the only synonym FASTPATH can retrieve from the problem description, it will ask the operator to choose a menu option. This system, however, will not always ask the user to choose a menu option, as this is just giving the user a chance to make the wrong choices from these menus and end up going down a completely different path that has nothing to do with the problem described.

If the system has determined a value for a child menu, and the child menu is descended from only one parent menu option, the expert system should be able to infer the value of the parent menu. That is the main premise of the INHERIT strategy; a child should know who his parent is.

***Note Of course there are cases, where a child menu might belong to several parents. In such cases, knowing the child menu does not mean knowing the parent menu and so the parent menu would still have to be asked. If your expert system shell has dynamic menu capabilities, though, at least you can remove menu options which could not possibly be parents. --And even if there is no way to eliminate the parent menu or some of the parent menu choices, you may be able to eliminate or reduce the grandparent menus.

Recall the PHONE ISSUE menu:

*** the PHONE ISSUE menu parameter ***

taken from ('Phone Problems',
'Change in Service',
'Functions')

If we know that we might have questions on both the usage of the phone's Transfer function and the problems with the phone's Transfer function, we would design our expert system so that the PHONE_FUNCTION menu is the child menu of both 'Phone Problems' and 'Functions'.

In such a system, the PHONE ISSUE parent menu would still have to be asked in order to resolve which of the two parents is involved in questions regarding phone TRANSFERS. However, the grandparent menu, SYSTEMS, would not have to be asked unless the Transfer Function was something common to more than one system.

How does INHERIT improve FASTPATH?

o INHERIT cuts down on typing. Rather than typing a long description, the user may only need to type low-level keyword/synonym(s).

o INHERIT eliminates the potential for the user to choose the wrong parent and go down the wrong path after having already properly identified a child unique to one particular parent menu option.

o INHERIT is more forgiving about typos that aren't pre-programmed; sometimes it can "fill in the blanks".

o INHERIT makes your system appear more intelligent. It permits the system to make the same assumptions a human expert might make and has the user explain (or disambiguate) the same things the human expert would need explained.

The INHERIT Implementation

The INHERIT strategy can be added right into the existing parsing rules. For example:

** INHERIT'S RULE: FOUND_TRANSFER **

If FOUND_TRANSFER > 0

Then PHONE_FUNCTION='TRANSFER',
PHONE_ISSUE='Functions',
SYSTEMS='Phone'

If PHONE_FUNCTION is a menu which belongs to two or more PHONE_ISSUES, you can still use INHERIT to eliminate the grandparent SYSTEMS menu:

** INHERIT'S RULE: FOUND_TRANSFER2 **

If FOUND_TRANSFER > 0

Then PHONE_FUNCTION='TRANSFER',
SYSTEMS='Phone'

Actually 'Transfer' is a very popular function. You can transfer phone calls; you can transfer files between MVS users; you can transfer files from the PC to VM, etc. That is, the Transfer function may not only not know its parent, it might not know its grandparent either. In such cases, INHERIT gets you no further than FASTPATH. So what you end up with is actually a FASTPATH rule. However, you might at least let all the potential parents know that they may be "giving birth" by coding a rule like this:

** FASTPATH RULE: FOUND_TRANSFER **

If FOUND_TRANSFER > 0

Then PHONE_FUNCTION='TRANSFER',
MVS_FUNCTION='TRANSFER',
VM_FUNCTION='TRANSFER',
PCDOS_FUNCTION='TRANSFER'

214
CHGPATH - Smooth Navigation

From time to time the operator will still go down the wrong path regardless of whether you use FASTPATH, INHERIT or menus. The application designer should provide ways for the operator to change paths quickly and efficiently when this happens. Before implementing CHGPATH, we had two time-consuming ways to accomplish this:

- Restart the expert system
- Back up (UNDO) one screen at a time

The CHGPATH Implementation.
To implement CHGPATH in E.S.E., you create inner and outer loops using the Max Instances properties of FCBs and set the DisposeWhenDone properties to TRUE. The outer loop gives the operator unlimited opportunities to query the knowledge base without restarting the system. The inner loop gives the caller unlimited opportunities to describe problems. You add a synonym parameter to recognize CHGPATH if an operator enters CHGPATH in the PROBLEM_DESCRIPTION. You design your screens so that the operator will always have the option of entering a problem description, selecting from a menu, or both. You also code rules to prematurely end inner FCB loops:

```plaintext
**** CHGPATH RULE: END_FCB_LOOP ***
If FOUND_CHGPATH > 0
Then dont pursue another name_of_FCB
```

CONCLUSION

FASTPATH, CHGPATH and INHERIT are three methodologies which have been developed to provide the expert system help desk application with a natural language interface. These methodologies do not follow the standard computational linguistic parsing approach. This does not mean that the language mechanism for FASTPATH is any less "natural" than computational linguistic approaches. In fact, it can be argued that the FASTPATH methodology allows for a more "natural" language interface than systems which confine sentences to those structures someone has pre-programmed. It would be preferable, though, to simply assert that FASTPATH is one additional methodology for coding natural language interfaces; INHERIT is a methodology for getting expert systems to make assumptions during a dialog just like experts do; and CHGPATH is a usability feature which acknowledges that human systems (which includes knowledge engineers and domain experts, not just end users) make mistakes for which the end user should not be unduly penalized.

REFERENCES


