AN ECLECTIC APPROACH TO BUILDING NATURAL LANGUAGE INTERFACES

Brian Phillips, Michael J. Freiling, James H. Alexander, Steven L. Messick, Steve Rehfuss, Sheldon Nicholl†

Tektronix, Inc.
P.O. Box 500, M/S 50-662
Beaverton, OR 97077

ABSTRACT

INKA is a natural language interface to facilitate knowledge acquisition during expert system development for electronic instrument trouble-shooting. The expert system design methodology develops a domain definition, called GLIB, in the form of a semantic grammar. This grammar format enables GLIB to be used with the INGLISH interface, which constrains users to create statements within a subset of English. Incremental parsing in INGLISH allows immediate remedial information to be generated if a user deviates from the sublanguage. Sentences are translated into production rules using the methodology of lexical-functional grammar. The system is written in Smalltalk and, in INKA, produces rules for a Prolog inference engine.

INTRODUCTION

The ideal natural language interface would let any user, without any prior training, interact with a computer. Such an interface would be useful in the knowledge acquisition phase of expert system development where the diagnostic knowledge of a skilled practitioner has to be elicited. As technicians are not familiar with formal knowledge representation schemes, a trained intermediary, a *knowledge engineer*, is generally employed to handcraft the internal format. This process is time-consuming and expensive.

INKA (INglish Knowledge Acquisition) permits task experts to express their knowledge in a subset of English, and have it automatically translated into the appropriate representational formalism. In particular, the version of INKA to be discussed here accepts input in a sublanguage called GLIB which permits the statement of facts and rules relevant to the troubleshooting of electronic systems (Freiling et al., 1984), and translates these statements into Prolog unit clauses for later processing by a specialized inference mechanism. Experiments with INKA to date have enabled us to construct sufficient troubleshooting rules to build a localizing troubleshooter for a simple circuit.

INKA is designed as one of the tools of DETEKTR, an environment for building knowledge based electronic instrument troubleshooters (Freiling & Alexander, 1984). DETEKTR supports an expert system development methodology which is outlined below. The design goal of INKA is that it serve as a natural language input system to facilitate transfer of knowledge during the knowledge acquisition phase of expert system development. INKA is not intended to stand alone as the sole mechanism for knowledge transfer, but to be supported by components capable of managing a coherent dialogue with the task expert. McKeown (1984) has pointed out a number of important aspects of the pragmatics that relate to the *usage phase* of an expert system. Similar pragmatics are required to insure adequate construction of the system's knowledge base during the *knowledge acquisition phase* of an expert system's development. The most important pragmatic facility is one to estimate the degree of *epistemic coverage* of the knowledge acquired so far, and to prompt the task expert for more knowledge in areas where the coverage is weak. It is unfeasible to assume that any task expert can simply perform a "memory dump" of expertise into some natural language interface and be done with it.

This paper discusses the natural language technology used in building INKA. The system incorporates a diverse collection of natural language technologies in its construction. Specifically, INKA utilizes a *semantic grammar* (Burton, 1976) to characterize the domain sublanguage, *lexical-functional semantics* (Kaplan & Bresnan, 1982) to translate to the internal form of representation, and an interface that includes left-corner parsing with in-line guidance to address the linguistic coverage problem that arises with sublanguages. We feel this eclectic approach is a useful for building application-oriented natural language interfaces. Although we are describing a knowledge acquisition application, the methodology can be used for any application whose sublanguage can be stated in the prescribed grammar formalism.

Tereisias (Davis, 1977) provides a natural language environment for debugging a knowledge base. INKA at present contains no facilities to modify an existing rule or to test the evolving knowledge base for some level of integrity; these are to be future additions.

INKA is written in Smalltalk (Goldberg & Robson, 1983) and runs on both the Tektronix Magnolia Workstation and the 4404 Artificial Intelligence System. INKA makes extensive use of the bit-mapped display and three-button mouse on these systems.

LANGUAGE AS A KNOWLEDGE ENGINEERING TOOL

The major bottlenecks in building knowledge based systems have proven to be related to the definition and acquisition of knowledge to be processed.

The first bottleneck occurs in the *knowledge definition phase* of system development, where symbolic structures are defined that represent the knowledge necessary to accomplish a particular task. A bottleneck arises because of the shortage of knowledge engineers, who are skilled in defining these structures and using them to express relevant knowledge.

† A summer intern at Tektronix, currently at the University of Illinois, Champaign-Urbana.

The second bottleneck occurs in the *knowledge acquisition phase*, which involves the codification of the knowledge necessary for a system to function correctly. A bottleneck arises here because in current practice, the presence of the knowledge engineer is required throughout this time-consuming process.

In the course of defining a viable methodology for the construction of expert systems (Freiling & Alexander 1984; Alexander et al. 1985), we have identified certain classes of problems where the task of defining the knowledge structures and the task of actually building them can be effectively separated, with only the former being performed by a trained knowledge engineer. The problem of building a large collection of knowledge-based troubleshooters for electronic instruments is an example. In order to support the construction of a large class of such systems, it makes sense to perform the knowledge definition step for the overall domain initially, and to build *domain-specific development tools*, which include problem-oriented subsets of English and special purpose graphical displays, that can be reused in the development of each individual knowledge-based system.

Even in the context of such an approach, we have found that there is usually a shortage of capable knowledge engineers to carry out the knowledge definition phase, and that a well-defined methodology can be of great value here in aiding non-linguistically oriented computer scientists to carry out this verbal elicitation task. The major issue is how to get started defining the forms into which knowledge is to be cast.

We have found it an effective technique to begin this process by recording statements made by task experts on tape, and transcribing these to fairly natural English. When enough recording has been done, the statements begin to take on recognizable patterns. It is then possible to build a formal grammar for much of the relevant utterances, using linguistic engineering techniques such as semantic grammars The symbols of this grammar and the task specific vocabulary provide convenient points for defining formal sub-structures, which are pieced together to define a complete symbolic representation. Once the grammar is reasonably well-defined, the mapping to symbolic representation can be carried out with mapping techniques such as the f-structure constraints of lexical-functional grammar.

Up to this point, we can imagine that the entire task has been carried out on paper, or some machine-readable equivalent. Even in such a rudimentary form, the exercise is useful, because it provides a conveniently formal documentation for the knowledge representation decisions that have been made. However, it is also the case that these formal definitions, if appropriately constructed, provide all that is necessary to construct a problem specific interface for acquiring utterances expressed in this sublanguage. In fact, the idea of using this technique to build acquisition interfaces, using INGLISH, actually occurred as a result of wondering what to do with a grammar we had constructed simply in order to document our representation structures (Freiling et al. 1984).

We do not intend to imply that it is possible in complex knowledge based system applications to simply build a grammar and immediately begin acquiring knowledge. Often the process leading to construction of the grammar can be quite complex. In our case, it even involved building a simple prototype troubleshooting system before we had gained sufficient confidence in our representation structures to attempt a knowledge acquisition interface.

Nor do we intend to claim that all the knowledge necessary to build a complete expert system need be computed in this fashion. Systems such as INKA can be justified on an economic basis if they make possible only the transfer of a *significant fraction* of the relevant knowledge.

GLIB – A PROBLEM SPECIFIC SUBLANGUAGE

The knowledge acquisition language developed for electronic device troubleshooting is called *GLIB* (General Language for Instrument Behavior), and is aimed primarily at describing observations of the static and dynamic behavior of electrical signals as measured with oscilloscopes, voltmeters, and other standard electronic test instruments (Freiling et al. 1984). The grammatical structure of GLIB is that of a semantic grammar, where non-terminal symbols represent units of interest to the problem domain rather than recognizable linguistic categories.

This semantic grammar formalism is an important part of the DETEKTR methodology because the construction of semantic grammars is a technique that is easily learned by the apprentice knowledge engineer. It also makes possible the establishment of very strong constraints on the formal language developed by this process. Two of the design constraints we find it advisable to impose are that the language be unambiguous (in the formal sense of a unique derivation for each legal sentence) and that it be context-free. These constraints, as will be seen, make possible features of the interface which cannot normally be delivered in other contexts, such as menus from which to select all legal next terminal tokens. While increasing complexity of the acquisition sublanguage may make these goals unfeasible past a certain point, in simple systems they are features to be cherished.

Figure 1 shows a fragment of the GLIB grammar. In the DETEKTR version of INKA, sentences in this language are accepted, and mapped into Prolog terms for processing by a Prolog based diagnostic inference engine. At present, the elicitation is unguided: responsibility resides with the user to ensure that all relevant statements are generated. We are still studying the issues involved in determining completeness of a knowledge base and assimilating new knowledge. One outcome of these studies should be means of guiding the user to areas of the knowledge base that are incomplete and warrant further elaboration. Future enhancements to the system will include explanation and modification facilities, so that knowledge may be added or changed after testing the inference engine.

THE NATURAL LANGUAGE INTERFACE DESIGN

INGLISH - INterface enGLISH (Phillips & Nicholl, 1984) - allows a user to create sentences either by menu selection, by typing, or by a mixture of the two. This allows the self-paced transition from menu-driven to a typed mode of interaction. In-line help is available. To assist the typist, automatic spelling correction, word completion, and automatic phrase completion are provided. INGLISH constrains users to create statements within a subset of English, here GLIB.

A statement can be entered as a sequence of menu-selections using only the mouse. A mouse-click brings up a menu of words and phrases that are valid extensions of the

```
<rule> ::=
      IF <condition> THEN <conclusion>

<condition> ::=
      <context independent predicate> |
      <context independent predicate> WHEN <structural context>

<conclusion> ::=
      <functional context>

<atomic functional context> ::=
      <device> HAS FAILED |
      <device> IS OK

<functional context> ::=
      <atomic functional context> |
      <atomic functional context> AND <functional context> |
      <atomic functional context> OR  <functional context>

<atomic structural context> ::=
      <device> IS REMOVED

<structural context> ::=
      <atomic structural context> |
      <atomic structural context> AND <structural context>

<context independent predicate> ::=
      <value predicate>

<value predicate> ::=
      <value expression> IS <value expression> |
      <value expression> <comparator> <value expression>

<comparator> ::=
      IS EQUAL TO | = |
      IS GREATER THAN | > |
      IS LESS THAN | < |
      IS LESS THAN OR EQUAL TO | <= |
      IS GREATER THAN OR EQUAL TO | >= |
      IS NOT EQUAL TO | != |
```

Figure 1: A fragment of GLIB

current sentence fragment. Once a selection is made from the menu using the mouse, the fragment is extended. This sequence can be repeated until the sentence is completed. Creating a sentence in this manner compares with the NLMENU system (Tennant et al., 1983). Unlike NLMENU, keyboard entry is also possible with INGLISH. Gilfoil (1982) found that users prefer a command form of entry to menu-driven dialogue as their experience increases. When typing, a user who is unsure of the coverage can invoke a menu, either by a mouse-click or by typing a second space character, to find out what INGLISH expects next without aborting the current statement. Similarly, any unacceptable word causes the menu to appear, giving immediate feedback of a deviation and suggestions for correct continuation. A choice from the menu can be typed or selected using the mouse. INGLISH in fact allows all actions to be performed from the keyboard or with the mouse and for them to be freely intermingled. As only valid words are accepted, all completed sentences are well-formed and can be translated into the internal representation.

Figure 5, in the "INGLISH" window, shows a complete sentence and its translation, and a partial sentence with a menu of continuations. The numbers associated with each menu item provide a shorthand for entry, i.e., "#12" can be typed instead of "RESISTANCE". As menu entries can be phrases, this can save significant typing effort.

Input is processed on a word-by-word basis. Single spaces and punctuation characters serve as word terminators. Words are echoed as typed and overwritten in uppercase when accepted. Thus, if lowercase is used for typing, the progress of the sentence is easily followed. An invalid entry remains visible along with the menu of acceptable continuations then is replaced when a selection is made.

The spelling corrector (a Smalltalk system routine is used) only corrects to words that would be acceptable in the current syntactic/semantic context. As Carbonell and Hayes (1983) point out, this is more efficient and accurate than attempting to correct against the whole application dictionary.

Word completion is provided with the "escape" character (cf. DEC, 1971). When this is used, INGLISH attempts to complete the word on the basis of the characters so far typed. If there are several possibilities, they are displayed in a menu.

Automatic phrase completion occurs whenever the context permits no choice. The completion will extend as far as possible In an extreme case a single word could yield a whole sentence! The system will "soak-up" any words in the completion that have also been typed.

The spelling corrector and automatic phrase completion can interact in a disturbing manner. Any word that is outside the coverage will be treated as an error and an attempt will be made to correct it. If there is a viable correction, it will be made. Should phrase completion then be possible, a portion of a sentence could be constructed that is quite different from the one intended by the user. Such behavior will probably be less evident in large grammars. Nevertheless, it may be necessary to have a "cautious" and "trusting" mode, as in Interlisp's *DWIM* (Xerox, 1983), for users who resent the precocious impatience of the interface.

The system does not support anaphora, and ellipsis is offered indirectly. The interface has two modes: "ENTRY" and "EDIT" (Figure 5). These are selected by clicking the mouse while in the pane at the top right of the interface window. Rules are normally entered in the Enter mode. When in Edit mode, the window gives access to the Smalltalk editor. This allows any text in the window to be modified to create a new statement. After editing, a menu command is used to pass the sentence to the parser as if it were being typed. Any error in the constructed sentence causes a remedial menu to be displayed and the tail of the edited sentence to be thrown away.

The INGLISH interface alleviates the problem of linguistic coverage for designers and users of natural language interfaces. A natural language interface user composes his entries bearing in mind a model of the interface's capabilities. If his model is not accurate, his interactions will be error-prone. He may exceed the coverage of the system and have his entry rejected. If this happens frequently, use of the interface may be abandoned in frustration. On the other hand he may form an overly conservative model of the system and fail to utilize the full capabilities of the interface (Tennant, 1980). An interface designer is confronted by many linguistic phenomena, e.g., noun groups, relative clauses, ambiguity, reference, ellipsis, anaphora, and paraphrases. On account of performance requirements or on a lack of a theoretical understanding, many of these constructions will not be in the interface. INGLISH allows designers to rest more comfortably with the compromises they have made, knowing that users can systematically discover the coverage of the interface.

THE IMPLEMENTATION OF INGLISH

INGLISH parses incrementally from left to right and performs all checking on each word as it is entered. The parser follows the *Left-Corner Algorithm* (Griffiths & Petrick, 1965), modified to a pseudo-parallel format so that it can follow all parses simultaneously (Phillips, 1984). This algorithm builds phrases bottom-up from the left-corner, i.e., rules are selected by the first symbol of their right-hand-sides. For example, given a phrase initial category c, a rule of the form $X \rightarrow c \ldots$ will be chosen. The remaining rule segments of the right-hand side are predictions about the structure of the remainder of the phrase and are processed left-to-right. Subsequent inputs will directly match successive rule segments if the latter are terminal symbols of the grammar. When a non-terminal symbol is encountered, a subparse is initiated. The subparse is also constructed bottom-up from the left-corner, following the rule selection process just described. When an embedded rule is completed, the phrase formed may have the structure of the non-terminal category that originated the subparse and so complete the subparse. If there is no match, it will become the left-corner of a phrase that will eventually match the originating category.

The parser includes a *Reachability Matrix* (Griffiths & Petrick, 1965) to provide top-down filtering of rule selection. The matrix indicates when a category A can have a category B as a left-most descendant in a parse tree. The matrix is static and can be derived from the grammar in advance of any parsing. It is computable as the transitive closure under multiplication of the boolean matrix of left daughters of non-terminal categories in the grammar. It is used as a further constraint on rule selection. For example, when the goal is to construct a sentence and the category of the first word of input is c, then rule selection, giving $X \rightarrow c \ldots$ will also be constrained to have the property $S \triangleq X \ldots$ The filtering is applicable whenever a rule is selected: during subparses the constraint is to reach the category originating the subparse.

A semantic grammar formalism is used in INGLISH, which make the grammar application dependent. As was mentioned earlier, this format was independently chosen as part of the knowledge engineering methodology for describing the application domain. The rationale for the choice for INGLISH was that the simultaneous syntactic and semantic checking assists in achieving real-time processing. A fragment of the grammar is shown in Figure 1.

Pre-processing on the grammar constructs the terminal and non-terminal vocabularies of the grammar, the reachability matrix, and an inverse dictionary. The set of all possible initial words and phrases for sentences can also be precomputed.

The Smalltalk system contains controllers that manage activity on a variety of input devices and from these a controller was readily constructed* to coordinate mouse and keyboard activity in INGLISH. Either form of entry increments an intermediate buffer which is inspected by the parser. When a complete word is found in the buffer it is parsed.

Every phrase in an on-going analysis is contained in a Smalltalk object. The final parse is a tree of objects. The intermediate state of a parse is represented by a set of objects containing partially instantiated phrases. After the first word has established an initial set of phrase objects, they are polled by the parser for their next segments. From these and the reverse dictionary, a "lookahead dictionary" is established that associates expected words with the phrasal objects that would accept them. Using this dictionary an incoming word will only be sent to those objects that will accept it. If the word in not in the set of expected words, the dictionary keys are used to attempt spelling correction and, if correction fails, to make the menu to be displayed. If the dictionary contains only a single word, this indicates that automatic phrase completion should take place. A new lookahead dictionary is then formed from the updated phrase objects, and so on.

KNOWLEDGE TRANSLATION

The internal form of a diagnostic rule is a clause in Prolog. Sentences are translated using *functional schemata*, as in lexical-functional grammar. The functional schemata are attached to the phrase structure rules of GLIB (Figure 2).

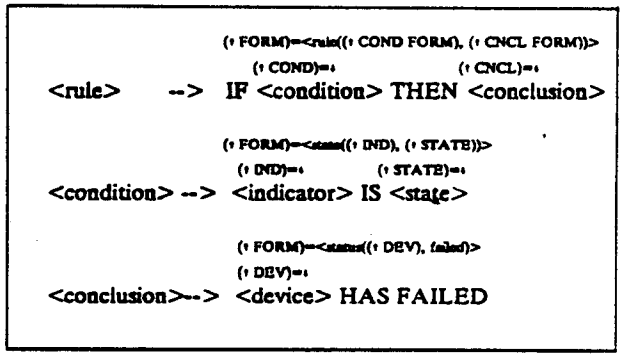

Figure 2: GLIB rules with attached schemata

Unlike lexical-functional grammar, the schemata do not set up constraint equations as the interface and the semantic grammar ensure the well-formedness and unambiguity of the sentence. As a result, propagation of functional structure is handled very quickly in a post-processing step since the applicable grammatical rules have already been chosen by the parsing process. Further, by restricting the input to strictly prescribed sublanguage GLIB, not English in general, the translation process is simplified.

---

* Smalltalk is an *object-oriented* language. Instead of creating a procedure that controls system operation, the user creates an object (usually a data structure), and a set of methods (operations that transform, and communicate with the object). Smalltalk programs create objects or send messages to other objects. Once received, messages result in the execution of a method.

Programmers do not create each object and its methods individually. Instead, classes of objects are defined. A class definition describes an object and the methods that it understands. Classes are structured hierarchically, and any class automatically inherits methods from its superclass.

As a result of this hierarchy and code inheritance, applications may be written by adapting previously constructed code to the task at hand. Much of the application code can be inherited from previously defined Smalltalk code. The programmer need only redefine differences by overriding the inappropriate code with customized code. (Alexander & Freiling, 1985).

The parser constructs a parse tree with attached schemata, referred to as a constituent-structure, or *c-structure*. Translation proceeds by instantiating the meta-variables of the schemata of the c-structure created by INGLISH to form functional equations which are solved to produce a functional structure (*f-structure*). The final rule form is obtained from the f-structure of the sentence when its sub-structures are recursively transformed according to the contents of each f-structure.

As an example, given the lexical-functional form of the semantic grammar in Figure 2 and the following sentence:

IF LED-2 IS ON THEN TRANSISTOR-17 HAS FAILED

the c-structure in Figure 3 would be produced. This shows that a rule has a condition part, COND, and a conclusion part, CNCL, that should become a clausal-form "rule(COND, CNCL)." The meta-symbol ↑ refers to the parent node and ↓ to the node to which the schema is attached.

The final phase of INKA interprets the f-structures to produce Prolog clauses. All of the information required to produce the clauses is contained in the FORM property in this example. The FORM property is printed, with all variables instantiated, to produce the final rule in the form of a Prolog clause. The f-structure of Figure 4 produces the Prolog clause

rule(state(led-2, on), status(transistor-17, failed)

KNOWLEDGE USE

Translated rules are sent to a diagnostic engine that has been implemented in Prolog. The diagnostic engine uses GLIB statements about the hierarchical structure of the device to build a strategy for successive localization of failures. Starting at the highest level ("the circuit" in GLIB terminology), named sub-circuits are examined in turn, and diagnostic rules retrieved to determine correctness or failure of the sub-circuit

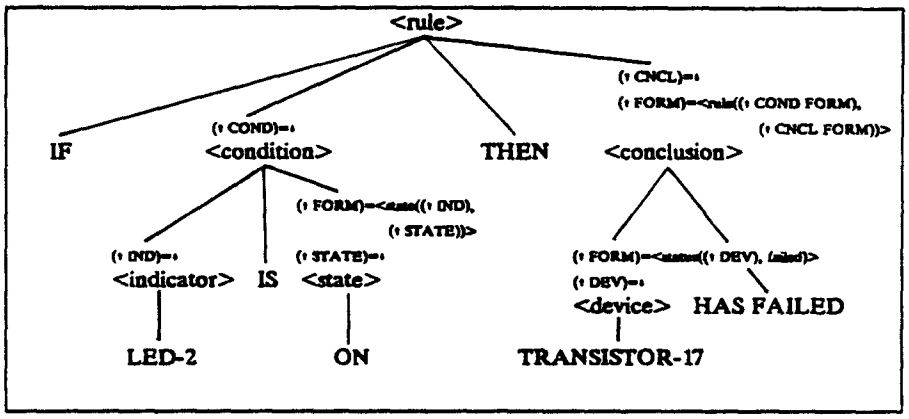

Figure 3: C-structure

The functional specifications of the example may be solved by instantiating the meta-symbols with actual nodes and assigning properties and values to the nodes according to the specifications. In the example given, most specifications are of the form "(↑ property)=value" where "value" is most often ↓. This form indicates that the node graphically indicated by ↓ in the c-structure is the specified property of the parent node (pointed to by ↑). Specifications are left-associative and have a functional semantic interpretation. A specification of (↑ COND FORM) refers to the FORM property of the parent node's COND property. The f-structure for the example is given in Figure 4.

in question. If no specific determination can be made, the sub-circuit is assumed to be functioning properly.

A sample session including acquisition of a rule and running of a test diagnosis is shown in Figure 5. The circuit used in this example consists of an oscillator which drives a light emitting diode (LED-2 in the schematic) and a power supply (LED-1 indicates when the power supply is on). The schematic diagram of the circuit is in the upper pane of the "Instrument Data" window; the circuit board layout is in the lower pane. Rules for diagnosing problems in the circuit

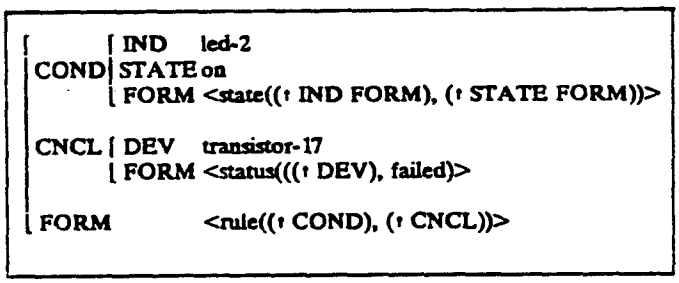

Figure 4: F-structure

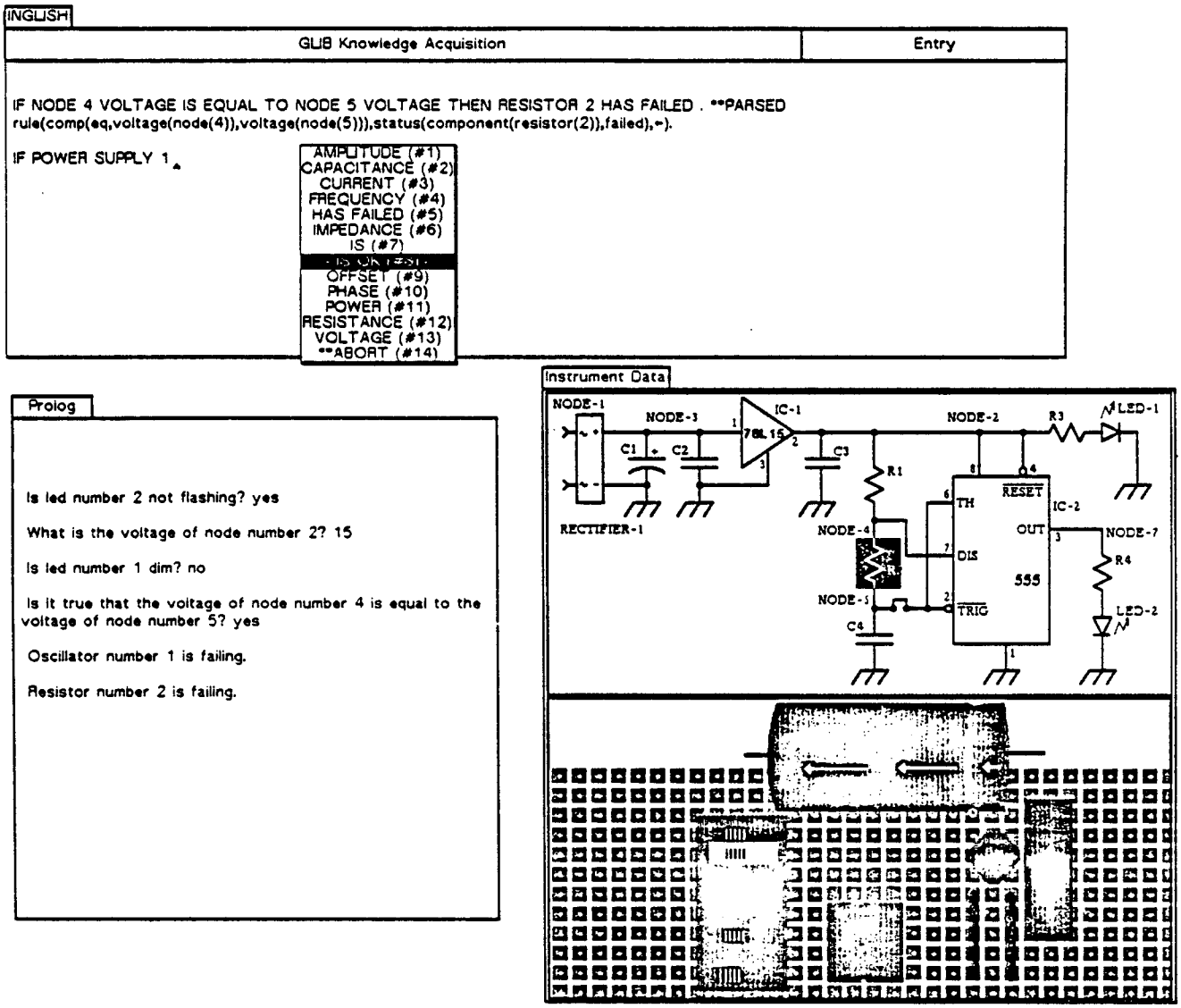

Figure 5: An INKA session

("troubleshooting" rules) are added to the system in the window labeled "INGLISH." The interface to the diagnostic engine is in the "Prolog" window. The "INGLISH" window shows a recently added rule, with its Prolog translation immediately below it. It also shows a partially completed rule along with a menu of acceptable sentence continuations. The user may select one of the menu items (either a word or phrase) to be appended to the current sentence. The "Prolog" window displays the results of a recent test diagnosis. This test was run after the first rule in the "INGLISH" window was added, but before the addition of the second rule was begun. The last question asked during the diagnosis corresponds to the first rule. Resistor 2, in both the schematic and board diagrams of the "Instrument Data" window, is highlighted as a result of running the diagnosis: whenever the diagnostic engine selects a specific component for consideration that component is highlighted on the display. Some 20 statements and rules have been collected for diagnosing the circuit; Figure 6 lists a portion of them with their Prolog translation.

```
THE CIRCUIT CONTAINS OSCILLATOR-1 AND POWERSUPPLY-1.
has_component(block(circuit), block(oscillator(1))).
has_component(block(circuit), block(powersupply(1))).

RESISTOR-1 IS PART OF OSCILLATOR-1.
has_component(block(oscillator(1)), component(resistor(1))).

IF LED-2 IS NOT FLASHING AND THE VOLTAGE OF NODE-2 IS EQUAL
TO 15 VOLTS THEN OSCILLATOR-1 HAS FAILED.
rule(and(not(state(led(2), flashing))),
         comp(voltage(node(2)), 15)),
    status(block(oscillator(1)), failed),
    []).

IF LED-1 IS DIM AND LED-2 IS OFF THEN RESISTOR-1 HAS FAILED.
rule(and(state(led(1), dim),
         state(led(2), off)),
    status(component(resistor(1)), failed),
    []).
```

Figure 6: GLIB rules with Prolog translations

DISCUSSION

Informal observations show that subjects generally need only a few minutes of instruction to start using INGLISH. Initially there is a preference to use the mouse to explore the coverage and then to begin to incorporate some typing. We have not had any long-term users to observe their trends.

Users could react negatively to limited language systems; even when the coverage is well-engineered users will occasionally encounter the boundaries. Fortunately, Hendler & Michaelis (1983) found that subjects were able to adapt to limited language systems.

INGLISH does not let the designer off the hook! A user can still have a statement in mind and not easily find a way to express it through the grammar. Diligent engineering is still needed to prepare a grammar that will allow a user to be guided to a paraphrase of his original thought. Nevertheless, the grammar design problem is simplified: when guidance is provided fewer paraphrases need be incorporated.

The use of a semantic grammar to define the fragment of English to be processed does impose limitations on the complexity of acceptable input. In the INKA system as it is currently constructed, however, there are two distinct ways in which the semantic correctness of an input can be enforced, first in the parsing of the of the semantically constrained grammar, and second in the translation process, as the functional structures are built.

In short, the our approach to building practical natural language interfaces does not depend on a semantic grammar to constrain input. In the future we intend to explore the use of a wider class of grammars that include a domain-independent kernel and a domain-specific component, like GLIB. In this approach we are in substantial agreement with Winograd (1984) who advocates a similar approach as an effective direction for further natural language research.

REFERENCES

Alexander, J.H., & Freiling, M.J. Building an Expert System in Smalltalk-80 (R). *Systems and Software*, 1985, *4*, 111-118,

Alexander, J.H., Freiling, M.J., Messick, S.L., & Rehfuss, S. Efficient Expert System Development Through Domain-Specific Tools. *Proceedings of the Fifth International Workshop on Expert Systems and their Applications*, Avignon, France, 1985.

Burton, R.R. *Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems* (Technical Report No. 3453). Cambridge, MA: Bolt, Beranek, & Newman Inc., 1976.

Carbonell, J.G., & Hayes, P.J. Recovery Strategies for Parsing Extragrammatical Language. *American Journal of Computational Linguistics*, 1983, *3-4*, 123-146.

Davis, R. Interactive Transfer of Expertise: Acquisition of New Inference Rules. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, 1977, 321-328.

[DEC] *TOPS-20 Reference Manual*. Maynard, MA: Digital Equipment Corporation, 1971.

Freiling, M.J., & Alexander, J.H. Diagrams and Grammars: Tools for the Mass Production of Expert Systems. *IEEE First Conference on Artificial Intelligence Applications*. Denver, Colorado, 1984.

Freiling, M., Alexander, J., Feucht, D., & Stubbs, D. *GLIB – A Language for Representing the Behavior of Electronic Devices* (Technical Report CR-84-12). Beaverton, OR: Tektronix, Inc., 1984.

Gilfoil, D.M. Warming up to Computers: A Study of Cognitive and Affective Interaction over Time. *Proceedings of the Human Factors in Computer Systems Conference*, Gaithersburg, MD, 1982, 245-250.

Goldberg, A. & Robson, D. *Smalltalk 80: The Language and its Implementation*. Reading, MA: Addison-Wesley, 1983.

Griffiths, T., & Petrick, S.R. "On the relative efficiency of context-free grammar recognizers." *Comm. ACM*, 1965, *8*, 289-300.

Hendler, J.A., & Michaelis, P.R. The Effects of Limited Grammar on Interactive Natural Language. *Proceedings of the Human Factors in Computer Systems Conference*, Boston, MA, 1983, 190-192.

Kaplan, R.M., & Bresnan, J.W. Lexical-Functional Grammar: A Formal System for Grammatical Representation. In J. Bresnan (Ed.), *The Mental Representation of Grammatical Relations*. Cambridge, MA: MIT Press, 1982.

McKeown, K.R. Natural Language for Expert Systems: Comparison with database systems. *Proceedings of the International Conference on Computational Linguistics*, Stanford, CA, 1984, 190-193.

Phillips, B. An Object-oriented Parser. In B.G. Bara & G. Guida (Eds.), *Computational Models of Natural Language Processing*. Amsterdam: North-Holland, 1984.

Phillips, B., & Nicholl, S. *INGLISH: A Natural Language Interface* (Technical Report CR-84-27). Beaverton, OR: Tektronix, Inc., 1984.

Tennant, H.R. *Evaluation of Natural Language Processors* (Technical Report T-103). Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1980.

Tennant, H.R., Ross, K.M., & Thompson, C.W. Usable Natural Language Interfaces Through Menu-Based Natural Language Understanding. *Proceedings of the Human Factors in Computer Systems Conference*, Boston, MA, 1983, 190-192.

Winograd, T. *Moving the Semantic Fulcrum* (Technical Report 84-17). Center for the Study of Language and Information, Stanford, CA, 1984.

[Xerox] *Interlisp Reference Manual*. Palo Alto, CA: Xerox Palo Alto Research Center, 1983.