AN ABSTRACT OF THE THESIS OF

Chris E. Jacobson for the degree of Master of Science in Computer Science presented on March 29, 1988. Title: ASTEK: A Multi-Paradigm Knowledge Acquisition Tool for Complex Structured Knowledge

Abstract approved:

Dr. Michael J. Freiling

Knowledge based systems can require large, highly complex and varied forms of knowledge. An effective knowledge acquisition tool to support such a system should allow the user to transfer and manipulate the different forms knowledge in a manner that is clear and intuitive. ASTEK is a knowledge acquisition tool that provides multiple paradigms for knowledge editing while maintaining a single, consistent framework designed using natural language discourse concepts.
ASTEK: A Multi-Paradigm Knowledge Acquisition Tool for Complex Structured Knowledge

by

Chris E. Jacobson

A THESIS submitted to Oregon State University

in partial fulfillment of the requirements for the degree of

Master of Science

Completed March 29, 1988

Commencement June 1988
APPROVED:

Redacted for Privacy

Assistant Professor of Computer Science in charge of major

Redacted for Privacy

Chairman of department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis is presented March 29, 1988

Typed by researcher for Chris E. Jacobson
TABLE OF CONTENTS

INTRODUCTION .................................................................................................................. 1
THE PROBLEM ...................................................................................................................... 3

PROBLEMS OF NATURAL LANGUAGE KNOWLEDGE ACQUISITION ................................. 6
  Multiple External Forms ................................................................................................. 6
  Utterance Planning ....................................................................................................... 7
  Fragmentation ............................................................................................................... 9
  Reference ..................................................................................................................... 10
  Reference Context ...................................................................................................... 10
  Open-Endedness ......................................................................................................... 11

A LOOK AT ASTEK .......................................................................................................... 13
  The Framework .......................................................................................................... 13
  The Editors .................................................................................................................. 16
    The Natural Language Editor ................................................................................... 16
    The Tree Editor ....................................................................................................... 17
    Other Editors .......................................................................................................... 19
  Reference in ASTEK .................................................................................................. 20
  Reference Scoping ...................................................................................................... 22
  The Recommend Function ......................................................................................... 22
  Setting Up an ASTEK Application ............................................................................ 23

CONCLUSIONS .............................................................................................................. 25

BIBLIOGRAPHY .............................................................................................................. 27

APPENDIX ....................................................................................................................... 29
PREFACE

The work behind this thesis took place as part of the research program of the Knowledge Engineering Program of the Tektronix Laboratories at Tektronix, Inc. The text of the thesis is a modified version of a manuscript which is to appear in the *Journal of Man-Machine Studies* and was presented in an earlier form at the *Second AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop* at Banff, Canada in November 1987. The manuscript and its earlier form was co-authored by Chris E. Jacobson and Dr. Michael J. Freiling.
INTRODUCTION

Knowledge acquisition is a process of gathering knowledge from one or more domain experts for utilization in a knowledge based system. To reach the necessary levels of performance, knowledge based systems can require large, highly complex and varied forms of knowledge.

A fundamental tenet of our approach to knowledge engineering is that effective knowledge based systems can and should be built by designing cognitively appropriate knowledge structures, and implementing inference strategies as an operational semantics for these structures in a process we call "knowledge analysis"[1,6]. There is a tradeoff, however, between the ease with which structures can be expressed by domain experts and the most tractable form for manipulation by an inference engine[10,15]. Effective knowledge acquisition tools must support paradigms for knowledge entry that are easy for the domain expert to use and minimize any cognitive burdens. They must then translate the actions within these entry paradigms into structures appropriate for operational interpretation. It is the need for multiple paradigms of knowledge entry that distinguishes our problem from that addressed in similar efforts[2].

In this paper we describe a multi-paradigm knowledge acquisition
tool called ASTEK (Acquisition of STructured Expert Knowledge). Unlike efforts directed at combining the tasks of knowledge analysis and knowledge acquisition described by[3] and efforts toward machine learning detailed in [20], ASTEK is focused on the expressibility aspect of the acquisition problem. ASTEK is implemented in Smalltalk and runs on Tektronix 4400 and 4300-series workstations. ASTEK permits domain-expert users to define structured knowledge fragments in winus. Winus combine a window for editing parts of a knowledge structure with a menu for selecting which part to edit. ASTEK attempts to exploit the wealth of current work in natural language systems while allowing the user to define each piece of knowledge using the most natural method of acquisition.
THE PROBLEM

We would like to build tools that domain experts find easy to use in entering the large amounts of knowledge needed for an effective knowledge based system. While domain experts are knowledgeable in their own field, however, they are often novices when it comes to entering large amounts of structured information into a computer system. To support the expert in this endeavor, we feel that an acquisition tool must serve to:

- provide an external form for the knowledge that has clear, intuitive semantics to an expert in the domain

- mediate the knowledge transfer from the external form, that which is natural to the domain expert, to the internal form, that which is most easily manipulated by the inference engine

- support input of knowledge that is "correct by construction"; the acquired knowledge should be constrained to be correct at the earliest possible time

- support knowledge management, such as revision control, and incremental testing; the tool should act as a comprehensive knowledge maintenance mechanism

Our research on knowledge acquisition has focused on acquiring knowledge for building troubleshooting assistant programs[5]. INKA
"[17, 18] was our first attempt to build a usable knowledge acquisition tool. The external form for troubleshooting knowledge was a set of English sentences generated by a context-free grammar[4]. The internal form was a collection of Prolog structures for use by a troubleshooting inference engine. The inference engine processed troubleshooting rules that had an extremely simple structure.

\[
\text{IF <measurement results>}
\{ \text{WHEN <setup procedure>} \}
\text{THEN <diagnostic belief>}
\] 

Because of their simple structure, rules were easily read and written as English sentences. The semantics for troubleshooting rules in English were intuitively clear. Defining the rule language in a context-free grammar insured that rules generated by INKA were correct by construction, at least to the syntactic level. INKA was powerful enough to build troubleshooting knowledge bases for small circuit board fragments.  

Subsequent experimentation with INKA and its underlying inference engine revealed several deficiencies when applied to a wider class of troubleshooting problems. First, expert troubleshooting technicians tend to have a high degree of explicitly procedural knowledge about how to perform setup and measurement tasks. This type of knowledge was quite cumbersome to encode in INKA rules, and required a better medium of articulation. 

Second, technicians became frustrated with trying to generate a single sentence containing all the information about how to reach one particular conclusion. A technician might generate part of the sentence, and then
decide that, rather than being the beginning of a rule description, it was really the middle of a troubleshooting rule (the fragmentation problem below). It was also not possible in INKA to relate parts of one rule (like its conclusion) to any other rule. What the technician wanted to generate was more like a paragraph. The increased structure induced by paragraph-sized descriptions introduced a need for visible cues as to the scope of each paragraph (the utterance planning problem below).

Based on this and other experiences, a more general set of troubleshooting knowledge structures was designed, and a new inference engine, HIPE[7,19] was built to process them. HIPE's knowledge structures included explicit procedures for tests and measurements, strategies to control test selection, and interpretations to link measurement results with diagnostic beliefs. With these larger and more complex knowledge structures came the need to rethink our assumptions about the tools necessary for acquiring them.
Several problem areas have been identified during our earlier efforts, some of which have been recognized in earlier papers as well. [2, 16] We make no claim that this collection enumerates all current problems with natural language based knowledge acquisition. Instead the collection represents the problems encountered that we have set forth to solve. These problems have contributed to the development of a set of principles used in the design and construction of ASTEK. Each of the problems and principles will be introduced and briefly discussed.

Multiple External Forms

The primary cause of our shift away from a pure natural language solution has been the need to describe procedural knowledge in a convenient manner. The tree-like structure of diagnostic procedures lies in stark contrast to the inherently linear nature of natural language. Yet in considering the concept of multiple forms we recognized that other forms as well may serve useful purposes. As well as methods to support the expression of natural language sentences and tree structures, such forms might include a menu selection method of acquiring a simple piece of information from a small, fixed set of choices or the fill-in-the-blank method utilized by Musen, et al in OPAL as well as many others.
Natural Articulation Principle

Different syntactic aggregates can best be expressed by using different media of articulation. Like Musen, et al.’s work in acquiring knowledge for ONCOCIN, we are using a domain model to "solicit and display knowledge" in a manner that is "intuitively understandable" to the domain expert. Our experience, and that of others, has shown that complex knowledge structures decompose into syntactic aggregates, structured groupings of symbols or other knowledge components. Each syntactic aggregate may have a preferred medium of articulation. A date, for example, is composed of a month, a day and a year and is readily formulated for expression using natural language. A troubleshooting procedure, on the other hand, is a tree structure whose nodes are tests and measurements and whose branches are conditions based on the results of prior tests and measurements. A procedure is cumbersome to describe using a linear tool such as natural language but can be readily expressed and modified with the use of a visual syntax.

Utterance Planning

With INKA the user is initially faced with a blank screen. INKA tried to be helpful by prompting with all words that could be used to start a sentence, but the users found this unsatisfactory. Once a user has managed to get started with the task of transferring knowledge, a challenge of formulating the knowledge still exists. Humans engaged in
conversation do not need to consciously plan a sentence because there are mechanisms in spoken language for extending and redirecting an initial utterance to capture a nearly arbitrary amount of subsequent detail. Our experience has shown that an expert, pressed to explain a process, will create a description like the following hypothetical example about how to play a record on a turntable:

"The playing needle is at the end of the arm. So you put the record on the turntable, start it going, and place the needle on. But before you do that make sure the volume is turned down. Oh, yes, you put the needle on the outer edge of the record."

When the knowledge sought is more structured, however, the expert needs to have an overall feel for all of the components a particular formulation in order to plan the expression appropriately. INKA provided no way to get an overall feeling for the type of expression to be expected.

The Recognition Principle

The medium of articulation should be recognition-based whenever possible. Research into the structure of human computer dialogues has shown that humans avoid the use of English constrictions (such as pronominal references) that would be difficult for a machine to interpret[11] Allowing the human user to recognize from a checklist the structure elements that need to be filled in makes it possible for the user to break-up the problem of expressing a large structure into simpler problems of expressing each component. Further the user does not have to worry about which connective to use between structural com-
ponents.

Menus serve a similar function at a lower level. Rather than fish for the lexical item a computer would most readily understand, the user is presented with lexical choices. The user has more of a feeling of being on the right track when one of the choices is appropriate. When none of the choices are appropriate, the user gets a clear signal that things are not going well. In addition menu selection of lexical items saves keystrokes and can lead to reduced time even for the mechanical process of entering an expression.

Fragmentation

INKA expected the knowledge to come in the form of a stream of self-contained sentences, each of which was well-formed. In practice, human users don’t seem to seem able to pack all that they want to say into a single sentence. Instead they tend to skip from structure to structure, or to pause to construct a reference to an object that they need to mention in the structure they are building.

The Fragment Expression Principle

When a relevant thought occurs to the user, it should be convenient to express it. The acquisition tool should be able to accept fragments of knowledge from the user without demanding the cognitive burden of formulation needed to construct a complete knowledge structure and without the need to
remember the knowledge fragments until such time as the tool desires them. When possible the user should even be able to create a fragment without any stipulation as to its contents, e.g. in forward referencing of a yet to be created knowledge structure.

Reference

It has been noted that for humans,

"The problem of reference, then, is the problem of specifying for the other person just which concept is meant, either with sufficient precision that the concept can be found, or with sufficient specification that it can be created"[16].

The most common form of reference used by domain experts to describe troubleshooting knowledge is a definite description involving noun phrases often composed of nominal compounds and with attached prepositional phrases. Spoken descriptions of troubleshooting knowledge also contain many pronominalizations but as Guindon noted, users often prefer to use the full object reference over the more efficient pronoun when communicating with a computer. Objects that need to be referred to in building knowledge structures include both real objects from the task domain, such as knobs or components, and conceptual objects, knowledge structures that have been or will be built.

Reference Context

In a reasonably sophisticated knowledge base for a troubleshooter, the number of objects that might be referenced grows to be too large to deal with as a whole. When presenting reference choices to the user, it is
necessary to constrain the set of objects to which the user may refer to those in which a reference would be meaningful.

The Reference Principle

The referencing of objects should be expedited. Research on the structure of natural language discourse has demonstrated that the acts of reference to objects and concepts constitutes a major part of discourse. Norman and Rumelhart go as far as to say, "In our analysis of language we assume that every noun phrase - in fact, every sentence - is an expression of reference". An act as pervasive as reference in a discourse tool must be strongly supported.

Open-Endedness

INKA could not make recommendations as to what types of sentences were appropriate to generate given the sentences already constructed. This leaves the human user with an uncomfortable feeling of open-endedness. There is no confirmation that the sentences being generated will add up to anything. An effective knowledge acquisition tool needs to be able to make suggestions about structures to work on or errors to correct[13]. To suggest an activity for the user, however, the acquisition tool needs a notion of both the completeness and the correctness of structures to a level beyond that of syntax.
The Correctness Principle

The knowledge entered by the domain expert should be constrained to be correct, to a level beyond that of mere syntax, at the earliest possible time. Simply stated the user should not have to enter a complex piece of knowledge only to find out much later that an object was referenced that was not valid in the context of the knowledge. The notion of correctness we use for knowledge bases is one of operational correctness. By that we mean that a correct knowledge base will produce the intended performance when operated on by the inference engine. Any more comprehensive theory of correctness is beyond the scope of this effort.

The Completeness Principle

A knowledge acquisition system should be able to identify missing components of knowledge given the current state of the knowledge base. A notion of completeness is necessary largely because the user is allowed to express knowledge in fragments. As with the term correctness, we use an operational definition of the term completeness. A knowledge base is considered complete if the knowledge structures in the knowledge base have all of their necessary components defined.
A LOOK AT ASTEK

We have discussed some of our past efforts at knowledge acquisition in this paper. As well we have identified some of the weaknesses of our past approach and postulated some principles for a better approach to knowledge acquisition. We would like to now turn to look at our most recent tool for knowledge acquisition, ASTEK. We will discuss winu structures, the media that have been implemented for editing knowledge components and the support in ASTEK for referencing objects and formulating knowledge. The figures referenced are parts of an annotated example contained in the appendix at the end of the paper.

The Framework

ASTEK is a general interface tool directed at aiding the user in efficiently communicating structured knowledge. As in OPAL[15] the acquisition process in ASTEK is guided by a model of the domain knowledge. This model is constructed by specifying the types of knowledge structures for the domain in terms of their components and the appropriate editing mechanism to apply to each component. Also utilized by ASTEK in our diagnostic applications is a model of the system being diagnosed.

ASTEK is implemented in Smalltalk-80 and is a window-based tool. An explicit "focus space" of objects that can be readily referenced is helpful in understanding human utterances in the context of a connected
discourse[9]. ASTEK, therefore, supports the user's need to continually refer to objects by maintaining an explicit and visible object reference stack in a window, and exploiting the concept of "pointing as reference"[5] to allow the user to readily refer to objects present on the screen without having to generate cumbersome referring expressions like "the node whose voltage was measured in the last test that I wrote". Figure 1(a) shows a sample focus space in the Discourse Manager window. ASTEK also supports descriptive reference to objects in its natural language editor.

When the technician makes a request to the discourse manager to edit a knowledge structure [Figure 1(b)], the appropriate winu is created. A winu combines a window for editing parts of the knowledge structure with a menu for selecting which parts to edit. Selecting a knowledge component (a syntactic aggregate) in the menu pane of the window installs the appropriate editing mechanism (the medium of articulation) in the editing pane. In figure 2 the technician has selected the Procedure component of muxStrategy and is expressing the first step of the procedure using the natural language sub-editor of the tree editor. As knowledge transfer is performed through manipulation of winus, ASTEK's overall framework is easy to learn. If the knowledge analyst can design classes of knowledge structures that consist of natural syntactic aggregates and assign a medium of articulation to each, ASTEK can be readily applied to a new domain.

The use of winus as the means for the user to define a knowledge structure assures that the user is always aware of the overall knowledge
structure to which he is contributing. Each time that an editor is invoked, a specific type of knowledge is known to be expected and its role in the overall knowledge structure is clear. The editing mechanism invoked will be one that is appropriate and lends support to the expression of the information.

The only limit on the number of winus that may be visible on the user's screen at any one time is the size of the screen. The user can readily maneuver from winu to winu as pieces of knowledge occur to him. The only complexity that this introduces is the shift of focus as the user moves to a new structure. ASTEK currently maintains a single, global object stack for use in referencing. Figure 3(c) shows how the discourse manager focus space changes as a result of editing muxStrategy, creating remoteChannelReset, and referring to busyChannel. We have seen a need for a more comprehensive scheme of "attentional states" through the introduction of "local focus spaces" as described in Grosz and Sidner's work and plan to implement such a scheme in future efforts.

Through the use of graphics, menus and word prediction, many simple pieces of knowledge are expressible without the cognitive burden of sentence formulation. This results in a natural and efficient means of communication between the user and the acquisition tool.

It is interesting to note that the implementation of ASTEK in an object oriented language has been a very natural approach. Object classes exist for each type of knowledge structure and the class "knows" which medium to apply to its components and how to convert the components
to the internal form for use by the inference engine. The acquisition environment is a structured interface environment that is independent of any particular set of knowledge classes or media. New classes of structured knowledge can be easily added for nearly any domain without modifying the ASTEK framework.

The Editors

ASTEK provides an open-ended framework for knowledge acquisition in that any editing mechanism supported by the underlying display hardware can be incorporated into the framework. The knowledge analyst, with the domain expert, can decide which method of articulation is most appropriate for each component of a knowledge structure. Several media are being used by ASTEK and some of these will be briefly discussed.

The Natural Language Editor

The natural language editor was constructed with the use of a grammar development environment developed in the Tektronix Laboratories[14]. KIERYA provides a rich functionality for developing and testing a unification-based grammar[12]. The editor offers such features as word prediction, which reduces the effort of sentence formulation, and word completion, for minimization of keystrokes. Figure 3(b) shows the natural language editor prompting the technician for a procedure name as the technician describes a step of a test procedure.
The editor provided with KIERYA also maintains an always-present cursor that turns thumbs-down when the sentence cannot be "understood" and turns thumbs-up when a valid sentence has been completed. This allows the user to detect any input problems as soon as they are created [figure 4(a)].

As mentioned earlier, KIERYA's editor supports descriptive object reference. Partial descriptions of objects such as "channel of remote node" are unified with full descriptions of the objects in the local context to determine the correct reference. In the cases where the unification is unable to completely disambiguate the reference, the user is asked to choose from the set of recognizable object descriptions that unified with the partial description. Also in support of the correctness principle, KIERYA's editor often has the objects from the context of the knowledge being edited pre-loaded into the lexicon when the editor has started. Only references to objects that are appropriate in the current context are, therefore, allowed by the parser, and unification disambiguation is the only process that needs to occur after parsing. The disambiguation occurs only over a highly constrained set of objects.

The Tree Editor

The procedures used by the inference engine, HIPE, are hierarchically related references to knowledge structures. Describing a hierarchical structure using a linear syntax, such as that of natural language, is cumbersome. The tree editor assists this task by utilizing graphical manipulation techniques to convey the structure. Initially the tree editor is
blank, as in figure 2(a). The expert technician chooses to add a step by selecting the activity from a menu then describes the step using the natural language editor that was invoked. A grammar is associated with the procedure tree editor that allows the description of the HIPE structures strategy, test, measurement, and action. The same tree editor, in combination with a different grammar, is in use to describe diagnostic conclusion structures [figure 4].

Another task of the procedure editor is to link local variables in the procedure context to the parameters of the procedure being referenced. Parameters for HIPE procedures are used to pass information about specific device instances. Perhaps as a corollary to the natural articulation principle, the user ought to be shielded from computer science notions such as parameters. ASTEK therefore creates all possible variable bindings when a procedure is referenced and warns the user when bindings cannot be made (when the appropriate local variables do not exist). The user can then simply add steps to the procedure that will provide the desired local variable(s). The parameters for a test or measurement are created automatically in the standard case where the structure is created by ASTEK when first referenced. The creation and binding of parameters into the local context provides a powerful support tool to a user with no experience with such programming concepts.

Conditions are added to branches of the trees to determine which path to follow. Like the other nodes of the tree, conditions are added using a menu selection and described using a natural language editor. The discourse manager establishes a dynamic lexicon from the context of
the condition editor that includes (a) all parameters of the structure that
the tree is a part of and (b) any output parameters of tests or measure-
ments on the path leading to the condition on the procedure tree. The
dynamic lexicon is combined with a static lexicon to produce word predic-
tion menus of the entities upon which the conditions can be based. The
use of parameter names in the description of conditions reflects an area of
weakness in ASTEK. Work needs to be done to refer not to the value of
parameter VerifyExceptions of measurement verifyChannel but rather to
"exceptions in performing verifyChannel", again in an effort to hide the
computer science concepts from the user.

Trees are constrained to be structurally valid at all times. HIPE
requires that the root node of a procedure tree not be a branch. The menu
item to add to an empty tree consequently does not allow branches to be added [figure 2(a)]. Implication trees, on the other hand, require that the
root node be a condition and are constrained accordingly. Nodes of the
tree can be selected and manipulated. In figure 3(a) the reference to
remoteChannelReset has been selected and the edit menu item chosen to
open an editor on the procedure definition. ASTEK permits the display of
the tree structures to be scrolled in either direction, or panned with a
mouse-driven panning cursor.

Other Editors

Other editors currently in use are a menu editor, allowing the selec-
tion of one of a set of pre-defined objects from a menu by pointing with
the mouse; a list editor, assisting in the creation of a set of objects; and a
Reference in ASTEK

Traditionally users of computer based tools have had to utilize correct words for the objects they wish to refer to or the actions they wish to occur. Studies described in [8] showed that two people favored the same term for an object or action with probability <0.20. The optimal strategy cited to remedy this situation was the use of unlimited aliasing. ASTEK employs this strategy in the natural language editor (see figure 4) and attempts to further increase support for referencing by broadening the scope of the referencing mechanisms.

Two modes of reference are employed in ASTEK, direct reference, using a mouse to point at a visual representation of an object, and descriptive reference, using a phrase that describes an object. In keeping with the design principles of ASTEK, direct reference is employed wherever possible and several forms of direct reference are employed:

- from a menu of object representations
- from a visible object stack
- from a graphical depiction of a model or a procedure

The use of descriptive reference is employed whenever the objects to be referred to are not readily at hand. This situation exists in ASTEK only when employing the natural language editor. To aid in descriptive reference and to support the design principle of constraining input to be correct, the lexicon is pre-loaded with recognizable names of tests and
measurements in the local context. This "dynamic lexicon" works well for a small collection of objects that are within the local context and minimizes the cognitive burden of searching through the users memory or even a long menu. For instance, figure 3(b) shows use of a dynamic lexicon of procedure names in defining a step of a procedure tree of the remoteChannelReset test structure. For references to elements of the model the user is allowed to utilize partial and pronominal descriptions such as "the thing connected to the terminal" to refer to a line driver. The user's description is translated into a feature structure representation and unified with a full description of each of the possible referents. If more than one object unifies with the description the user is asked to select from a menu containing a system-generated description of each of the potential referents.

We are planning to extend the allowed referring expression for a measurement within a test procedure to include such notions as the first, last or n-th step of a procedure or invocation of a measurement. We refer to these as "dynamic cues" because editing the steps of the procedure can change which cue is appropriate in a referring expression for a given step. These cues would create features in the description that act as guides to the disambiguation process rather than representing characteristics of the referent.

It is important in the support of object referencing for ASTEK to have the capability to generate unique and recognizable object descriptions. It is critical that these descriptions be concise, especially with the use of menus. Ideally the user should be able to learn to express an object
reference to the system by mimicking the system generated object descriptions. If ASTEK utilizes dynamic cues, however, the system can generate descriptions of objects that may not be valid when the user attempts to mimic, thus the ideal cannot be fully realized. A "dynamic object description", one generated using dynamic cues, is only valid as long as the order of the procedure relative to the objects disambiguated by the dynamic cue remains unchanged.

Reference Scoping

The number of objects in just a modest knowledge base is too large to sort through in order to make a simple reference. ASTEK therefore constrains any menus of objects that can be chosen in a reference activity to those that would be potentially meaningful in the local context. In the natural language editor this is implemented through the use of the dynamic lexicon.

The Recommend Function

To combat the feeling of open-endedness felt by the user, the desire to know whether the knowledge being entered is enough to add up to a complete set of knowledge, the user can request that ASTEK identify a structure to work on next. The recommendation is based on several pre-defined criterion:

- a structure that is complete but has erroneous information in it

- a structure that has been referenced by another structure but does not yet exist
a structure that has been edited but is not yet complete

The recommendation of an incomplete structure is prioritized last with the intuition that if a structure is left incomplete but the user is still asking for assistance, then the user may have had some reason for leaving the structure unfinished. This capability will aid the user in creating a knowledge base that is complete and correct (to some level).

Figure 5(a) shows the technician requesting a recommendation after completing the remoteChannelReset test. ASTEK determines that the test contains a reference to a busyChannel measurement that has not been defined [figure 5(b)], so the user is given the option of editing the measurement definition or having ASTEK recommend another structure.

The recommend function aids in reducing the number of incomplete or incorrect knowledge structures in the knowledge base but it is not a comprehensive theory in guiding the user to create a complete and correct knowledge base. We leave this hard problem for future efforts.

Setting Up an ASTEK Application

ASTEK has been designed to be used by a domain expert with minimal involvement of a knowledge engineer. The construction of a new knowledge base within an existing task domain, such as troubleshooting, can be done without modification to any Smalltalk-80 code. For an expert to use the system, the only training needed is in the HIPE view of a model and the structures that it employs and in the basics of mouse movement and window manipulation.
The initial setup of the system for a completely new task, however, requires the efforts of a knowledge engineer with the ability to program in Smalltalk-80 and the ability to construct a semantic grammar. In our environment it is assumed that a knowledge engineer has performed an analysis of the application domain that includes the creation of a corpus of sentences that the domain experts use to explain their activities and a resultant set of relevant knowledge structures. The grammars used by the natural language editor can be developed as a mapping from the sentence forms of the corpus into translations representing the information of the knowledge structures. The ASTEK structures are then created as implementations of the knowledge structures from the analysis. The slots of the new structures, representing the individual syntactic aggregates, are assigned the most appropriate of the available editing mechanisms. New editors may also be implemented and utilized if desired. Finally methods are implemented for each structure to generate the internal form of the knowledge to be utilized by the inference engine.

Many applications, such as the troubleshooting application, require significant reference context support. For each knowledge structure slot that uses a natural language editor, a method must be implemented which specifies where to find the domain objects that may be referenced in the context of the slot. While it would be desirable to implement the automatic creation of these reference contexts, further effort is needed to develop a theory that allows the analysis phase to produce information to support the activity.
CONCLUSIONS

The path that we have taken in knowledge acquisition tools was to start with a simple natural language solution and extend it where the natural language paradigm failed to meet our needs. This has led us in the direction of a form and editor paradigm of knowledge transfer. This paradigm better supports utterance planning, fragmented articulation and variations in the medium of articulation. By constructing the acquisition interface around knowledge structures derived from an analysis of the domain, the user needs only to learn the Smalltalk-80 interface to rapidly come to speed with the use of the system that we have developed. ASTEK looks much like Musen, et al’s OPAL, except that form entries themselves may be complex structures or references to complex structures. It may be fair to ask whether natural language plays a sufficiently important role in ASTEK. We feel that natural language is, indeed, important in the following ways. At the level of small knowledge fragments, natural language grammars support the medium of articulation when preferred. At a deeper level, ASTEK was designed with the realization in mind that a knowledge acquisition is an extended dialogue between a human expert and the computer. Drawing on work related to the needs of extended dialogues, we have added features, such as object reference and a recommendation function, that make knowledge acquisition more than the isolated editing of templates. The discourse tools help individual acts of articulation be performed as desired while bringing together the individually
articulated knowledge fragments into a single, coherent knowledge base as the acquisition task reaches completion.
BIBLIOGRAPHY


[10] Gruber, T.; Acquiring Strategic Knowledge from Experts; University of Massachusetts Experimental Knowledge Systems Laboratory; Amherst, Massachusetts; February, 1987


[20] Witten, I.H. and MacDonald, B.A.; Using Concept Learning For Knowledge Acquisition; 2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop; Banff, Canada; October, 1987
APPENDIX
APPENDIX

ANNOTATED ASTEK SESSION

Although ASTEK can be tailored to the acquisition of most forms of structured knowledge, we will illustrate its features with respect to the task of capturing knowledge about troubleshooting a data communications multiplexer. A multiplexer accepts communication signals from several channels, and combines them into one high-speed signal for transmission across a communication line. At the other end, a companion multiplexer accepts the high-speed signal and breaks it up into its component channels for further transmission along separate communication lines.

The example we will deal with involves a test and repair procedure for resetting some particular channel card on one of a pair of multiplexers. Although some knowledge-based troubleshooting systems distinguish between diagnosis and repair tasks, our experience is that the two are inextricably intertwined. The channel reset procedure will repair several simple multiplexer faults, but its failure (and the way in which the reset procedure fails) yields information useful for further diagnostic attempts.

The test we define is performed on a particular multiplexer pair, one node of the pair (either the "local" or the "remote" node), and one channel that the multiplexer. The same procedure can be invoked to troubleshoot channel 7 of the local multiplexer, channel 4 of the remote mul-
teplexer, etc. Thus we need to parameterize the test with parameters to identify an instance of a mux, node, and channel. ASTEK automatically creates and binds parameters as tests are constructed. The HIPE inference engine distinguishes four levels of troubleshooting activity, strategies, tests, measurements, and actions. Strategies guide the overall localization process. Tests are structured procedures determining the status of specific objects in the system. Measurements are procedures for acquiring values in testing an object, and actions are the atomic units of behavior, which involve affecting the object under test.

The reset procedure is quite simple. First the channel is "busied out" to establish a known state. Then the reset measurement is executed, and finally a verification test is performed to establish that the channel has indeed been reset to an operational state. A separate interpretation section of the test knowledge structure describes what diagnostic conclusions may be drawn from different results. If the procedure succeeds, the problem is believed to be fixed. If the reset procedure completes, but the verify measurement fails, the problem is believed to be with the channel hardware itself. On the other hand, if a timeout occurs in attempting to busy out the channel, the problem can only be localized to the multiplexer.
Figure 1 -- The object reference stack is displayed in a pane of the Discourse Manager window (a). Also visible is a window displaying the model for the system under test. The model is described with nodes, representing the modules of interest, connected by arcs, representing part-of relationships. The timeplex model contains module types with multiple instances distinguished by keywords (e.g. local and remote nodes) and module types in which many identical instances exist (e.g. channels) that are distinguished only by instance numbers. In (b) the user has selected a procedure, `muxStrategy`, and is asking to edit it.

Legend: {a strategy} [a test] (a measurement)
Figure 2 -- In an edit window for a procedure, the slots of the structure are listed in the upper pane with a summary of the value of each slot. Selecting a slot in the upper pane causes the appropriate editor for the slot to be instantiated in the lower pane. In (a) the user has selected the procedure slot and is using the procedure tree editor to add a step to the procedure. The first step has been described using a natural language editor (b). The procedure referred to, remoteChannelReset, has not been previously defined and is not contained in the grammar. Astek recognizes this and offers to create a skeletal procedure definition for the name. Astek analyzes the object reference to a module, "channel of remote node", and the model of the system to determine that there is not enough information present to determine which channel of the remote node is to be reset. A warning is generated to the user that a step may be needed in the procedure to determine the channel number (c).

![Diagram](image.png)
ImuxStrategy WhichMux

- name -> muxStrategy
- device type -> timeplex
- device -> ParameterList (/WhichMux/)
- procedure -> an empty tree
- completeness -> weak
- inertia -> medium

2682>>

describe the next step

<table>
<thead>
<tr>
<th>test channel of remote node using remoteChannelReset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notifier</td>
</tr>
<tr>
<td>Proceed to create new structure</td>
</tr>
</tbody>
</table>

You may want to add steps before this one that reference the following devices:
- channel

Proceed to continue.
Figure 3 -- The user has selected the reference to the test remoteChannelReset in muxStrategy and has asked to edit it (a). An edit window opened shows the test definition with the slots that were automatically filled in by Astek. Most interestingly the set of parameters needed to refer to the specific instance of the module type (or device) that the test applies to are determined for the user. In (b) the user is describing the first step of the procedure for the reset test. The word prediction feature shown reflects the dynamic lexicon and the prediction of all entries that would be grammatically correct. The user is selecting an existing measurement definition busyChannel. This causes the object reference stack in the discourse manager (c) to update itself to reflect that busyChannel is the most recently referenced procedure. This is useful in later object reference disambiguations. The completed procedure definition is presented in (d).
<table>
<thead>
<tr>
<th>Procedure: remoteChannelReset</th>
<th>completeness: name &gt; remoteChannelReset</th>
</tr>
</thead>
<tbody>
<tr>
<td>purpose:</td>
<td></td>
</tr>
<tr>
<td>author:</td>
<td></td>
</tr>
<tr>
<td>date: 2 December 1987</td>
<td></td>
</tr>
<tr>
<td>device: ParameterList (WhichMux / WhichNode / WhichChannel)</td>
<td>applies if: true</td>
</tr>
<tr>
<td>procedure: an empty tree</td>
<td>result implications: an empty tree</td>
</tr>
</tbody>
</table>

**Parameter List**: WhichMux WhichNode WhichChannel

---

**Discourse Manager**

- TimeplexKB
- AutoKB

**Busy Channel**

- remoteChannelReset WhichMux WhichNode WhichChannel
- muxStrategy WhichMux
- verifyChannel WhichMux WhichNode WhichChannel
- resetChannel WhichMux WhichNode WhichChannel
whichMuxStrategy

- name -> muxStrategy
- device type -> timeplex
- device -> ParameterList (/WhichMux/)

procedure: an

completeness -> [remoteChannelReset WhichMux WhichNode WhichChannel]

inertia -> medium

author -> Bill Sayles

Date -> 2 December 1987

device -> ParameterList (/WhichMux/ /WhichNode/ /WhichChannel/)

applies if -> true

result implications -> an empty tree

---

2682>>[remoteChannelReset WhichMux WhichNode WhichChannel]

2725>>(busyChannel WhichMux WhichNode WhichChannel)

2728>>(resetChannel WhichMux WhichNode WhichChannel)

2731>>(verifyChannel WhichMux WhichNode WhichChannel)
Figure 4 – The implications tree (a) specifies what can be believed about the system under test by virtue of having performed the steps of the test procedure. A natural language editor is accepting a description of a diagnostic conclusion. ASTEK is indicating the successful parse of the sentence with the thumbs-up cursor. The grammar can be tailored to any variety of regional dialects, in this case Valspeak (a much parodied dialect originating in the San Fernando Valley). This is accomplished both through aliasing of lexical entries and through the use of multiple forms of sentences that result in the same semantic content. Astek has interpreted the conclusion described and constructed a meaningful interpretation for use by the inference engine (b).
name -> muxStrategy
device type -> timeplex
device -> ParameterList (/WhichMux/)
procedure -> a tree with 3 nodes

completeness -> author -> Bill Sayles
date -> 2 December 1987
device -> ParameterList (/WhichMux/ /WhichNode/ /WhichChannel/)
applies if -> true

T2735 >> !channel is ok@strong!
T2739 >> !channel is bad@certain!
T2731 >> /VerifyExceptions/ = noErrors?
Figure 5 -- The user is requesting that ASTEK recommend a procedure that needs to be edited (a). In (b) the busyChannel measurement has been suggested because it is as yet incomplete. The user has accepted the recommendation causing ASTEK to open an edit window for the measurement (c).