WL-TR-91-4131

AD-A243 972

‖‖‖‖‖‖‖‖‖‖‖‖‖‖‖‖

DTIC
ELECTE
DEC 2 0 1991
C
D

ENLIGHTEN: A LOW COST UNIFIED EXPERT SYSTEM TOOL
FOR MANUFACTURING

R.S. Insley, P. A. Evans, P. W. Kennedy,
R. F. Matejka, M. Samadar, Ph.D

Universal Technology Corporation
4031 Colonel Glenn Highway
Dayton, OH 45431-1600

December 1991

Final Report for period December 1988 - June 1991

91-18552
‖‖‖‖‖‖‖‖‖‖‖‖‖‖‖‖

MATERIALS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	Approved for public release; distribution is unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) WL-TR-91-4131

6a. NAME OF PERFORMING ORGANIZATION University Technology Corp.	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Materials Directorate (WL/MLIM) Wright Laboratory

6c. ADDRESS (City, State, and ZIP Code) 4031 Colonel Glenn Highway Dayton, OH 45431-1600	7b. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, OH 45433-6533

8a. NAME OF FUNDING / SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-89-C-5701

8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS			
	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
	65502F	3005	51	24

11. TITLE (Include Security Classification)

ENLIGHTEN: A LOW COST UNIFIED EXPERT SYSTEM TOOL FOR MANUFACTURING

12. PERSONAL AUTHOR(S)
R.S. Insley, P.A. Evans, P.W. Kennedy, R.F. Matejka, M. Samadar

13a. TYPE OF REPORT Final	13b. TIME COVERED FROM Dec 88 TO Jun 91	14. DATE OF REPORT (Year, Month, Day) December 1991	15. PAGE COUNT 70

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Knowledge engineering, Expert systems, Artificial intelligence, Manufacturing, Semantics

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The positive impact of expert system technology upon manufacturing can be heightened by eliminating some of the impediments in the creation of knowledge based systems. This report discusses the development of a semantic-based expert system development software tool, titled Enlighten, which is able to acquire, refine and deliver knowledge from a domain expert with limited computer expertise. Enlighten builds upon the Apple Macintosh philosophy for the user interface providing a tool that non-computer literate experts can use without the need of an intermediary. The design effort relies on semantic interpretation of typed English-like statements to create linked hierarchies of related concepts. Significant challenges in semantic interpretation and knowledge representation were encountered and are documented in this report.

| 20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Walt Griffith	22b. TELEPHONE (Include Area Code) (513) 255-8787	22c. OFFICE SYMBOL WL/MLIM

DD Form 1473, JUN 86   Previous editions are obsolete.   SECURITY CLASSIFICATION OF THIS PAGE

DISCLAIMER NOTICE
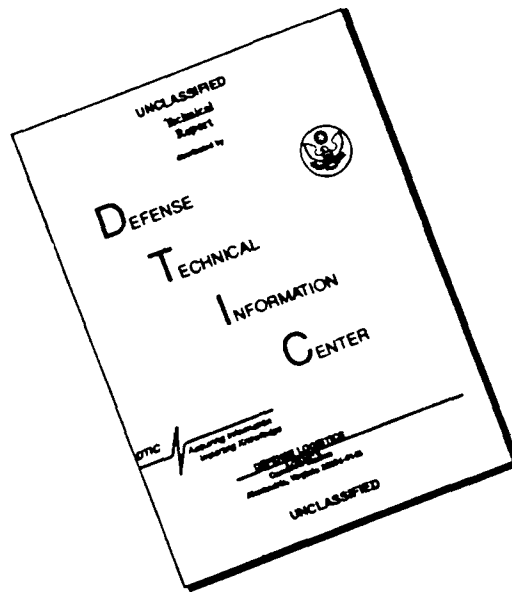
THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.
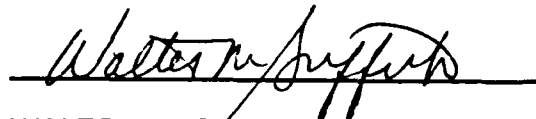
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

OLIVER D. PATTERSON, Captain, USAF
Technical Program Manager
Manufacturing Research Branch
Integration and Operations Division
Materials Directorate

WALTER M. GRIFFITH, Ph. D.
Branch Chief, Manufacturing Research
Integration and Operations Division
Materials Directorate

MATHEW DIBIASE, MAJ, USAF
ASSISTANT DIVISION CHIEF
INTEGRATION & OPERATIONS DIVISION

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WL/MLIM , WPAFB, OH 45433- 6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

Table of Contents

List of Figures

List of Table

1.0   Introduction

This Phase II Small Business Innovative Research (SBIR) effort created a software tool to aid in the development of expert systems for manufacturing. Generally, the tool was to facilitate knowledge acquisition, refinement, and delivery in some way which would minimize the requirement of the user to understand computer programming-related tasks of expert system development. The domain of focus is manufacturing with emphasis on aerospace manufacturing. The following is the final report which summarizes the effort.

1.1.   Goals and Objectives

This project is entitled "Low Cost Unified Expert System Tool for Manufacturing." The key phrases of the title reflect the objectives of the project. Each is discussed separately.

The intent is to provide a tool to a large number of users, thus the "Low Cost" requirement. Many of the better known expert system tools have costs only large companies can afford, often $20,000 or more for the software alone. This cost does not reflect the expert, knowledge engineer, or programmer time or training. Some of the more successful tools include Knowledge Engineering Environment (KEE), ART and Nexpert. The product, called Enlighten, was developed to be very affordable from both a software and hardware perspective.

"Unified" suggests that Enlighten contains an integrated set of features to facilitate the entire expert system development process. The tool is intended to provide a novice computer user with the opportunity to add information to the system, have the system perform some sort of refinement on that information, and then deliver that information as a complete expert system. This acquire, refine, and deliver strategy is intended to let the user capture and convey information about various concepts in which he or she is interested with a single tool.

"Expert System" implies that Enlighten uses knowledge and reasoning, resulting in something more than "data processing" for the user. This system accepts information about various topics of interest to the user and processes them, rather than simply placing data into a file. The system analyzes the content of information being added and attempts to find relationships between the different concepts, and can display these relationships to the user.

"Tool" implies that Enlighten amplifies the inherent power of the user. First generation expert system tools, usually called expert system shells, allow a user to build an expert system by contributing all domain knowledge necessary to make useful conclusions. When the system is used later, the expert or another user must answer a series of questions from the expert system in order to gain information or reach conclusions. Enlighten moves beyond first generation tools since it contains knowledge which assists users in building specific expert systems. Thus, Enlighten is an expert system which helps users build expert systems. Enlighten also allows users to explore information in any order desired. In traditional expert systems, the user follows the expert system, answering questions as desired; in Enlighten the expert

system follows the user, showing all requested information on demand. Because it is an expert system assisting in expert system building and because it allows the user to maintain control, Enlighten is a second generation expert system tool.

Finally, "Manufacturing" is the primary domain of focus for Enlighten. As will be discussed in Section 2.1, the scope of this project was narrowed to include only classification or diagnosis tasks as measures of the success of the system.

From its beginning, this project sought to create a usable product. There are a multitude of programs which receive little use, if they are even purchased. Ease of use, both start up (learning) and long term, and product usefulness were foremost considerations during product design and implementation. The better known tools mentioned above are flexible and comprehensive, but are difficult to learn to use and require that the user provide a complete and detailed structure of domain knowledge when building an expert system. The technical team chose an engineering approach rather than a research or scientific approach as the best way to attain the goals of this project. As a result, Enlighten has a complete design. Its program is easily supported and can be expanded to include other target knowledge domains.

1.2 Scope

When this effort was begun, the SBIR project team was aware of the limited time and resources available for system development. Thus the focus was always on what was considered doable, given existing constraints. In this report, the reader will be shown how the design and implementation of the final system incorporates the most important concepts involved without containing extraneous "bells and whistles." These frills, while perhaps desirable for commercialization, are not necessary for measuring the successful implementation of concepts. They may be added in future versions of Enlighten.

1.3. Background

1.3.1 SBIR Phase I

Many of the ideas behind Enlighten actually began in July 1987, with the commencement of the first phase of this SBIR project. At the root was the idea that elimination of a common bottleneck in expert systems development could greatly benefit manufacturing operations. The problem is known as the "knowledge engineering bottleneck" or the "knowledge acquisition bottleneck." It would be desirable to create a product which removed the knowledge engineer filter and placed expert system development directly in the hands of an expert. The expert could then create an expert system based on his or her own knowledge without the additional personnel and time needed for the knowledge engineering process required in first generation expert system tools.

A knowledge-rich methodology was proposed and prototyped during the Phase I effort. Knowledge-rich implies that the system itself has some knowledge which it uses to operate on a user's input. Consider a knowledge continuum where one end is knowledge-free and the other is knowledge-rich as shown in Figure 1.

```
<--------------------------------------------------------->
Knowledge-free                                 Knowledge-rich
```
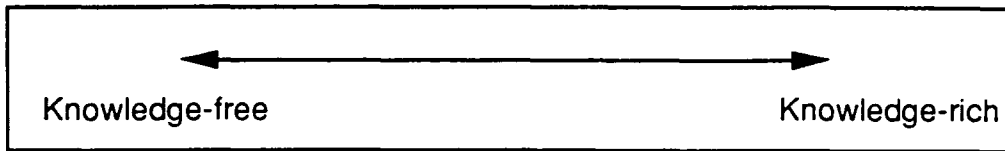
Figure 1.   A Knowledge Continuum

Expert systems, having no knowledge of their own, reside at the knowledge-free end of the continuum. These expert systems do not have any domain knowledge or a semantic understanding of concepts. At the knowledge-rich end, an expert system would have the ability to mimic human thinking, for example, generalizing about concepts or drawing conclusions based on relationships between concepts.

If indeed a knowledge-rich system could be developed, the expert would no longer be required to know how a computer operates or how to program a computer. (The user would not have to discover how to model his or her own thoughts in a program.) Instead, the system, which contained a general model of the user, would accept user input in some natural way, e.g. voice input or handwritten text. The system could accept information the user enters and do something with it, rather than wait for the next instruction from the user. For example, the system would be programmed to organize and map new information, which the user spoke, into the previously existing information. This has been a goal since the inception of the Low Cost Unified Expert System Tool for Manufacturing project.

Phase I showed feasibility of this kind of system. It resulted in a program called CCDEMO which allowed easy input from an expert and gave output in the form of a decision tree. While limited, this effort showed that it was possible to move off the low end of the knowledge continuum. This effort is summarized in the Phase I final report.

1.3.2 SBIR Phase II

Phase II took the knowledge-rich concepts several steps further. Enlighten contains knowledge about knowledge engineering and general knowledge about English language semantics. Information is entered by the user via a natural-language-like syntax described in Appendix A. Enlighten then operates on this information and allows the user to review his or her own knowledge in various ways. Enlighten also allows other users to explore and, possibly build upon, previously created knowledge bases. These previously developed knowledge bases can become part of the knowledge the expert system tool possesses.

By no means does UTC claim to have moved to the knowledge-rich end of the knowledge continuum. However, Enlighten steps away from the knowledge-free end of the continuum. In this sense Enlighten is a second generation expert system building tool.

The lofty goal of a knowledge-rich system loses its luster if there is no useful application for the system. Thus Enlighten has a "use" metaphor which describes its usefulness. The metaphor differs according to the user role. For an expert user role, the metaphor is that of a notebook, in which the expert scribbles ideas to be referred to

later. The notes may seem disjoint to other users and indeed to the expert when he or she looks at the notes several months later, thus Enlighten is a software tool which organizes the notes. Additional notes may be added to the notebook later, and Enlighten's goal is to place these notes appropriately within the framework of prior information. This view of Enlighten shows the tool in a knowledge acquisition and refinement mode.

A second type of user role, a non-expert, may use Enlighten as a dictionary. For example, if an expert has built a knowledge base (notebook) giving details about tool classifications, a non-expert may review that knowledge base, going from one detail to another based on what he or she finds of interest. This mode of operation of Enlighten is that of an expert system, the knowledge delivery aspect of the tool. These metaphors may be somewhat difficult to understand upon first consideration, therefore they will be explored in significantly more detail later in this report.

It is important to reiterate the concept of a unified tool here. Enlighten provides a natural language-like syntax for knowledge acquisition. It organizes the knowledge, finding relationships between concepts where possible, and makes the knowledge available to the user in the original form as well as other forms for exploration.

1.3.3 Overview of Report

Aside from this introduction to the report there are seven additional sections which provide more detailed information about the effort and its results. This report provides background information in Section 2. Section 3 is devoted to discussing the details of Enlighten's design. The result of the implementation is discussed in Section 4. UTC provided prototype versions to two university people, a Ph. D. candidate and a professor, for review. Their responses are in Section 5. Section 6 discusses UTC's conclusions of this effort and potential work directions. The main report is followed by a bibliography and an appendix, which is a brief tutorial on Enlighten's use.

2.0 Project Tasks

The initial Phase II tasks sought to narrow the focus and select measures of success. This set of tasks assisted in limiting the scope of product development for Enlighten. Additionally, a cursory market survey was conducted which highlighted the "do's and don't's" of software development as well as the competition in the area of expert system building tools.

2.1. Identify Applicable Manufacturing Tasks

The first task was to identify applicable manufacturing tasks for which a Low Cost Unified Expert System Toc' would be useful. The design team identified a relatively complete classification of manufacturing tasks. These tasks were compared to the requirements for an expert system. The technical team chose fault diagnostics as the target manufacturing domain.

4

2.1.1. The ICAM Manufacturing Hierarchy

Several years ago, the Integrated Computer Aided Manufacturing (ICAM) program office at the Materials Laboratory at Wright-Patterson AFB, Ohio, developed a comprehensive hierarchy of manufacturing tasks. This hierarchy is shown in Figure 2.

The SBIR project team reviewed this hierarchy extensively to select appropriate manufacturing tasks on which to focus the development. On the initial review of the hierarchy two questions were used to evaluate the lowest node of each branch. The first question was "Does the task involve decision making?" Some tasks involve a high level of skill, but no decision making. Such tasks would not benefit from a decision support or expert system tool. This question culled many performance tasks, e.g., "perform routine maintenance" and "assemble product." These tasks involve skill, but all decisions, such as what maintenance is routine and in what order to assemble the pieces, were made prior to the actual operation, as part of another task (such as planning tasks). No critical decisions related to the task are made at the time the task is performed.

The second question related to the level of expertise required to make a decision: "Does the task require expertise?" This question further narrowed the field of tasks for which an expert system tool could be useful. For example, consider the selection of sources for a material bid. Procedures require the selection of qualified sources for a request for bid. Selecting qualifying sources does involve decision making, however, the decision is based on pre-defined criteria. The "expertise" required to make the decision is limited in this instance. There are many decisions like this throughout the manufacturing process that do not require specialized knowledge or expertise.

Evaluating the hierarchy required assumptions about both the tasks and the methodology used when constructing the hierarchy. One assumption was that some lowest branch nodes described tasks that were outside the scope of the model, whose highest level task is "Manufacture Product." For example, providing people is a requirement to manufacturing. How those people are provided and the decisions and tasks that go into providing people occur outside "Manufacture Product" domain.

No assumptions were made about the environment where the tool may be implemented. For example, availability of expertise is usually an appropriate consideration when building expert systems, whether too much expertise is already available, thus an expert system is frivolous, or no expertise really exists, thus an expert system is impossible. Because of the situational nature of availability and the generic nature of the expert system tool the identification of applicable tasks ignored the issue of expertise availability.
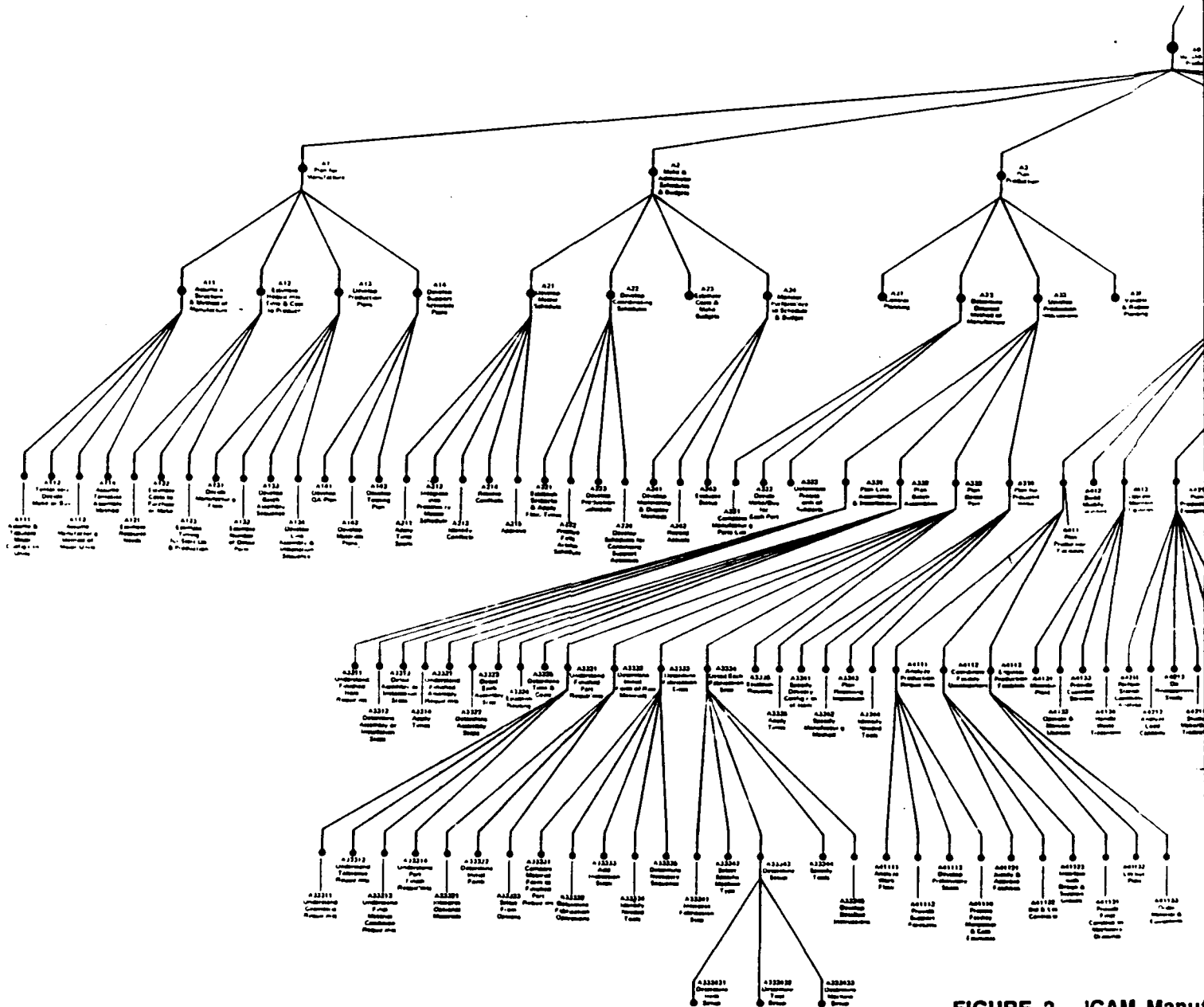
FIGURE 2. ICAM Manu
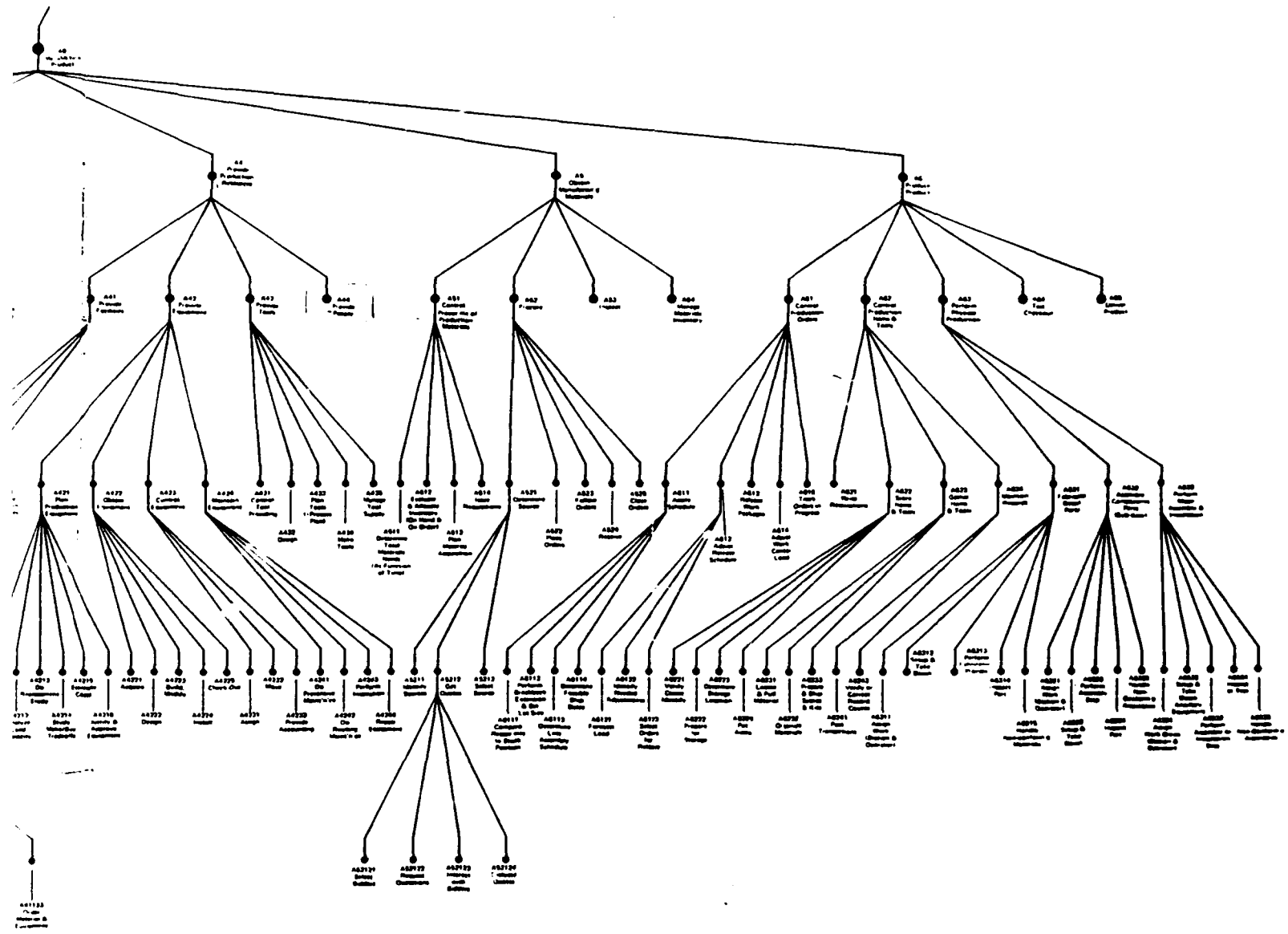
Manufacturing Hierarchy

2.1.2. Classification of Tasks

After applicable manufacturing tasks were identified from the ICAM hierarchy, the next step was to classify or categorize tasks to make general yet meaningful statements about them. The specific breakdown of expert system categories used was chosen from the book A Guide to Expert Systems by Donald A. Waterman. These categories are shown in Table 1.

Table 1. Waterman's Expert System Categories

Control	Governing overall system behavior
Debugging	Prescribing remedies for malfunctions
Design	Configuring objects under constraints
Diagnosis	Inferring system malfunctions from observables
Monitoring	Comparing observations to expected outcomes
Instruction	Diagnosing, debugging, and repairing student behavior
Interpretation	Inferring situation descriptions from sensor data
Planning	Designing actions
Prediction	Inferring likely consequences of given situations
Repair	Executing plans to administer prescribed remedies

Manufacturing tasks from the ICAM hierarchy were mapped into Waterman's categories. The fit was not always perfect. How does one categorize the task "Determine storage location for purchased materials and tools?" Often the text description of the task and its position relative to other tasks were used to help categorize it. The difficulties and misfits can be attributed to forcing one mapping into a different structure: the hierarchy of manufacturing tasks into expert systems classes.

Note that Planning is characterized in Table 1 as "designing actions." This definition is very broad and seems to encompass even Diagnosis and Prediction to some extent. This definition was used when the result of a manufacturing task was a procedure or plan, instead of a physical product. Design was characterized by its direct impact on a physical item.

The initial mapping of tasks to expert system classes resulted in several empty classes. These include Repair, Instruction and Control. Repair of tools required for manufacturing is not manufacturing; it is maintenance. Some routine maintenance and inspection tasks do appear in the manufacturing hierarchy. These were perceived as "action" tasks (some action must be taken to complete the task) which required skills, but not decision-making or expertise. Maintenance and Inspection involve performing a pre-defined list of tasks at specific times. Decisions and expertise to determine tasking and frequency were determined independent of the hierarchy, i.e., outside the high level scope of "Manufacture Product."

7

Instruction falls out because it is also outside the scope of the high level task "Manufacture Product." Instruction expert systems are used to train users and combine the characteristics of the Diagnostic, Debugging and Repair classes of expert systems. The personnel required for the manufacturing process were assumed to be trained, qualified individuals. For this reason, Instruction falls outside the scope of the task and no lower level manufacturing tasks are mapped into this category.

The Control class of expert systems did not initially contain any manufacturing tasks. These tasks were placed in the "action" category because machine control involves planning prior to implementation. Monitoring tasks did not imply any decision making on the part of the system. These also fell into the "action" category. The value of Monitoring expert systems is in removing the human factor from extremely mundane tasks. These were not in need of a decision support expert system tool.

The remaining list of potential application domains was Interpretation, Prediction, Diagnosis, Design, and Planning. Design and Planning were eliminated because they both involved synthesis: aggregation of many parts to create a *new* whole. Current literature in expert system development indicates that synthesis-type tasks are very difficult and a continuing research topic. The focus on development precluded pursuing research-based synthesis-type tasks for the target domain of this SBIR effort.

The Interpretation tasks involved interaction with another system. These tasks require information about the state of a system via sensor data. While not all systems are computerized, a user would most likely want to be able to digitally communicate with other computer systems to gather the sensor data. An automated level of interaction between Enlighten and a sensor system was viewed as beyond the scope of the Phase II effort.

Prediction and Diagnosis tasks seemed very similar in that both required the expert system to determine the state of a manufacturing system given the current description of the system. Both classes of expert systems require information about the current state of the target system. This information did not have to be sensor generated, but only based upon "observable" data. Prediction involved the future state of the system with its associated uncertainty. The lower level of complexity of Diagnosis led to choosing Diagnosis as the area of focus in the manufacturing domain.

2.2. Choice of Application Domain

Diagnosis seemed to fit the metaphor of the user roles quite well. Current state of practice in expert system development required the system developer to create a network of relevant information about possible symptoms and combination of symptoms and correlate them to possible causes. Little effort was made to understand and capture why the expert might pursue a particular train of thought or why a particular answer made the expert change approaches. The goal was to create an expert system for diagnosis of some system. Once the tool was complete, any changes required extensive testing to verify that previous knowledge was not affected by the modifications. This contributes to the current bottleneck in knowledge base development, incorporation of additional knowledge into the knowledge base.

2.3. Survey of Computational Capabilities of Existing Tools

At the outset of this project, the team considered commercial hardware and software tools and techniques available to assist in program development. Several selections were made and they are discussed in the following three subsections.

2.3.1. Computing Platform

One of the objectives of the project was to create a low cost tool. From a large perspective, low cost must include not only the software tool, but also the hardware on which the software must run. Two of the most commonly used, relatively inexpensive computer platforms were reviewed. They are the IBM PC compatible computer and the Macintosh computer. The Macintosh computer platform was selected.

The Macintosh computer has several features the project team believed to be advantageous for the project. One feature is an intuitive, often graphical interface. All Macintosh applications use similar user input devices (e.g., keyboard and mouse) and commands (e.g., menus from which the user makes selections to move about among the computer software). Thus a Macintosh user may simply install a program on the computer and begin to use the program without having to learn new ways to operate the computer software. While restrictive at times to program new software for the Macintosh interface, it has its own model of the user which closely matched the design goals of the SBIR effort for modeling the user.

Memory availability was the second desirable feature for which the Macintosh computer was selected. IBM PC compatible computers have certain limits placed on memory by the operating system and hardware configuration. Macintosh does not. Since the project team was uncertain about the memory that would be needed for the program they were to develop, it made sense to select a machine without the IBM PC restrictions. Enlighten was developed for the entire line of Macintosh computers, however the resulting software currently requires a large memory capacity, on the order of 2 megabytes.

2.3.2. Programming Language

After selecting a hardware platform, the project team turned to selection of an appropriate programming language. The Forth language was selected for its excellent text handling capabilities. Forth has a built-in dictionary and a self-contained parsing capability, thus, these do not have to be built.

Also, Forth is a rather "low-level" language--that is--instructions written in Forth do not require the step-by-step breaking down required in other, more high-level, languages. Therefore, Forth runs quickly, and applications written in Forth, in general, run faster than they would if other languages were used.

2.3.3. Choice of OOPS

After some consideration, the members of the project team decided to use an object oriented programming (OOPS) paradigm. This paradigm maintains certain philosophies in programming and concept-relationship representation, which, as the

reader will be shown later in this report, are key to Enlighten. In addition, object-oriented programming lends itself to easier maintenance than traditional structured programming.

2.4. Study of Existing Expert Systems Developed by Conventional Means

As background for the work which is reported here, conventional expert systems were reviewed to discover expert system techniques that could be used in Enlighten. A general look at expert systems as well as a look at automated knowledge acquisition systems is given below.

2.4.1. Expert Systems

In The Handbook of Artificial Intelligence Volume IV, Barr, Cohen, and Feigenbaum have this to say about expert systems[1]:

> Expert systems are distinguished from conventional programs in several important respects. Although none of the characteristics in the following list are missing entirely from other well-designed software, all of them together describe a distinct class of programs. Note that few expert systems exhibit all of the following five desiderata to the same degree. An expert system is a computer program that:
>
> a. Reasons with domain-specific knowledge that is symbolic as well as numerical (this is what we mean by calling an expert system a knowledge-based system).
> b. Uses domain-specific methods that are heuristic (plausible) as well as following procedures that are algorithmic (certain). [Conventional expert systems usually have an "IF-THEN" paradigm--if something is true, do something else. This allows the system to move through a set of questions logically.]
> c. Performs well in its problem area.
> d. Explains or makes understandable both what it knows and the reasons for its answers.
> e. Retains flexibility.

These desiderata, while defining expert systems, also pinpoint the techniques used in successful expert systems. Each of these five techniques are part of Enlighten, to at least some extent. In short, Enlighten reasons with domain-specific knowledge (both knowledge engineering knowledge and domain-specific knowledge, which may be symbolic or numeric), uses both heuristic and algorithmic domain-specific methods (for user input, manipulating the user input below the surface of the system, and for displaying output), performs well in its problem area (manufacturing classification), understandably lays out all of the information a user desires, and mostly, Enlighten is flexible. The user retains control.

---

[1] Barr, et al. pg 151.

10

2.4.2. Knowledge Acquisition Systems

It was important to review knowledge acquisition systems, since the main thrust of this project was to reduce the knowledge acquisition bottleneck. A sampling of knowledge acquisition systems is reviewed below.

2.4.2.1. Auto-Intelligience

Auto-Intelligence[2] is a knowledge acquisition tool used to build expert systems for structured selection or heuristic classification tasks. The system provides aids in decision making and choice selection based on available criteria. The system uses induction (generalization from examples) to capture knowledge and forms a purely rule-based expert system. The system consists of five main modules:

1. Interview Manager which interacts with the expert and captures knowledge.
2. Structure Discovery System which captures the structure of the knowledge, helps identify the key components, and checks for inconsistency and redundancy.
3. Example Manager which provides bookkeeping tasks.
4. Induction System which creates rules from the data and examples given.
5. Expert System Generator which generates an expert system.

2.4.2.2. KAT: Knowledge Acquisition Tool

KAT[3] provides a variety of ways in which a knowledge engineer can query a domain expert. It operates in three different modes called Clarification, Prediction, and Diagnosis. An expert defines the complete process in an AND/OR graph in the Clarification mode. In the Prediction mode, the AND/OR graph is presented in a flow chart format to prompt the expert to think of additional details for the given information. During Diagnosis, the expert examines the defined process by an automated "step-through" to check for any incompleteness in the knowledge base.

2.4.2.3. SALT: A Knowledge-Acquisition Tool for Propose-and-Revise Systems

SALT[4] is a knowledge acquisition tool that is used to build expert systems in synthesis-type domains such as design. It constructs, rather than selects, a solution. The domain expert is required to provide three types of knowledge: procedure, constraint, and fix, for each value in a design. Procedural knowledge is used to determine configuration pieces and their values. Constraint knowledge describes the constraints on each piece and provides a procedure to determine its value. Knowledge represented in fixes is used to remedy constraint violations.

---

[2] Parseye, K. and S. Murphree

[3] Blaxton, T.A. and C.R. Westphal

[4] Marcus, Sandra, pp81-124.

The domain expert can add knowledge in pieces with no particular order. The acquired knowledge, which is transformed into rules, combined with the problem-solving shell creates a domain-specific knowledge base. SALT checks the knowledge base for completeness and consistency then informs the user if there is missing or inconsistent knowledge. The generated expert system uses a propose-and-revise method. It creates a design by proposing a value for each of the design parameters, then examines the constraints that are applicable to the parameters. The system provides a revision where a constraint is violated.

2.4.2.4. KRITON: A Hybrid Knowledge Acquisition Tool

The KRITON[5] system is a hybrid knowledge acquisition tool that uses many methods to capture different kinds of human knowledge. Automated interview methods are used to capture declarative knowledge. A combination of the repertory grid technique, forward scenario simulation, and laddering is used to acquire knowledge during the interview. The repertory grid technique acquires an expert inputs of knowledge in natural-language sentences and uses a scaling factor to differentiate between them. Forward scenario simulation is a technique in which experts define relevant terms and concepts which are used to determine the steps in problem solving. The laddering technique requires experts to define important concepts of the problem domain. These concepts are further used to lead an interview of the expert. Procedural knowledge is captured using protocol analysis methods. This technique requires that experts think aloud without attempting to rationalize their problem-solving activities. The results of these knowledge elicitation methods are captured by an intermediate knowledge representation system, which uses concepts and relationships.

2.4.2.5. OPAL: A Knowledge Editor

OPAL[6] elicits knowledge using a domain model, which consists of forms. This is a "fill-in-the-blank" method. OPAL is specific to a certain expert system called ONCOCIN, which is a cancer-therapy management expert system.

2.4.2.6. Rulemaster 3

RuleMaster 3 is an expert system shell. It has a knowledge acquisition module which elicits information in the form of examples. It then induces rules based on the examples. The expert system is then used very much like most other expert systems.

2.4.2.7. Conclusions

Automated knowledge acquisition systems using a variety of techniques have been developed. As the reader will discover later in this document, Enlighten uses a number of techniques in common with the reviewed systems, as well as some new

---

[5] Musen, M.A., L.M. Fagan, et. al., pp 83-94

[6] ibid, pp 257-273

techniques. For example. KAT, OPAL, and Enlighten use fill-in-the-blank techniques. SALT and Enlighten both allow the user to input information in any order; however SALT does have an order of output, where as Enlighten allows the user to choose what to review at any time. KRITON and Enlighten both use the concept/relationship approach to knowledge elicitation and representation.

2.5. Identify Capabilities of Approach and Implementation

During the course of selecting the specific domain and outlining the requirements of the tool, which tool capabilities were desired became more apparent. These capabilities can be separated into those which will be implemented during the current phase of product development and those which are beyond the scope of this effort. The short term tool capabilities will be developed, refined and used in Enlighten. Long term tool capabilities represent research level issues and may be explored after the current effort for inclusion in a second version of Enlighten. These capabilities are discussed below as short term and long term tool capabilities. The reason for, and implications of, the classification of the capability in terms of product development are also discussed.

2.5.1. Short Term Tool Capabilities

The short term tool capabilities would be sought within the time available for this Phase II effort. These capabilities should be incorporated into the final product. Each of these capabilities are listed and discussed in terms of their impact on the SBIR Phase II product. This section discusses the capabilities in terms of desires and requirements. The success of Enlighten in achieving these goals in discussed in Section 6.

2.5.1.1. Knowledge Acquisition

At the minimum Enlighten must be able to accept knowledge added by a novice computer user. Both a domain expert and an eventual users must be able to interact with the tool in as natural a fashion as possible. The choice of an English-like syntax is a minimum capability. In the short term, implementation goals emphasized ease of interactions with the computer-based system. The knowledge not only must be acquired, but stored and available for the rest of the system. The technical team decided on a tradeoff: the implementation of knowledge storage and retrieval would be sacrificed to ease the knowledge acquisition process, if necessary.

2.5.1.2. Knowledge Refinement

Knowledge refinement involves the restating of knowledge acquired by whatever means. For example, refinement of freehand notes is an outline. The information contained in the notes was acquired by the reader and translated into another format. This allows the original writer to see any incompleteness, inconsistency, or incorrectness of the notes. It may also allow the original writer a new view of the same old information. The translator may use a different grouping scheme than the author used when taking the notes. This can lead to new insight on the part of the author. Knowledge refinement as a minimum must present the acquired information in a different format than it was originally acquired. Because this is such a fundamental

13

concept in terms of the overall goals of Enlighten, it must be accomplished in the short term.

2.5.1.3. Knowledge Delivery

The product developed at this phase of the SBIR effort was supposed to implement a knowledge acquisition, refinement and delivery tool for expert systems. Enlighten must deliver the acquired knowledge is usable form. A short term capability required of Enlighten is a knowledge delivery mode.

2.5.1.4. Knowledge Engineering Capabilities

Enlighten needs incorporate some characteristics of a knowledge engineer. The goal was to formulate a model of knowledge engineering activities. A knowledge engineer's responsibilities include checking for completeness of the knowledge acquired, verifying the consistency of new knowledge before adding it to the knowledge base, and deriving the relationships between the required inputs and expected system outputs. Incorporating all knowledge engineering capabilities into a program was not the objective; some human actions are beyond present computational technology. From the model those activities which can be reasonably incorporated into a computer program should be used in Enlighten. This capability then involves two tasks: develop a model of knowledge engineering and choose those aspects of the model to be incorporated into the tool. Because of the goals of the tool, this capability must be attained in the current phase of the development.

2.5.1.5. Types of Reasoning

Within the diagnosis class of expert systems, knowledge about diagnosis can be seeded in Enlighten. The types of reasoning relevant to diagnosis needed to be included as a minimum. For example, to proper diagnose the failure in an engine, the position of various components must be known. This requires spatial reasoning. Additional reasoning types, which must be included in Enlighten for diagnosis, are temporal reasoning, hierarchical reasoning, connection reasoning, and structural reasoning. Structural reasoning does not mean that Enlighten is seeded with the knowledge A is part of B, rather it must be seeded with the knowledge of what it means that A is part of B. For example, if A is part of B and B is part of C, is A part of C? This must be resolved in the development of Enlighten.

2.5.1.6. Metaknowledge

The inclusion of metaknowledge is critical to the implementation of Enlighten. The metaknowledge will introduce the necessary vocabulary, reasoning concepts such as temporal, structural, etc. and knowledge engineering knowledge, such as inconsistent information, and other target application domain knowledge. Domain specific knowledge discussed in the types of reasoning above is critical metaknowledge to be included in Enlighten. A longer term goal would be to allow Enlighten to be seeded with knowledge about different target application domains such as prediction.

14

2.5.2. Long Term Tool Capabilities

Long term tool capabilities are those capabilities requiring further research in areas beyond the scope of this phase of the project. There may be several reasons to avoid these areas at this time, including the length of time required, impact on the tool, and priority of the project goals. The use of the tool by manufacturing experts will highlight additional features that may be required of the tool. These may become part of the tool capabilities in future versions.

2.5.2.1. Extension to Other Domains

As discussed in Section 2.1, the project must focus on a specific application domain. The tool, however, can easily be extended to other domains. Testing Enlighten in another domain will be a high priority goal after the current development effort. The prediction domain may be the next area explored due to the similarity to diagnosis. Manufacturing system Control and Monitoring are other possibilities.

2.5.2.2. Machine to Machine Interfaces

The current development effort requires a human for input of information and interpretation of output. This dependence on humans maintains a largest possible user community with a reasonable programming effort. While completely automated interfaces (computer-to-computer) are desirable in some cases, they are highly specialized. The initial effort will develop for a single type of input -- human textual input -- and a single type of output -- textual output to be interpreted by a human.

The machine interface can be either input or output. Machine communication of input can be used by a domain expert or final user to access information. This information may be in the form of a large database. This would require a specialized query language interface. The tool could also interact with signal generator to monitor a process. In both cases this enhancement lessens the input requirements of the human user of the expert system.

The second type of machine to machine interface is communication between the tool and a machine for process control. This type of communication can be used to control processes or machine tools like an automated machining cell. Both types of communication involve development of custom interfaces which are not part of the goal to develop a generic tool.

2.5.2.3. Alternative User Interfaces

The proposed natural-language-like interface requires the user to use textual input. While this method maximizes the size of the potential user community, it is not the best interface method for all domains. In some domains, experts may better express information with symbols or graphics. A long term goal is to expand the user interface to include non-text-based input and output.

15

2.5.2.4. Future Natural Language Research

No natural language research was undertaken due to the extremely complexity of the natural language problem and since existing technology proved adequate. In the future, this are may be explored to enhance the user interface.

2.5.2.5. Future Mathematical Implementations

Computational capabilities will not be developed during the Phase II effort. Several elemental manipulations, however, are necessary to create a truly useful tool, i.e., comparisons. For example, two numeric diameters of drilled holes may need to be compared to determine which is larger. While necessary these capabilities are not achievable in the time available to complete the development.

Moving beyond the initial scope of this project and extending it to the likely computing power available to the average user in a few years, it can be envisioned that a system which is capable of evaluating mathematical functions via a natural language dialog, and perform reasoning upon the equations and algorithms extracted from the user's language input will exist. Such a system would be able to perform manipulations and reasoning on data either entered by hand or retrieved from other sources.

3.0 Development Objectives

The development of Enlighten was driven by three primary objectives. The first was the choice of theories appropriate for the representation of knowledge at a level deeper than that of traditional expert systems. A wide variety of representation theories were surveyed including semantic nets, frames, and first and second order predicate logic. The key criterion in the selection process was that the representation had to be robust enough to capture a deeper semantic meaning of knowledge than simple IF-THEN relations of conventional expert systems.

The second objective was the choice of an appropriate user interface that would simplify the capture of knowledge in comparison to traditional expert systems. The two primary methods for the input of knowledge investigated were graphical and textual. The most general form of communication is textual, though not always the most appropriate. Seeking to create a useful tool for all levels of users required specifying a generic and appropriate interface. In this case a text-based user interface paradigm was chosen.

The final objective was to develop a model of the user that would result in a product that the user found easy to learn and simple to use. A model of the user refers to the the typical way that a user views the world. For example, if the user finds it natural to describe the world in terms of textual information organized into outlines, the model of the user would be an organizational framework based on text. In a conventional expert system, there is no model of the user. Rather, the user is forced to learn the model of the software or, more accurately, the designers' model of the software. While every system will require some amount of learning, it was one of the goals of this project to develop an expert system that was easier to use and learn by basing its design on a model of typical users.

3.1 Development Approach

The development of Enlighten can be separated into three distinct phases: theory, design and implementation. The theoretical phase focused on the creation of a complete theory for the design. This is the stage at which the technical team sought philosophical theories which would satisfy parts or all of the goals. These theories were then integrated to synthesize new theories which completely satisfied the goals and objectives of the effort. These new theories distinguished the possible from the impossible.

The design phase was concerned with integrating the various theoretical ideas gathered in the first phase into a unified system design. This phase addressed questions of how the theory was to be implemented to create a software tool. More importantly, it was during the design phase where basic engineering design principles were applied - the goal was the development of a product that implemented the theories, and not a research tool that proved the theories. This is the first point at which tradeoffs were identified. In general, these tradeoffs were between taking the "purist" approach and implementing the theories completely, or taking an "engineering" approach and creating a useful product that may not be completely compatible with the theory.

The implementation phase focused on the actual development of the code to implement the design. The primary concern at this point was developing an appropriate implementation of the design that successfully balanced the two scarce resources - memory and processor speed. Every implementation level decision involves a balance between these two resources. For example, a choice to develop a portion of the code using arrays as opposed to linked lists may be faster, but it consumes more memory. A poor implementation can have a direct impact on the user in terms of usefulness and ease of use.

The following sections will describe the three phases of the development of Enlighten in detail. The emphasis is on not only what decisions were made, but why they were made.

3.2 Theoretical Background

The choice of theoretical basis for Enlighten was driven primarily by the need for a representational strategy that would allow the semantic content of a statement or clause to be captured and applied. This differs from the approach taken in conventional expert systems where there is no attempt to capture the semantics or meaning of a statement. Rather the statement is simply assigned a value of true or false, depending on its logical interpretation - no attempt is made to analyze the statement at a deeper level or to gather additional information or knowledge. The primary theory investigated as a means of attaining these goals is abstract set theory.[7,8] The notion of categorizing information through the use of sets closely

---

[7] Copi, Irving M.

17

approximates one model that a typical user has of reality. Additionally, the ability to represent relations between objects as sets further strengthens the applicability of this approach.

3.2.1 Sets and Relations

As the name implies, abstract set theory is a formal theory for manipulating sets or classes. The term *set* is a primitive term describing a collection or aggregate of objects. These objects are referred to as the members or elements of the set. More accurately, it is said that a membership relation holds between a set and its elements. A membership relation is symbo. zed by the Greek letter "$\in$." It is important to note that the membership relation is not transitive. That is, if "Ohio" is a *member* of "The United States," and "The United States" is a member of "The United Nations," it does not hold that "Ohio" is a member of "The United Nations."[9]

The above notion may be confusing in view of conventional set theory. In conventional set theory, if A = { 1, 2, 3 }, and B = { A, 4, 5 }, then all of the elements of set A are members of set B. This is not the case with abstract set theory. In order for this to be true, it must be stated explicitly that A is a *subset* of B.

A set can be defined in either of two ways: by *extension,* or by *intension.* An extensional definition of a set would be a definition based solely on an explicit list of *all* of its elements. An intensional definition is a specification that states the property that must be true of all of the elements of the set.[10] The intensional definition is often referred to as the meaning of the set. For example, the following statement is an extensional definition of a set: { 2, 4, 6, 8, 10 }. Note that it is not possible to know what common property binds the members of this set. This information would only be available in the intensional definition. One possible intensional definition for the set is "even integers between 1 and 10."

There are a number of basic operations for creating new sets from other sets and specifying relationships between sets. These are union, intersection, complement, difference, subset, and proper subset. A detailed explanation of these operators can be obtained from any introductory text on set theory. For the purposes of this project, the primary operator of interest is the subset operator. By definition, if "A" is a subset of "B," then all of the members of "A" are members of "B." Note that the converse is not true - all of the members of "B" are not members of "A." The importance of the subset operator is that it gives rise to a *type hierarchy* .

A type hierarchy is the structure formed by a set and its associated subsets, and any subsets associated to the subsets, etc. The principal feature of a type hierarchy is the notion of *inheritance.* Inheritance is a mechanism whereby any members of a set's

---

[8] Whitehead, A. N. & Russell, B.

[9] ibid, 1

[10] Sowa, J. F., pp. 368

subsets are also assumed to be members of the set. In this case, if "A" is a subset of "B," and "B" is a subset of "C," then all of the members of "A" and "B" are also members of "C." The basis for inheritance is the transitive nature of the subset operator, in direct contrast to the intransitive nature of the membership relation.

In addition to the notion of a set is the notion of a relation. As is the case with membership, a relation is a definition that holds between two or more sets. In general terms, a relation refers to linguistic entity that relates two or more objects in a sentence. For example, in the sentence "a car requires fuel," the term "requires" is the relationship holding between the object "car" and the object "fuel." Symbolically, this relationship is often depicted as requires(car, fuel). An important observation is that the order of the objects in the relation is important in accurately representing the relation. It would be incorrect to state that "fuel requires a car" means the same thing as "a car requires fuel."

A relation is defined *extensionally* in a manner similar to that of a set. In this case, the relation is a set consisting of all of the *ordered pairs* of entities that stand in that relation. In the previous example, the relation "requires" would be a set containing the ordered pair <car, fuel>. An additional member of that set could be <fish, water>, implying that "a fish requires water." The use of ordered pairs is a direct result of the fact that the order of the objects standing in the relation is important.

It is often advantageous to speak of the *domain* and the *range* of a relation. The domain of a relation is the set of all first coordinates of the ordered pair and the range of a relation is the set of all second coordinates of the ordered pair. It is possible to provide a conceptual interpretation of these two sets for a given relation. For example, given the statement, "a car requires fuel," the range could be interpreted as "the set of things that a car requires." In this case, "'fuel" would be a member of the set. In a similar manner, the domain could be given the interpretation "the set of things that requires fuel" and "car" would be a member of the set.

3.2.2 User-Driven Search

From an end-user's perspective, a conventional expert system functions as an investigator who asks a series of questions in order to arrive at some conclusion. Like its human counterpart, the software expert system "investigator" is in charge - it determines what questions to ask and in what order. The people being interrogated have little control in the matter. They simply provide information. While the question and answer sessions may be made less painful through use of various computer interface gimmicks, e.g. providing a set of predefined responses from which to choose, the underlying philosophy is still that of a *computer-driven search*.

The computer-driven approach to problem solving used in almost all conventional expert system can be very frustrating to the end-users. The burden is placed on them to read and interpret the multitude of questions and provide an appropriate response. If the knowledge engineers who developed the expert system perceived the situation differently than the users (i.e., their model of the user was incorrect) the users may be faced with questions that are difficult to answer correctly. This can lead to a situation where the user realizes *after* a response has been chosen that the response was incorrect. Unfortunately, since the expert system is driving the search, there is typically

no way for the user to go back and change a response in order to follow a different line of reasoning.

An alternative approach to the computer-driven search strategy is a user-driven search. The primary idea behind a user-driven search is that the user is in control. He or she decides *what* to investigate and *when*. The expert system still executes its internal inference engine in order to arrive at some conclusion, but the user has more control over this process.

This second approach has a number of benefits from a user's perspective. First, because they are able to follow different lines of reasoning at will, the users will be more inclined to explore alternate possibilities that may be equally probable. Second, the ability to logically browse through the expert system knowledge base would enhance the utility of the system as a tool for training. And finally, by allowing users to gracefully recover in situations where they did not respond correctly to a question, the expert system will appear more user-friendly, and thus more useful.

3.3 Design Goals

There were two major drawbacks identified with conventional expert system technology: the "brittleness" of conventional representational strategies and the need for an intermediary, i.e. the knowledge engineer, to acquire and translate an expert's knowledge into the syntax of the expert system shell. One of the major design goals aimed at alleviating some of the brittleness inherent in conventional expert systems was to develop a representational strategy that captures the knowledge behind an expert's heuristics.

A situation will often arise during the delivery phase where the expert system is unable to identify any suitable solutions to the problem. This occurs any time that the user encounters a situation that was not originally envisioned by the expert. Even if the situation closely resembles one of the scenarios described by the expert, the expert system is unable to generalize the expert's "rules of thumb" or heuristics in an attempt to find a solution to the problem. This lack of generality is a common problem with conventional expert systems known as "brittleness."

Traditionally, the expert's heuristics are converted to a collection of IF-THEN clauses that can be applied to solve a limited range of *known* problems. Unfortunately, there is no attempt to encode the other body of general knowledge that an expert uses to solve *new* problems. By capturing and organizing this body of *non-heuristic* knowledge, a user could actively apply it to solve new problems for which no heuristics exist.

Because of the complexity of the more powerful conventional expert systems, a knowledge engineer is typically required. The function of the knowledge engineer is to acquire and elicit the knowledge from an expert and then convert it into the form required by an expert system shell. More often than not, the knowledge engineer must be a skilled programmer in addition to having specific training in the development of knowledge-based systems.

The task of converting an expert's knowledge into a form acceptable to the expert system shell is an error prone process. The subtleties of verbal communication will almost always result in some error in interpretation. Because of this, the knowledge engineer must provide a prototype system for the expert to test. Upon locating errors or deficiencies in the expert system knowledge, the expert must then attempt to explain what the problem is, and the knowledge engineer, repair the fault. This is the stage of refinement in a conventional expert system.

The delivery of an expert system refers to the actual use of the completed system by the end-user. The end-user will rarely if ever be the original expert, but rather someone that is less skilled in the particular area of concern. The typical way that the end-user interacts with a completed expert system is through a computer-directed question and answer session. The end-user has very little freedom in this process.

The second design goal was designing an expert system shell that could be used *directly by the expert*.s By removing the knowledge engineers from the process and putting the experts in charge of directly putting their knowledge into the shell, the bottleneck in knowledge acquisition would be greatly reduced. Experts would be *more in control of their own schedules, and management of the development of the expert system*. More importantly, the resulting knowledge base would more closely model what the experts intended since there is no error in translation through an intermediary.

The critical consideration in removing the knowledge engineer from the development loop was the creation of a shell that is simple enough for non-programmers to use, yet powerful enough to create complex expert systems. To achieve this, a *syntactic interface* was designed. The key feature of this interface was that it provided both the expert and the end-user with a constrained, natural language-like syntax for interacting with the system. While the syntax is constrained in the number of syntactic forms it can distinguish, it does not limit the ability of the user to represent their non-heuristic knowledge.

3.4 The Design

The design of Enlighten can be divided into five major elements: representational structure, grammar, representational philosophy, inference mechanism, and user interface. The representational structure is used to physically represent the non-heuristic knowledge entered by the expert user. The grammar provides a constrained, natural language-like syntax for entering the knowledge into the representational structures. The representational philosophy describes how the meaning behind the grammar is interpreted and converted into the representational structure. The *inference mechanism allows the information in the structure to be retrieved and interpreted*. The user interface provides a simple way for a user to enter, manipulate, and retrieve non-heuristic knowledge. The subsections of 3.4 provide detailed explanations of the first four elements of the design. Section 4.0 discusses the user interface as it is implemented.

3.4.1 Representational Structure

The primary element of the representational structure is the *category*. From a theoretical standpoint, a category is functionally equivalent to a set. It is used to represent the basic components of the representational grammar, i.e. actions, properties, events, and things (objects), as discussed later. Figure 3 provides a structural view of a category and its various components.
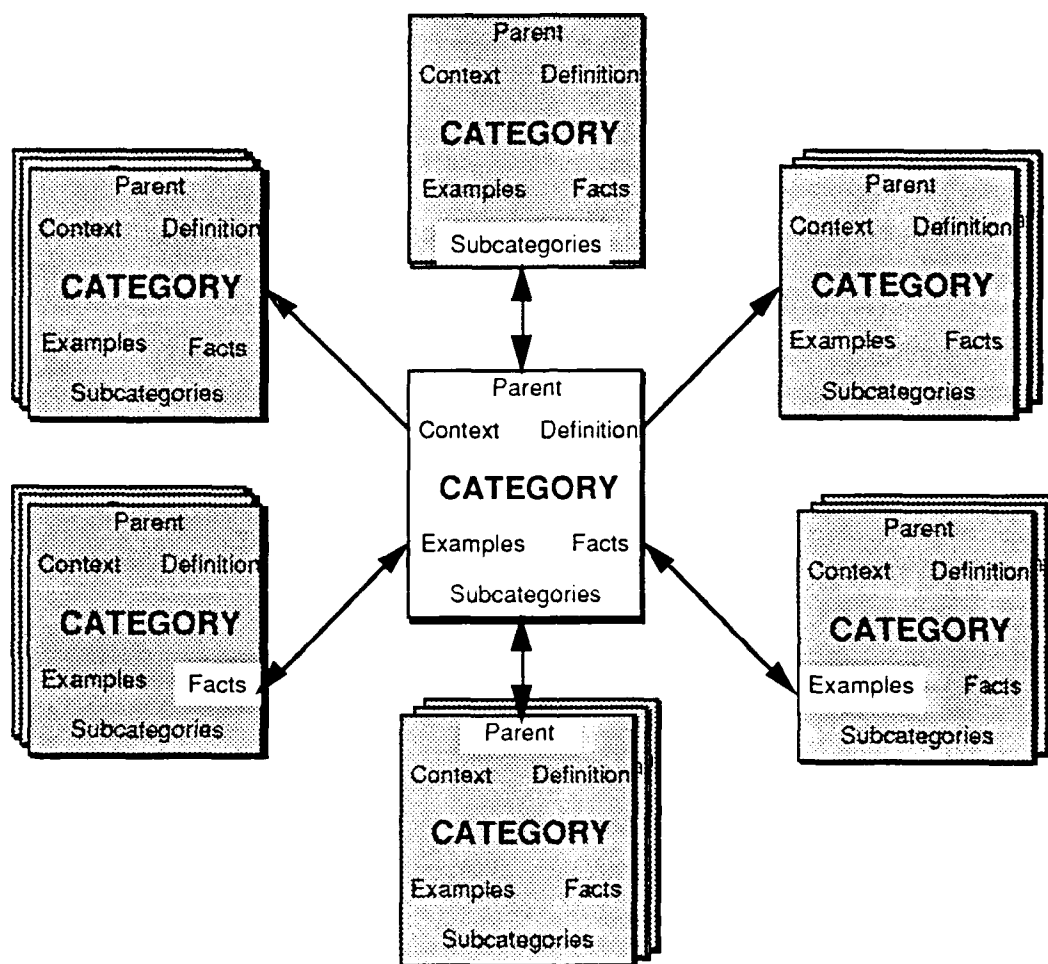
Figure 3. A structural view of a category

A category is composed of seven basic parts or relations: names, context, subcategories, parent, examples, facts, and a definition. The names of a category are the *syntactic* means by which a category is referenced. The context of a category works in conjunction with its names. It is possible for multiple categories to exist, each having the same name, but defined in a different context. For example, the term "tanker" when referred to in the context of a truck refers to a different "thing" than when referred to in the context of an airplane.

While many in the AI community handle this problem of reference by resorting to hyphenated names, e.g. truck-tanker and airplane-tanker, this approach places the burden of semantic interpretation on the user. Context, in Enlighten terms, allows the

22

user to naturally refer to "tanker" while the system works at determining which tanker is being referenced.

The parent and subcategories of a category are the theoretical equivalent of superset and subsets respectively. Thus, the parent of a category is defined as the set containing all of the members of the category. By definition, a category may have only one parent category. Also, it is possible that a category may have no parent defined for it. In contrast, a category may have an unlimited number of subcategories. The subcategories simply provide a method for *categorizing* the examples of the category. Assume that category "A" contains the integers 1 through 10 as examples, i.e. members. It is possible to create two subcategories, the first containing the numbers 2, 4, 6, 8, and 10, and the second containing 1, 3, 5, 7, and 9. The obvious semantic interpretation for these sets is "even integers between 1 and 10" and "odd integers between 1 and 10."

The examples of a category are theoretically equivalent to the members of a set. The same duality that exists between a subcategory and a parent also exists with examples. In this case, the dual of examples are facts. Assume that the category 'knife' is an example of the category "weapon." Viewed from the perspective of the category "knife," it has a related fact category "weapon." That is, a fact about a "knife" is that "a knife is a weapon" and from the perspective of "weapon" an example is "knife."

Recall that the names of the category provide a syntactic method for referencing a category. The definition of the category provide a semantic or intensional method for referring to a category. As the name implies, the definition of a category defines the intension or meaning of the category. It typically describes the primary distinguishing characteristic that all of its members have in common. In addition to this, it also provides information that defines the category's parent.

More important however is the semantic interpretation of a category containing these various relationships to other categories. The parent of a category can be interpreted as describing a *generalization* of the category. For example, given the category "dog" and its parent category "animal," "animal" would be described as a generalization of "dog" or simply, "a dog is a animal." Note that the phrase "is a" can have many different interpretations.[11] In the case of a category's parent and subcategory links, the "is a" phrase can be interpreted to mean that one category is a specialization of the other. When viewed from the perspective of the parent, its subcategories are said to be *specializations*. Thus, "dog" is a specialization or type of "animal." In this sense, the parent and subcategories give rise to a traditional type hierarchy.

The semantic interpretation of examples is straight forward - they are simply elements or members of the category. For example, consider a category "weapon" with the following example categories: "gun," "knife," and "dog." In this case, "gun," "knife" and "dog" are examples of a weapon. Stated simply, "a gun is a weapon," "a

---

[11] Brachman, Ronald J.

23

knife is a weapon," and "a dog is a weapon." In this case, the "is a" phrase is being interpreted *differently* than it was with the subcategories and parent. It is interpreted here to mean *is a member of.* [12] This relationship is different from specialization in that "is a" is not interpreted as *is a type of* as described above.

The definition of set membership (not to be confused with the definition of a category) requires that all of the examples for a category share some common characteristic. Thus, when viewed as a fact, a category is describing some characteristic or feature of its associated example category. Consider the previous example: while each of the three example categories are apparently different types of things (i.e. they are not directly related via a subcategory or parent relation) they still all share some common characteristic. In this case, the characteristic is that of "weapon-ness" or the quality of being a weapon. From the viewpoint of the "gun" category, the category "weapon" appears as the fact "a gun is a weapon," and is therefore a characteristic of a gun. Here again, the same "is a" phrase is used to describe yet another type of relationship. In this case, it can be interpreted as *is something that has the characteristic.*

The meaning of the definition of a category has two parts. First, it defines the category's parent. From this perspective, it defines what the category is a specialization of. The second part of the definition is a fact describing the characteristic that all of the examples of the category (members of the set) have in common. For example, if the definition for the category "weapon" was "a tool that is used in combat," the first part of the definition says that "a weapon is a tool" or that the category "weapon" is a specialization of the category "tool." The second part says that a weapon is something that has the characteristic "is used in combat."

There is no actual meaning ascribed to the context of a category. Rather, as previously described, the context is used to disambiguate one category from another having the same name.

The representational structure described provides a framework for representing semantic information. The next section describes the representational grammar.

3.4.2 Grammar

The grammar of Enlighten is based on the notion that the goal of language is to assert some *fact* about some *object*. To accomplish this, all languages provide some method for specifying which of the multitude of objects in the world is being referenced. Similarly, the language provides some means for specializing the factual information being asserted.

Consider the statement "a drill with a tip that is made of carbide is required to drill holes in titanium." At its most general level, the statement is expressing the idea that "a drill is required." Through the use of prepositional phrases, e.g. "to drill," and restrictive phrases, e.g. "that is made of carbide," the English language permits the

---

[12] ibid,4

general references to be specialized. Thus, the object being specifically referred to is "a drill with a tip that is made of carbide," and the specific fact is "is required to drill holes in titanium."

An analysis of English statements at a *semantic* rather than *syntactic* level indicates that a wide variety of concepts can be conveyed as specializations of factual assertions about general categories of objects, as previously suggested. Thus, the primary focus in developing the grammar was to decompose the English language into a set of semantic components and a set of rules for manipulating these components or primitives.

The semantic primitives identified were objects, facts, and events. An object refers to some general conceptual category. For example, "dog," "ocean," and "furniture" are all objects referring to some general category of things. Together with the second syntactic form, facts, an object forms a statement. Syntactically, an object corresponds to the subject of a sentence or phrase when used within a statement.

Conceptually, a fact describes some characteristic or feature of an object. Syntactically, a fact corresponds to the predicate of a statement. There are two basic syntactic forms of a fact. The first form which is the conceptually easier to understand contains two parts: an action and an object of the action. For example, in the statement "a car requires fuel," the fact "requires fuel" is comprised of the action, "requires," and the object of the action, "fuel."

The second syntactic form for a fact conceptually assigns a value or parameter to some property of the object. Syntactically, this form of the fact consists of a single word that directly follows the object it is associated with. In many cases, the word may be preceded by "is." Consider the following statement: "coal is black." Conceptually, the value "black" is being asserted about some unstated property of the object "coal." Knowing what the meaning of "black" is allows one to determine that the unstated property is actually "color." From a syntactic viewpoint, the property phrase is simply a shorthand notation for the expanded phrase "the color of coal is black." For the statement "a wheel rolls," the semantic interpretation is not as obvious. It is important to realize, however, that semantically there is *some* property that is being assigned this value. One possible interpretation for this unstated property is an "action" leading to the expanded phrase, "an action of a wheel is rolls."

In addition to objects and facts, there is a special class of linguistic entities that semantically correspond to events. Syntactically, an event is composed of either a single action called the *event action* or an event action followed by an object. Valid event actions are any action that has an "ing" ending, e.g. drilling, running, killing, sawing. Thus, the phrase "curing composites" corresponds to an event whose action is "curing" and object is "composites." There is also a special case of events that follow the preposition "to" but do not end in "ing." Thus, in the phrases "to drill holes," "to run," and "to examine fossils," "drill," "run," and "examine" are all semantically defined as event actions. Note that syntactically, an event will *always* follow a preposition.

Although events are syntactically defined as transitive verbs, semantically they are defined as a type of object and are therefore another semantic primitive. Because of

25

this, it is possible to speak of facts about an event as is the case in the statement "curing composites requires the use of an autoclave." In this case, the fact "requires the use of an autoclave" is being asserted about the event "curing composites."

```
<Statement>        →    <Object><Fact>
                   →    <Event><Fact>

<Fact>             →    <Action><Object>
                   →    <Value>
                   →    is<Value>

<Event>            →    <Event Action>
                   →    <Event Action><Object>
```

Figure 4. Summary of semantic primitives

The majority of English statements are rarely as simple as the previous examples. Typically, the references to objects and their associated facts are complex statements that focus the attention of the reader on a specific object or a specific fact. Consider the following: "a car with chains on the tires moves in the snow." In this case, the phrase "a car with chains on the tires" refers to a specific car. Similarly, "moves in the snow" is a statement that refers to a specialization of the general fact "moves." This notion of *conceptual specialization*, where a general concept is further specialized to refer to a specific concept, provides a powerful technique for semantically decomposing a complex statement into smaller parts or concepts. The phrases that modify the concepts will be referred to as *conceptual modifiers*. Figure 5 summarizes Enlighten's conceptual modifiers.

```
<Specialized Object>   →    <Object><Prepositional Phrase>
                       →    <Object><Restrictive Phrase>

<Specialized Action>   →    <Action><Prepositional Phrase>

<Prepositional Phrase> →    <Preposition><Object>
                       →    <Preposition><Event>

<Restrictive Phrase>   →    <Restriction><Fact>
```

Figure 5. Summary of conceptual modifiers

There are a number of syntactic indicators that aid in determining what part of a phrase is functioning to specialize a concept. The two most common syntactic indicators are prepositional phrases and restrictive phrases. A prepositional phrase is syntactically defined as a preposition, e.g. of, in, to, on, with, above, and into, followed

26

by an object, referred to as the object of the preposition. For example, "of a car" is the prepositional phrase that modifies "a part" in the statement "a part of a car." Notice that the phrase restricts or specializes the reference "part." Prepositional phrases may modify actions, objects, and events.

A restrictive phrase is defined as a restriction, e.g. "that" or "which," followed by a statement. For example, in the statement "a car that runs on alcohol," the restrictive phrase would be "that runs on alcohol." As with a prepositional phrase, the restrictive phrase functions to specialize the reference to "car." Restrictive phrases may only modify events and objects.

3.4.3 Representational Strategy

The representational strategy behind Enlighten is closely tied to both the representational structure and grammar. It provides a conceptual framework for interpreting and organizing non-heuristic knowledge. This conceptual framework in essence defines the computer model of the user. From typical users' perspectives, it describes their model of the world. More importantly, the strategy provides a method for mapping the concepts conveyed by the grammar into the representational structure.

In the representational strategy of Enlighten, everything in the world, both physical and abstract, can be represented as a category. As a means of organizing such a collection of abstract concepts, everything is assumed to belong to one of four possible base categories: Actions, Events, Properties, and Things. Conceptually, the Actions category contains any concept that describes an activity. Thus, "run," "eat," "kill," "cure," etc. all are valid members of the Actions category. The Events category contains examples of concepts that describe *actions being applied* such as "running," "eating," "killing," and "curing." The Properties category contains examples that conceptually describe characteristics of things. For example, "weight," "color," "height," and "speed" are types of properties. Finally, the Things category contains everything else not falling into one of the other three categories. Its intended examples include both physical and abstract concepts, including objects.

The strategy adopted for representing knowledge within the framework is based on the theory of conceptual specialization described in Section 3.4.2. This theory suggests that a general concept can be specialized giving rise to a more specific concept. In other words, given a category representing some concept, it is possible to represent a specialization of the concept as a subcategory. This results in the generation of a complex hierarchy of concepts that has a direct correlation to the grammar. The following sections describe the strategy taken in representing the *semantic primitives of the grammar*: objects (or things), facts, and events.

3.4.3.1 Representing Objects

In its simplest form, an object is grammatically equivalent to a single word. Thus, "dog" is a grammatical representation of an object, which maps directly to a category in the representational framework. The grammatical label, in this case "dog," is used as the name of the category. Once added to the representational framework, any *grammatical* reference to "dog" is an equivalent *conceptual* reference to this category.

Within the framework, objects are analogous to Things and are appropriately added as examples to the Things category. Figure 6 shows a sample conceptual representation of an object in Enlighten's representational framework.
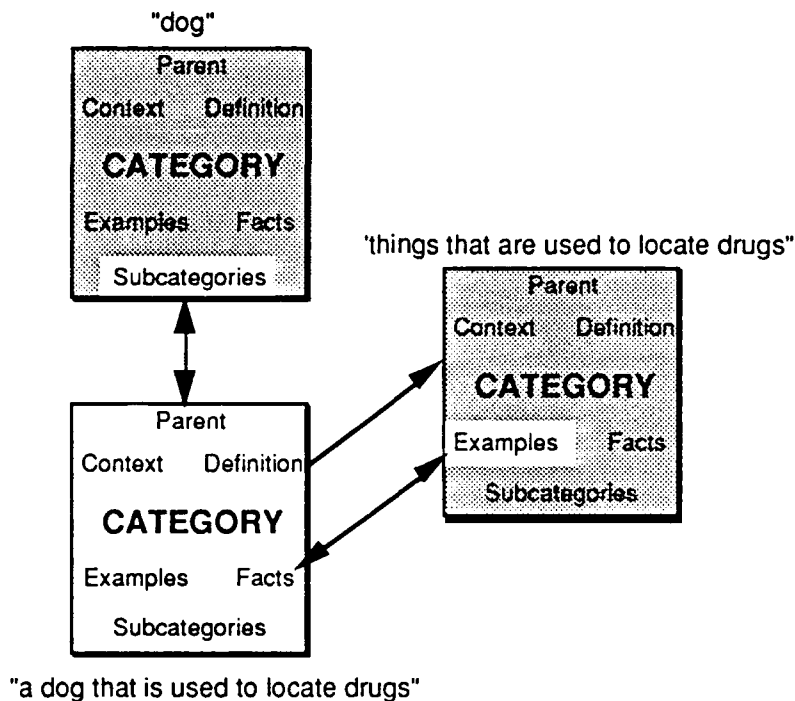
Figure 6.   Conceptual representation of an Object

Recall that a category (set) can be defined either extensionally or intensionally (see Section 3.2.1). The process of conceptual specialization actually provides an intensional definition of the subcategory being referenced. In this case, the characteristic that differentiates the single subcategory from any others is captured in the phrase functioning as the specialization. In the preceding example, the restrictive phrase "is used to locate drugs" defines this distinguishing characteristic. From the standpoint of the subcategory, this characteristic, which is also represented as a category (see the following section for additional details), is a fact of the subcategory. That is, subcategory is related to this characteristic category or fact category via a fact relation. In addition to this, since it is the primary fact that defines the subcategory, the two are also related through the definition relation.

An important point needs to be made with regards to how these subcategories are referenced grammatically, identified, or named. Unlike their base category that has a single-word name as an identifier, the subcategories defined via conceptual specialization require the entire grammatical phrase to completely specify which subcategory is conceptually being referenced. It is however, possible through the list of names associated with a category to provide a shorter identifier for the category. In the above example, the category "a dog that is used to locate drugs" could be given the name "police dog." Any subsequent references to the grammatical label "police dog" would result in the category "a dog that is used to locate drugs" being identified since the two grammatical references are conceptually equivalent.

28

3.4.3.2 Representing Facts

Grammatically, facts can take one of two forms: an action followed by an object (a relation), or a value of an unstated property of the object (a property). These forms are represented within the framework differently.

3.4.3.2.1. Representing Relations

In one form, a fact describes a relationship that holds between two objects: the object of the action, and the object for which it is being asserted. This relationship can be defined extensionally as a set of ordered pairs containing the objects that stand in the relation (see Section 3.2.1). From the perspective of the representational framework, the representation of a relation would require the creation of a new primitive in Enlighten - the ordered pair. This method is viewed as inadequate since there is no conceptual basis for representing relations in such a manner.

In contrast to the extensional definition of a relation, the definition of the *domain* of a relation provides a logical conceptual interpretation. More importantly, the members of the domain are *objects* rather than ordered pairs. For these reasons, relations are actually represented in terms of their domain within the representational framework. Consider the following fact: "require fuel." The domain for this relation conceptually would be the set of "th   that require fuel." This would be defined within the framework as a conceptual specialization of the Things category and would be grammatically referred to as "things that require fuel." The object for which the fact is asserted would be an *example* of this fact category.

At a more detailed level, the conceptual specialization of a fact actually generates *two* subcategories. The first subcategory has the Things category as a direct parent. It conceptually corresponds to a very general category of objects that have the specified relation to something. For the relation "requires," this subcategory of Things would be referred to as "things that require something." The motivation for including this abstract category in the representational framework is the ability to assert facts about "things that require something." These facts can be viewed as knowledge about the relation requires and, through the inference mechanism described in Section 3.4.4, will automatically be inherited by any object that stands in this relation, i.e. a subcategory.

The second category, which corresponds to the actual range of the relation, is actually a subcategory of the first subcategory. That is, it is a conceptual specialization of the other subcategory with the object of the action providing the defining characteristic. In the previous example, the subcategory "things that require something" could further be specialized to generate a second subcategory that would be "things that require <object>." For the fact, "requires fuel," this subcategory would be "things that require fuel," which is the range of the relation. Further, for an assertion of the fact that "a car requires fuel," the category "car" would become an example of the category "things that requires fuel."
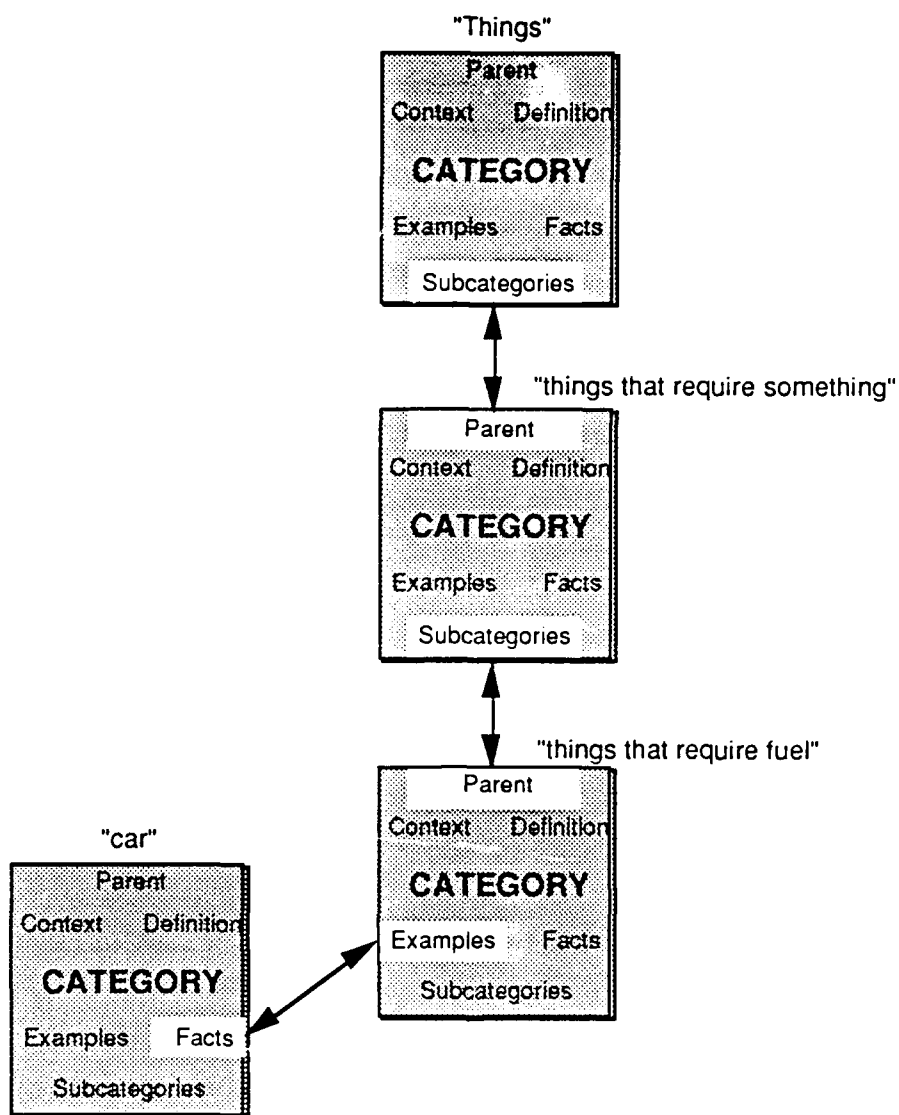
"Things"

Figure 7. Conceptual representation of a relation

To summarize, all facts within the representational network are represented as conceptual specializations of the Things category. Conceptually, these categories correspond to the range of the relation. The examples or members of these fact categories are the objects for which the fact is asserted.

3.4.3.2.2. Representing Properties

A property, the other grammatical form of a fact, is conceptually concerned with associating a *value* to some unstated property of the object of the fact. The difficulty in representing properties within the structure stems from the fact that the property is usually unspecified. In the example "coal is black," the actual property of coal that has the value black is not specified. This statement is a "conceptual shorthand" for the more complete idea, "the color of the coal is black." In general, the expanded form is "the <property> of <object> is <value>." This example is depicted in Figure 8.

30
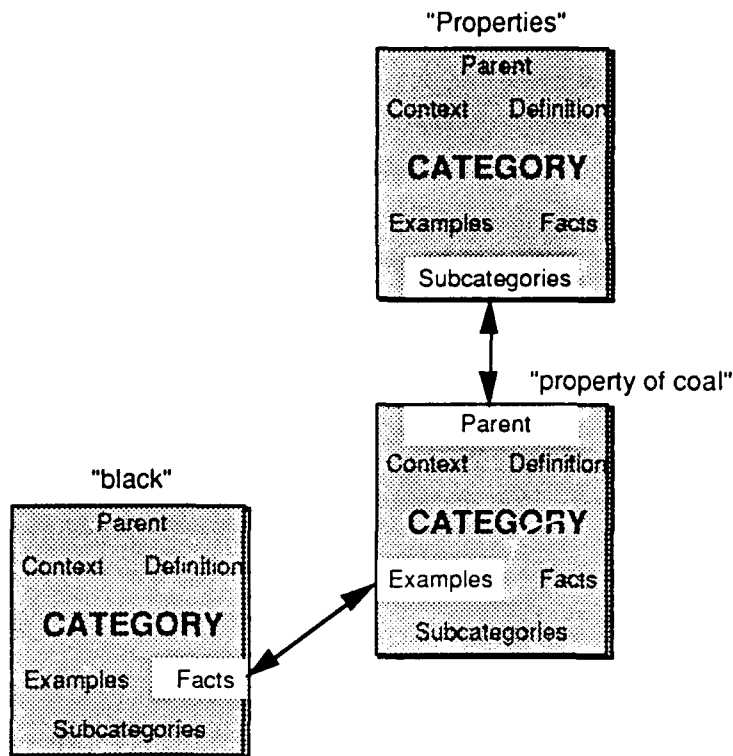
Figure 8. Conceptual representation of a property

With the above notion in mind, it is should be evident that to accurately represent the conceptual structure of a property requires a specialization of the base category Properties. This conceptual specialization is accomplished via the prepositional phrase "of <object>" where object refers to the object being assigned the value of the property. For the phrase "coal is black," this will be a subcategory of Properties referred to as "a property of coal." Continuing, the final link in the representation is to add the value, black, as a new example of "property of coal."

3.4.3.3 Representing Events

Grammatically, an event consists of, as a minimum, an action. In this case, the event is simply represented as a category that is an example of the base category Events. In many cases, the action will be followed by an object. Although it appears that conceptually the object specializes the action, a further analysis of common syntactical forms indicates that this form of the event is actually abbreviated. Consider the example "curing composites" and a counter example, "the curing of composites." Note that conceptually, these statements are identical. In this case, the latter statement is the expanded grammatical form.

With the above discussion in mind, it is apparent that an event is simply a conceptual specialization of the Events category. The named event is a new category which is an example in the base category Events. For the above example, this is the new category "curing." "Curing of composites" is defined As a subcategory of "curing." The definition and facts relations of the category for the named event are associated with the new category. Note that conceptually the definition and facts of the "curing of

31

composites" category can be abstracted to a category, "things of composites," which simply indicates that the examples of the category are in some way related to "composites." The figure below provides the complete representation for this form of the event.
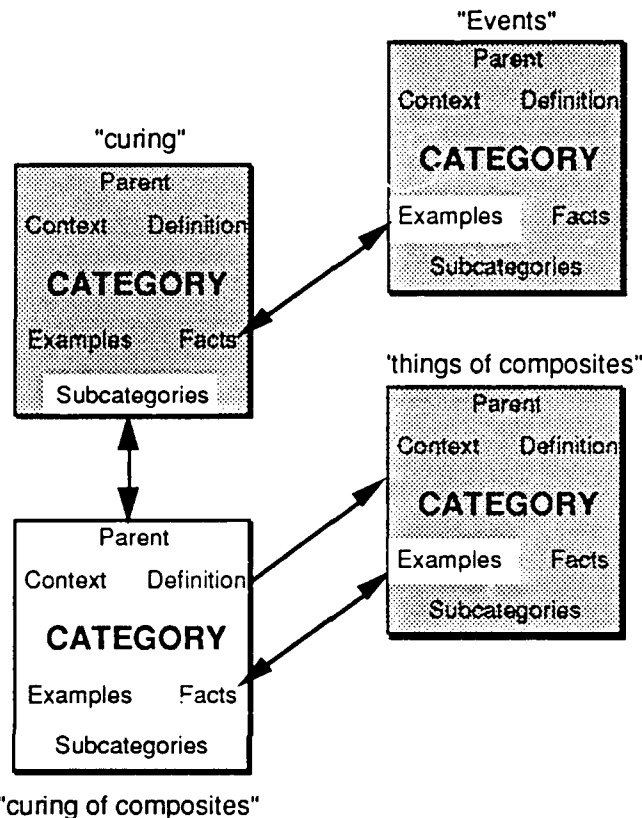
Figure 9. Conceptual representation of an event

3.4.4 The Inference Mechanism

The inference mechanism of Enlighten provides a method for retrieving knowledge. It defines which paths through the representational framework of related category information can be validly taken to arrive at some conclusion. There are two types of information that can be indirectly inferred via the inference mechanism: examples and facts. Note that examples and facts can also be obtained directly by viewing a given category. This is not considered as a method of inference but as simple data retrieval. Indirectly inferred facts and examples are information that is obtained from categories that are indirectly related to some category.

Additional examples are inferred via a category's subcategories. This type of inference is permitted since, by definition, all of the examples of a subcategory are also examples of its parent category. Further, since the subcategory relation is transitive, this process can be continued for any subcategories of the subcategories.

The inference of facts about a category occurs in two ways (shown in Figure 10). First, categories inherit the facts of their parents. For example, if "animals requires

32

food," and a dog is a subcategory of animals, then it can be inferred that "a dog requires food." Once again, because of the transitive nature of the subcategory and parent (supercategory) relation, additional facts can be inherited from the related parent categories. Note that the text reflects the exact wording used in Enlighten. Proper English grammar has not been encoded in this version.
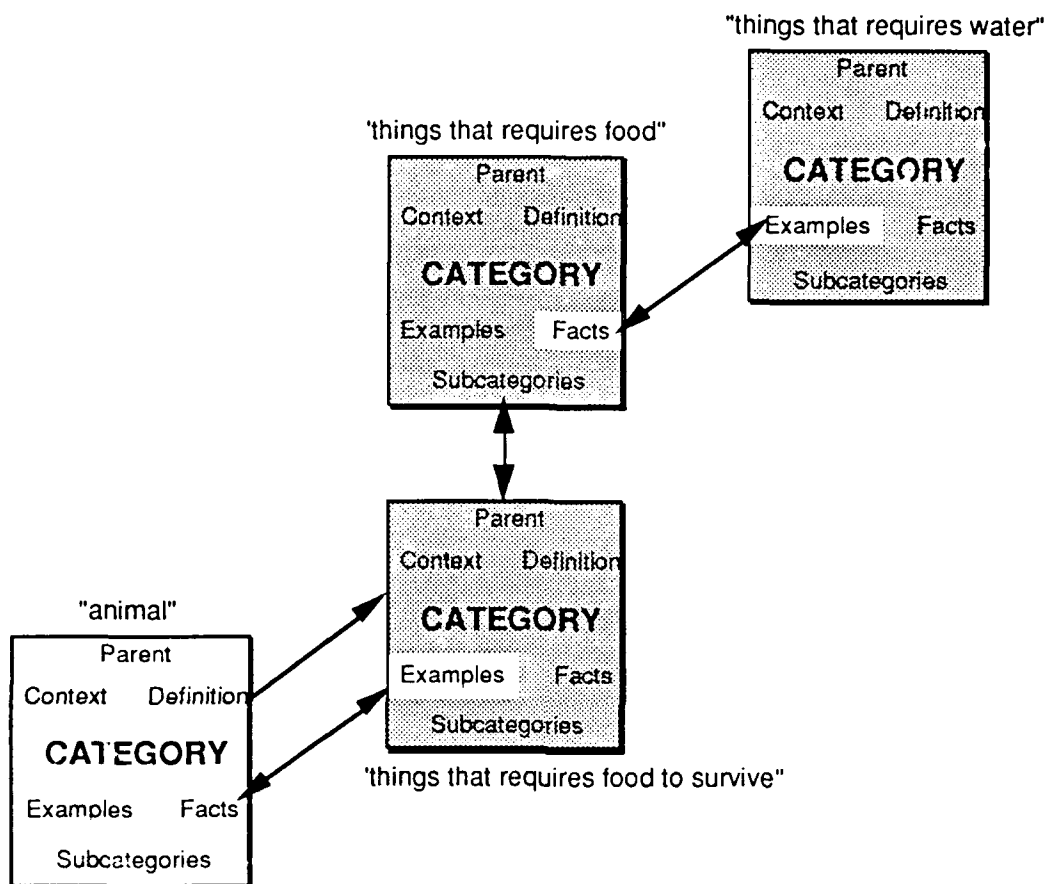
Figure 10. Inheritance of facts via fact categories

Secondly, the examples of a category inherit the category's facts. Recall that when the following statement, "an animal requires food to survive" is asserted, that the category "things that requires food to survive" is created and "animal" is added as an example of the category. Because Facts are the analogue of Examples, "requires food to survive" becomes a fact about "animal." Further, note that the category "things that requires food to survive" can have its own set of facts such as "require water." Because "animal" is an example of "things that requires food to survive," it can directly inherit all of its facts. Thus, in addition to the fact "requires food to survive," the fact "requires water" would also be inferred through this other fact.

It is important to realize that because the fact "requires food to survive" is actually represented as the category "things that requires food to survive," additional inferences can also be made though the parent categories of this category. Thus, if the parent category of "things that requires food to survive" is "things that requires food," any facts from this category would also be inherited by "animal" *and,* any facts

33

from this categories facts would *also* be inherited. This technique provides a means for rapidly creating a very rich network of general knowledge.

4.0 Enlighten, The Result

4.1. General Overview

The user cannot be completely unaware of the intended use of a software tool. Like a homeowner's hand tool, the computer user must know that something is to be gained by using the tool and have some idea about how the tool will achieve the desired result. For example, a homeowner knows that a hammer is used to hit objects, usually with more force than can be exerted with a hand. The possible uses of a hammer are left to the homeowner to explore. The same is true of computer software tools. However, more often than not, the model of the software must be made explicit to motivate the potential users before they are left alone to apply the tool.

This section attempts to explain the intended use of Enlighten. After a discussion about potential uses of Enlighten, the user roles and their probable interactions are described. The discussion is in terms of metaphors which describe ordinary tasks normally accomplished without any software tool. The discussion includes possible motivations for each user group to use Enlighten.

Given Enlighten's objective, a unified acquisition, refinement, and delivery expert system tool, the software has three modes of operation: acquisition, refinement and delivery. Because the objective also includes the creation and delivery of an expert system, it accommodates two user roles: knowledge "inputters" and knowledge users. Traditionally, knowledge inputters were knowledge engineers who translated an expert's knowledge and encoded it in the expert system tool. In Enlighten, knowledge engineering is accomplished directly by the expert .

Experts will use the system primarily in the acquisition and refinement modes. Experts might also want to use Enlighten as a tool after the knowledge has been added as a data retrieval resource. Non-experts will typically use the system in a delivery mode only. Each mode of operation of Enlighten is discussed below.

4.1.1. Input Methods

Interactions with Enlighten are text-based. The expert captures the concepts and their definitions in words which must be typed into the system. The non-expert user queries the expert system by requesting more information about specific concepts entered as typed words or phrases. This reliance on text provides the most generic interface mode for all applications. Text is used in all disciplines, even those with highly formalized symbols such as chemistry and mathematics. Text is the one common denominator among manufacturing disciplines.

This choice of a text-based interaction also created additional design considerations, most importantly the syntax. To allow non-programmers to interact with Enlighten, a set of natural language-*like* syntax templates were defined. (See Appendix A for listings of these templates.) These templates force users to rephrase proper English into Enlighten's more limited syntax. All of the limits on the allowable

syntax take into account the potential for user errors and the effects of syntax rule violations to the integrity of the knowledge base.

These syntax limits in no way compare to the prevalent use of hyphenated words to capture the object's semantics. The hyphenated word "drilled-hole" relies on the reader to derive its semantic meaning; it has no semantic meaning to the system. While this method of encoding semantics is used in most expert system shells, it places the burden of semantic consistency on the user; all users must interpret the word combinations consistently. The potential for misinterpretation of information is very high, degrading the usefulness of current expert system tools. Word hyphenation, while allowed, is not necessary to capture semantics in Enlighten. More importantly, it captures semantics through definitions as opposed to such hyphenated labels. The design of Enlighten sought to allow common English forms (templates) to be used to name and define categories so as to maximize the naturalness of its syntax.

The design is based upon a model that assumes people think of, and organize, information in categories. To capitalize on this model, Enlighten allows a user to enter information about a category in the form of facts, examples and subcategories. Enlighten uses this information to build relationships or associations between categories forming a complex interrelated network of knowledge.

While Enlighten's user interface is dependent upon typing information on the computer, the design attempts to eliminate some of the effort required. One feature is that typing the subject of each fact is eliminated. For example, the category "weapon" could have the facts:

is used to inflict harm on people
is capable of killing people
is used to protect the user
causes fear

in each case the subject is assumed to be "a weapon." Appendix A discusses the details of adding information to Enlighten.

4.1.2. Summary Methods

Enlighten summarizes and displays the information it has gathered using indexes. Enlighten includes a core index consisting of the four base categories: Things, Actions, Properties, and Events. New indexes may be created by the user to view the relationships created by Enlighten.

An Enlighten index is similar to a document outline. The context of the category and its associated categories are displayed in a "hierarchy" with the most general category flush left and each restricted category indented and below its parent. The list of categories at any each level, for each category is alphabetical. The order in which the expert entered the categories does not determine their sequence of the index.

The associations or contexts of the objects and actions are displayed in the indexes. For example, "hole that is drilled" is a subcategory to "hole." This context is maintained by having "hole that is drilled" indented under the concept "hole." This

context is very different than a "tooth that is drilled," even though both have the same restriction of being drilled. The index for objects would be displayed for this set of object as:

```
hole
     hole that is drilled
tooth
     tooth that is drilled
```

Special indexes can be generated to summarize all facts, examples, or subcategories of a category. These indexes include information directly associated with the category, and information inherited from related categories. For example, an index of facts for the category "hole that is drilled" will appear as:

```
opening
     fact1
     fact2
     •
     •
     hole
         is round
         fact2
         •
         •
         hole that is drilled
             has sides that are smooth
             fact2
             •
```

This example assumes that "opening" is a parent of "hole" and the above restriction of hole is also in the the knowledge base. This format allows the user to see the level at which a fact was asserted. The facts at the "opening" level apply to all examples, all subcategories of "opening," and all subcategories of subcategories of "opening." This index method shows the user information that may not have been apparent. For detailed discussion of how Enlighten gathers this information see Section 3.3.4.

4.2. Knowledge Acquisition

The first step in the development of an expert system is knowledge acquisition. Knowledge acquisition is the gathering of relevant information about a specific topic. Traditionally, this information is acquired by a knowledge engineer who usually interviews the expert or presents sample situations for the expert to solve. The information is recorded and later massaged into an expert system shell.

In contrast, knowledge acquisition in Enlighten is performed by the expert through the process of introspection. An expert can use Enlighten to capture and organize this

knowledge through a process similar to taking notes. Starting with a single category or topic, the expert provides facts about that category. Writing details reminds the expert of other information which is then added to the category. This process continues until the expert believes that the knowledge is complete enough to function as an expert system.

In Enlighten the expert controls the knowledge acquisition process. The expert is free to add information in any order and at any time. Thus the user is free to add facts, examples and subcategories, but is not *forced* to do so. However, the user model of Enlighten assumes that including facts, examples and subcategories adds richness to the knowledge base not available in traditional expert systems.

An expert's use of Enlighten goes beyond the metaphorical note-taking. While input method is not as flexible as note-taking, Enlighten allows the expert to put the information in any order. As the expert adds new knowledge, Enlighten creates organization transparently. Unlike a notebook, Enlighten automatically incorporates the additional knowledge in the correct place and context, in contrast to a paper-based system of cluttering the margins of a notebook with later observations.

The expert is not forced to follow a strict procedure to add knowledge to the system. Nor is he or she forced to maintain a model of the expert system representational framework to create or modify a knowledge base. The elaborate network of rules in a traditional expert system can be scrambled with the addition of a single new rule, rendering the system useless. This is not the case when adding or modifying knowledge in Enlighten.

4.3. Knowledge Refinement

The acquisition phase should be followed by a refinement phase. Traditional knowledge base development separates knowledge acquisition and refinement. Over a period of time, the knowledge engineer uses the acquired information to create a prototype knowledge base. Once the prototype has been created the expert and knowledge engineer work together to evaluate the prototype knowledge base for completeness, consistency, accuracy, etc. This process usually results in the expert getting a "different" view of the knowledge, which often exposes nuances that he or she had not previously seen.

Enlighten allows the expert to refine the knowledge base in a user-driven fashion. The expert can review the acquired knowledge for completeness, consistency, accuracy, etc. at any point in the acquisition process. Additionally, Enlighten provides a summary view of the acquired knowledge in the form of indexes. These indexes highlight where additional information may be needed to complete the knowledge base, indirectly prompting the expert to add more information.

By generating an index Enlighten traverses the hierarchies and displays the relevant relationships *between* the categories. This may result in new, formal knowledge for the expert. In this way Enlighten also facilitates refinement by showing a different view of the expert's knowledge.

4.4. Expert System Delivery

Enlighten's delivery system allows for the potential of two distinct user roles: the domain expert and the non-expert. The non-expert is treated as having no prior knowledge of the domain or, at most, minimal knowledge. The domain expert can use the system as a non-expert to test it, and as an expert to explore the domain with a "knowledgeable partner." The objectives and methods of each user role differ considerably.

4.4.1. Non-Expert User

A complete Enlighten knowledge base can be viewed as an expert's body of knowledge about a topic. The non-expert explores this collection of information to achieve some end.

The non-expert user interacts with Enlighten in two ways: through category windows and indexes. Category windows show all of the information gathered about a single category. Indexes summarize information about the entire knowledge base or an individual category, specifically the facts, examples, or subcategories of the named category. Non-expert users can use both types of feedback to get information from an Enlighten expert system.

For the non-expert user, Enlighten must already contain knowledge, i.e. the "notebook" must have already been filled. The non-expert user uses the notebook as dictionary or encyclopedia to look up the desired information. The user can explore the knowledge base from general to specific topics. The user chooses an initial topic for which Enlighten displays all of its associated facts, examples and subcategories. From there the user determines which, if any, of these new categories are relevant to the objective of the search. By pointing and clicking on desired categories the user navigates through the information contained in the knowledge base. The associations between the categories are those of the expert; the search path is chosen by the non-expert. Appendix A shows an example of navigating through the knowledge base.

Cross references in this "dictionary" are built-in. With clicks of the mouse the user can quickly move between categories exploring meanings and relationships with other categories. Categories are connected via facts, subcategories, and examples. Enlighten shows these relationships as cross references between categories in the indexes and in both category windows. Whenever the user wishes to know more about any category in the knowledge base, Enlighten's interface allows easy navigation through the references. See Figure 3 for a summary of interrelations between a category and its subcategories, facts and examples.

The level of detail explored for any given category is user-specified. The user controls what information about any category is shown. The user can explore relationships between categories using only the index windows or actually look at the facts, subcategories, and examples of the categories using the category windows. Indexes summarize information for both types of users, expert and non-expert.

The delivery mode of an Enlighten expert system is similar to the interaction between a non-expert and an expert in a problem solving or information gathering

exercise where the non-expert is leading the discussion. While this is not often true in expert/novice interactions, this approach can lead to more effective communication in answering the questions of the relative novice. Human experts can lose patience with the continuous question of a novice or resort to lecture after assuming the direction of the questioning. Enlighten maintains the control in the hands of the current user. This is part of the user model encoded in Enlighten.

4.4.2. Expert User

The expert user can have two levels of interaction with a resultant expert system: testing and use. The expert user tests the expert system by playing the role of a non-expert user. In this case the expert tests the knowledge base for completeness, consistency, etc., modifying the information as necessary. More importantly, the expert can simply use the expert system as an on-line tool for storing and retrieving information. It is this ability that allows an expert to benefit from the use of Enlighten.

4.5. Inquiry Capability

The user, either expert or novice, can force Enlighten to search the knowledge base in three ways: through categories, indexes and facts. The most general way to inquire information from Enlighten is through a category. The user can retrieve facts, examples and subcategories simply by "asking" Enlighten for the category by name. This allows for basic "what" type questions e.g. what requires fuel, to be asked, through restatement of the question as a category, e.g., things that requires fuel.

The second method is through the generation of indexes. Index generation, see Section 4.1.2, gathers information from throughout the knowledge base.

Enlighten can respond to a specific query about the validity of some fact. Is a statement asserted or not. This is a separate function in Enlighten as a menu option. When queried, Enlighten searches in the context provided of the fact; it can use synonyms and inheritance to validate the information in the query. The query system of Enlighten is one of the most limited aspects of the tool in its current version. The power of the information contained in an Enlighten knowledge base is currently untapped.

5.0 Review of Enlighten

The prototype version of Enlighten was subjected to use by two university-associated people. Ms. Evans of The Ohio State University developed a knowledge base about wave theory and music to test Enlighten's ability to distinguish different categories with the same names. Dr. Deep of the University of Dayton, Ohio evaluated Enlighten for its potential as a commercial product. The goal of this portion of the project was to validate the concepts as implemented.

5.1 Summary of Review 1

This section gives a reaction to Enlighten by one reviewer. The first reviewer, Ms Evans, gave both positive and negative comments about the software. Aside from a number of software "bugs" (non-working or strangely working features), the reviewer's

comments are generally positive. Typing concepts into the system did result in easy knowledge input. However, the format required by Enlighten was sometimes difficult to follow. Some information had to be worded oddly to fit into the system. The reviewer found the indexes to be quite interesting and liked the idea of combining different concepts in a knowledge base, but also found that some concepts were parsed or connected inappropriately. Inappropriate connections may have been due to the reviewer's lack of familiarity with the limited syntactic templates allowed by Enlighten. Overall, the reviewer liked Enlighten and said she would use it to explore thoughts and concepts, and to look for new connections she had not previously considered.

The feedback from Ms. Evans was used to find and address the inappropriate linking of categories. The delivered version of Enlighten addresses these bugs as discussed in Section 6.0. Also, the review pointed out several deficiencies in the context feature. The implementation of the context feature was frozen and left for follow-on work.

5.2 Summary of Review 2

Dr. Deep's review is very extensive. This review discusses his comparison of Enlighten to commercially available expert system shells as well as other expert system development tools. His review is from the perspective of Enlighten as a personal information manager (PIM) with a delivery mode as an expert system. Major portions of this section are direct quotes (set off as block quotes) from Dr. Deep's response.

> Enlighten's structure can be likened to Hypermedia where entities are linked in a non-linear fashion. Hypermedia allows the user of such a system to peruse the information contained in any order or non-order. However, Enlighten stands against an entrenched competition in the commercial market place where several established hypermedia packages have grab significant market share. All of these other hypermedia packages have also established a niche such as personal calendar (Lotus Agenda), data base (Apple HyperCard), and outlining (Symantec More). Enlighten will have to approach a very specific niche and address that niche market better and, preferably, before any other product. The general approach taken as a white collar software tool is too general and sure to fail in a competitive market.

> Enlighten is based on the Macintosh philosophy of the user determining what to do and when, rather than relinquishing control to an inference engine. In some sense, this is an illusion; for the most part, the computer seizes control for shorter intervals under the Macintosh philosophy. Enlighten may afford random access to the information thanks to hypertext, but most production knowledge-based system can fire rules in virtually any order in response to a given situation. Thus the approach taken by UTC to develop Enlighten on the Macintosh does not necessarily give it the upper hand in user interface and user control.

> In terms of an expert system shell, Enlighten again is up against stiff competition. Many vendors are now developing front-end knowledge

40

acquisition modules. Also, almost all significant expert system developed today operate in concert with other forms of knowledge (data bases, spreadsheets, graphics, communications packages, forms, word processing, report generators, links to higher order languages, and hyper text applications for the HELP modules, personal information managers, etc.) Today's expert system shells exhibit the following properties:

- Implementable on relatively inexpensive PC's,
- Computationally efficient comparable to dedicated computers,
- Allow multiple modes of chaining (forward, backward, mixed),
- Allow several hundred rules (limited by computer memory),
- Permit bridging feature to external computer structures,
- Available run-time-only versions,
- Provide an integrated synergistic environment, and
- Allow for easy information input.

Uses of Enlighten could traditionally include a word processor, a data base, an appointment calendar, an outliner, a text retrieval, an idea generator, etc. Or, untraditionally, the most outstanding use of Enlighten would be to initiate an alter ego information source which never forgets, and accepts and organizes the bits and pieces of information fed it daily. Many times humans try to recall something and search through piles of books, articles and papers. With an implementation of Enlighten, they would never have to do it again, and they would probably want to feed Enlighten a whole lot of information. Again, this demands easy, friendly, flexible information input and output.

Enlighten's raison d'etre should be to flexibly and easily accept and manipulate information from users as the information occurs to them (order independent). Then it would be desirable to have Enlighten contain an inductive component which could not only accept examples from users, but which could distill the examples into efficient summaries or rules. Enlighten presently retrieves the bit and pieces as a more coherent whole that when the bits and pieces were entered.

There are several kinds of knowledge: declarative, semantic, procedural, episodic, etc. Enlighten uses declarative knowledge or just plain facts (e.g., the ball bounces), derives semantic knowledge from the syntax, because it uses the relationships between concepts and facts, procedural knowledge, because it follows the action verbs, and some episodic knowledge. Enlighten must not only take in information , but must also get rid of information at the user's request.

It may be argued that expert systems are brittle, but in fact expert systems do not profess to go beyond their knowledge bases. Since Enlighten does not parse for semantics, there are a lot of questions up front that make Enlighten brittle in some sense. Enlighten knows what it doesn't know, but could well exhaust the patience of a laic user unaware of the difficulty of imparting commonsense to a computer program.

How does Enlighten handle contrary and contradictory statements? By giving both on retrieval requests? For example, the wall is blue; the wall is red; the wall is not red. It would be nice if Enlighten could flag contrary knowledge, but it only returns products of the links.

Enlighten presently cannot handle adjectives or adverbial modifiers, thus severely limiting user description and calling for stilted input. There may be artificial ways of getting around this limitation; for example, A CAR HAS FOUR TIRES can be formed into THE NUMBER OF TIRES ON A CAR IS FOUR; but there is an implied price to pay.

How does Enlighten address fuzzy knowledge? For example, icy roads are slippery. Rick is very bright. Smoking can cause cancer. Simply by recall may not be sufficient. For example, BIRDS FLY. OSTRICHES ARE BIRDS. OSTRICHES DO NOT FLY. OSSIE IS AN OSTRICH. Does OSSIE fly?

Finally, Enlighten's novelty is its inheritance hierarchy, but this hierarchy puts the responsibility on the developer and the user to construct -- no easy task -- and one that involves abstraction. For example, here are some questions that need to be answered:

- What objects and relations are important in the domain?
- What kind of queries will be made of the system (links)?
- Will the facts change over time (like due dates)?
- Is there a well-understood taxonomy for the information?
- Can the structure be broken, changed, and mended at will?
- What happens if CATEGORIES are misused or misidentified?
- Is the structure computationally efficient?
- How will the knowledge be organized?
- Will categories be nested , but only accessed from within?

Considering the relatively short development time thus far for Enlighten, what has been done is well within the reach of most software houses which do development efforts for a living. Thus competition is and will remain keen. Although Enlighten's idea of mingling hypertext with hierarchy may be considered novel, both features are well developed and understood in and of themselves. The existing data structure technique called "frames" exploits inheritance and hierarchy as well. Object-oriented applications are based on nested inheritance, encapsulation (chunks), and polymorphisms (links) and are often combined with hypertext.

Enlighten could be easily learned with context-sensitive help. Enlighten will need to make good use of color, Dr. Deep has reviewed (by reading) over a dozen packages on the market and to prepare Enlighten for the commercial market is a formidable task. A niche for Enlighten needs to be determined so that if commercial preparations are in order, the preparations can be tailored to the niche. Dr. Deep would assign a very high risk to the endeavor based on the existing commercial products. In any case, since Enlighten is being billed as a white collar personal information manager (PIM), it is best to have a very gentle learning curve, i.e., immediately easy to use. A completed Enlighten should allow users to easily and

flexibly input their own knowledge and retrieve it in a useful way so that the users need only query their alter ego structures.

5.3 Response to Review 2

In response to Dr. Deep's comments, UTC believes he has several valid points and identifies areas of future work. However, UTC also disagrees with other points. Dr. Deep states that the choice of the Macintosh platform does not provide any advantage over other computing platforms. The whole Macintosh operating system philosophy is based on user knowledge and control. All of the possible actions, in both the operating system and all applications, are readily available for the user to select; nothing need be memorized. The value of this type of interface is obvious from the market trend over the last 5 years; the market share for graphical user interface-based operating systems (GUI's) has dramatically increased. Enlighten takes advantage of the Macintosh's interface and philosophy of making things easiest for the user (not necessarily for the programmer) and minimizing the potential and severity of mistakes.

The increasing availability of knowledge acquisition modules on current commercial expert system shells does not necessarily bring these tools any closer to the expert. No matter what the shell, backward and forward chaining systems require an extensive network of rules. To optimize this type of system, the user must understand the way the rules are encoded in the software. The potential for creating spaghetti networks is very high. The potential of a novice modifying an existing, functional knowledge base system by adding a single rule is low because of the network structure of current rule-based systems. Enlighten is *not* rule-based rather it is a hierarchy of related categories based on the information added by the expert. The expert has no need to understand the hierarchy to begin to add information to Enlighten.

Dr. Deep has valid criticism in how Enlighten handles conflicting and contradictory statements. This is part of the incompleteness of the implementation. It is part of the design to flag these as they are added to the knowledge base, or at a point when the expert asks for all of the conflicts, allowing the expert to eliminate the incorrect statement, leaving the correct one. Deletion of existing knowledge is an incomplete aspect of the current implementation making resolution of conflicts impossible.

Towards the end of his review, Dr. Deep asserts that any development house could have developed Enlighten. True, any development house worth its existence can code a complete design. Development of innovative design is the objective of the SBIR program. It is an opportunity for a company to get started as a development house. With a new and innovative approach to an existing problem, the solution can benefit the Air Force. Enlighten offers a new approach to gathering and using information as an expert system which both the author (expert) and non-author can use.

43

6.0 Results and Conclusions

6.1 Overview

The primary goal of this SBIR program was to the development of a low-cost tool that would allow experts in a given field to easily and quickly construct useful expert systems. Further, it was essential that the expert was capable of accomplishing this task without the assistance of a knowledge engineer and with little or no programming experience. From a design perspective, the most critical goal was viewed as the ability for an expert to use the "shell" directly without a knowledge engineer as an intermediary.

It is UTC's view that current rule-based expert system shells require too much detailed information regarding the actual inner workings of the shell and force the user to "codify" their knowledge in a form that is not as natural as was once believed, i.e. rules. As an alternative to this rule-based approach, a more natural, "facts about objects" approach was chosen. With this approach, the experts input their knowledge in terms of facts about specific topics, e.g. "a composite is a material that is manufactured in an autoclave". Further, these topics can be organized in a hierarchical fashion by the expert. This not only aids the expert in the development process, but also facilitates retrieval of the knowledge by a different, non-expert user.

From a non-expert user's perspective, traditional expert systems exhibit a "computer-driven user" philosophy. It is the expert system that drives the user through a series of questions requiring a set of specific answers; the user has little or no choice in the particular line of inquiry chosen. Worse, it is often impossible for a user to retract a previous answer or compare the expert system's outcome if they had chosen a different path in the inquiry. Thus, rather than allowing the user to freely explore the various solutions to a problem, the expert system leads the user to its best solution.

In contrast to this, UTC chose to design a shell that put the user in charge of the inquiry process, i.e. a user-driven computer. The "facts about objects" approach to representing knowledge was viewed as the fundamental technique in obtaining this mode of operation. For example, given a knowledge base that is organized by the domain expert as facts about different categories of problems with a particular device, a user can quickly narrow in on a solution to the problem by working from very general categories of problems - "a problem with the engine," to very specific categories - "a problem with the engine when starting the car on days when the temperature is below 70°F." Further, potential causes of the fault are immediately accessible in the category - "can be caused by excessive humidity in the air." In the same manner, solutions could also be viewed - "is prevented by decreasing the amount of fuel injected into the carburetor."

The user interface was seen as a key design issue in order to obtain this functionality. Because text is the most general form of communication, it was chosen as the primary method for entering information into the shell. More importantly, a need to easily organize and relate the incoming information to existing information was needed. To facilitate this, a common method for organizing text-based information was chosen - the outline. By automatically organizing the various information entered by an expert into hierarchies of categories and displaying this information in an

abbreviated form via an outline, called an index in Enlighten, both the expert and the non-expert user can quickly and easily access large amounts of information.

6.2 Results

The primary goal of a phase two SBIR is to generate a commercial-quality product that can then be marketed during the phase three effort. Due to time limitations and the complexity of the task, UTC was unable to completely meet this goal. While the current version of the software satisfies many of the original design goals, a considerable amount of additional programming is required for its commercialization. The majority of the effort in developing the current version of the software was expended in the user interface which comprises approximately 90% of the actual code.

Functionally, the current version of Enlighten allows the user to create, view, index, and edit topic categories, and to enter facts, examples, and subcategories related to any of these categories. In addition to these features, a simple query engine exists that allows Yes/No types of questions to be answered. For example, having asserted the above example, when asked if a car requires fuel, Enlighten would respond, yes.

Where appropriate, textual entries are linked to their corresponding categories (hot text). This not only provides a means to quickly navigate through the information network, but it also aids in resolving problems in reference due to context. Thus, if a user were to double-click on the example "engine" within the category "a part of a car," and then double-click on the example "engine" within the category "a part of a jet," two different category windows would be opened. The first would be an engine in the context of a car, and the second, engine in the context of a jet.

The information entered into Enlighten is internally organized as described in Section 3. For example, the addition of the fact that "a car requires fuel for power" is not only added to the category car, but also causes the categories "things that requires fuel" and "things that requires fuel for power" to be generated. These additional categories include the reference to car and are directly available to the user to add additional facts about these general categories or view what additional objects within the knowledge base meet these criterion.

The current version of Enlighten also includes the basic commands necessary for opening, closing, and saving a knowledge base. While this allows the user to create multiple knowledge bases for different domains, only one knowledge base may be loaded at a given time.

6.3 Limitations

The primary functional limitations of the current version of Enlighten are in the area of its inferencing capabilities. The original design goal was to include a robust set of inferencing capabilities to include hierarchical, temporal, structural, causal, spatial, and conditional relationships. Due to time limitations however, only hierarchical inferencing is complete. Structural inferencing is partially complete, allowing a user to view simple structural hierarchies. The deeper form of structural inferencing relating to analysis based on structure, however, is not complete.

45

It is UTC's view that while the implementation of these inferencing modes is not complete, the overall design of the system supports this type of inferencing through the use of semantic knowledge as opposed to simple, heuristic knowledge. ʼ ˌ, in fact, these inferencing capabilities that drove the design of the representational structures. The extreme amount of effort required to develop a usable user interface for entering and viewing a knowledge base precluded the development of some of these very desirable features. It was decided that the overall goals of the project would be better exemplified by having a functional user interface with partial inferencing, rather than a complete inference engine with an interface that only a knowledgeable programmer could use.

Another functional limitation is in the area of context. Of particular concern during the development of Enlighten was the ability for a user to reuse the same words in different situations or contexts. As in the previous example, the engine of a jet is not the same as the engine of a car, but both can be referred to as engines within their respective contexts. Enlighten includes the necessary mechanisms to internally represent these categories differently in the different contexts. Unfortunately, there was inadequate time to develop the associated user interface elements that would allow the user choose a category in situations where Enlighten is unable to determine the appropriate context.

The support of conjunctions in statements is also not included in the current version of Enlighten. This would provide a powerful method for not only querying Enlighten, but also asserting facts, e.g. "a car requires fuel, a battery, a driver, and tires." Once again, the implementation of these ideas was curtailed due to time constraints as opposed to design flaws. An earlier prototype did, in fact, support conjunctions.

Other functional limitations of Enlighten, including the representation of quantities and the use of adjectives and adverbs in descriptions, were ignored due to time limitations. The representational structures are currently able to accommodate these concepts. The problem in their implementation however lies in the development of a parsing strategy that could deduce the word type, e.g. adjective, adverb, or quantity, from the sentence without asking the user or referring to some predefined list of options.

While most parsers utilize a predefined list of words to make these determinations, this still does not solve the problem of what to do when an unknown word is encountered. Continual bantering of the user with inappropriate questions such as "is this word an adjective" are considered unacceptable by UTC. What is sought, is some transparent method for making these determinations or, as a minimum, some simple method for allowing the user to unambiguously inform Enlighten of the word type without relying on an understanding of English grammar.

Because development of the user interface was so time-intensive, there was only adequate time to complete a minimal interface. A complete commercial version of Enlighten will include extended editing within the category dialogs including individually resizable fields, greatly enhanced indexing capabilities that will allow a user to generate indexes along any given topic, e.g. "cars that require fuel," and an expanded query dialog and associated query engine to handle complex questions involving conjunctions, e.g. "What requires fuel AND is used by the military."

6.4 Conclusions

While the development of Enlighten did not result in a ready-for-market software package. the current implementation provides an excellent prototype of a second generation expert system shell. The key elements of Enlighten that move it ahead of traditional heuristic expert systems are its use of semantic knowledge as opposed to purely heuristic knowledge, a user-driven approach where the user is in charge of deciding where to go first with Enlighten providing assistance only when asked, and the ability to actively organize incoming information into a network of related concepts further providing a method for refining knowledge.

In retrospect, there are a number aspects of the original design that would be changed or augmented having worked with the current version of Enlighten. Perhaps the most important would be the development of a syntax that would allow users to organize their knowledge into a single text file that is developed off-line. This would be done using a similar constrained syntax with perhaps key words being used to set the current context of the incoming information. Consider the following:

TOPIC: car

NAME:
automobile

DEFINITION:
a machine that is used for...

FACT:
requires fuel to function
costs money to maintain

EXAMPLES:
Mustang Probe

In this example, the key words TOPIC:, NAME:, FACT:, and EXAMPLES: could be used to organize the information textually in a manner that is similar to that done graphically by the user interface. This is viewed as an engineered solution that avoids the difficulties associated with true natural language parsing. Further, this method of organization actually requires less text than its equivalent form in prose as shown below.

A car, or automobile, is a machine that is used for.... It requires fuel to function and costs money to maintain. The Mustang and Probe are both examples of cars.

Bibliography

Aiello, Luigia, and Levi, Giorgio, The Uses of Metaknowledge in AI Systems, Elsevier Science Publishers B.V., 1988, pp. 243-255.

Barr, Avron, Paul R. Cohen, and Edward A. Feigenbaum, The Handbook of Artificial Intelligence, Volume IV, Addison-Wesley Publishing Co., Inc., Reading, MA, 1989.

Blaxton, T.A. and C.R. Westphal, "Combining explicit queries with simulation techniques during knowledge acquisition," 1988.

Brachman, Ronald J., What IS-A and Isn't: An Analysis of Taxonomic Links in Semantic Networks, Fairchild Laboratory for Artificial Intelligence Research, Computer Magazine, October 1983, pp. 30-36.

Carnap, Rudolf, Meaning and Necessity: A Study in Semantics and Modal Logic, The University of Chicago Press, 1956.

Copi, Irving M., Symbolic Logic, Fifth Edition, Macmillan Publishing Co., Inc.,1979.

Dement, Charles W., et. al., PACIS: A Platform for the Automated Construction of Intelligent Systems, Ontek Corporation, 1988.

Facione, Peter A., and Schere, Donald, Logic and Logical Thinking A Modular Approach, Woodbridge, Connecticut, Ox Bow Press, 1984.

Finin, Tim, and Silverman, David, Interactive Classification as a Knowledge Acquisition Tool, Expert Database Systems, 1986, pp. 79-90.

Henghold, W. M. , Insley, R. S., Evans, P. A., and Park, J., Low Cost Unified Expert System Tool for Manufacturing, AFWAL-TR-88-4041, 1988.

Israel, David J., and Brachman, Ronald J., Distinctions and Confusions: A Catalogue Raisonne, unpublished manuscript.

Lenat, Doug, et. al., CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks, The AI Magazine, Winter 1986, pp. 65-92.

Marcus, Sandra, Automating Knowledge Acquisition for Expert Systems, Kluwer Academic Publishers, Boston, 1988.

McArthur, Tom, Lexicon of Contemporary English, Longman Group UK Limited, 1981.

Musen, M.A., Fagan, L.M., et. al., Knowledge Acquisition Tools for Expert Systems, Volume 2, Academic Press, 1988.

Parsaye, K. and Murphree, S., "Automating The Knowledge Acquisition Process", Intelligence Ware Technical Report, March 1987.

Patel-Schneider, Peter F., et. al., <u>Term Subsumption Languages in Knowledge Representation</u>, AI Magazine, Summer 1990, pp. 16-23.

Ralescu, Anca L., <u>A Case for the Use of Conceptual Graphs for Knowledge Representation In a Knowledge Based System</u>.

Regoczei, Stephen, and Hirst, Graeme, <u>Knowledge Acquisition as Knowledge Explication by Conceptual Analysis</u>, Technical Report CSRI-205, January 1988.

Regoczei, Stephen, and Hirst, Graeme, <u>On 'Extracting Knowledge from Text': Modelling the Architecture of Language Users</u>, Technical Report CSRI-225, January 1989.

Regoczei, Stephen, and Plantinga, Edwin P.O., <u>Creating the Domain of Discourse: Ontology and Inventory</u>, Knowledge-Based Systems, Vol. 2, pp. 293-309.

Selfridge, Mallory, <u>Cognitive Expert Systems and Machine Learning: Artificial Intelligence Research at the University of Connecticut</u>, The AI Magazine, Spring 1987, pp. 75-79.

Sowa, J. F., <u>Conceptual Structures Information Processing in Mind and Machine</u>, The Systems Programming Series, Addison-Wesley Publishing Company, 1984.

Whitehead, Alfred N., and Russell, Bertrand, <u>Principia Mathematica</u>, Volume 1, Second Edition, Cambridge The University Press, 1957.

Wilensky, Robert, <u>Some Problems and Proposals for Knowledge Representation</u>, University of California, Berkley.

Table of Contents

List of Figures

1.0 Getting Started

This tutorial provides a general overview on how to use Enlighten. It explains how categories are created, defined, and edited. It describes how indexes are created to provide an overview of a category's facts, examples, and subcategories. It also explains how to quickly access a particular category using 'hot text'. Just as importantly, the tutorial addresses Enlighten's user interface limitations. Understanding these limitations is essential to prevent user frustration and enable productive knowledge base development.

1.1 Loading 'Core' Information

After the application has finished loading, a menu bar like the one shown in Figure 1 will be displayed. Select **Open** in the **File** menu, and select the file named "Core" in the Open dialog box. This action loads the core knowledge that Enlighten uses to interpret and organize new information entered by a user.

```
  File   Edit   Category   Index
```
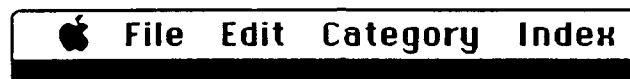
Figure 1. Enlighten's Menu

Once the Core is loaded, the application's main index window will appear (Figure 2). Notice that the window contains four root categories: *action,*

```
▦□▦▦▦▦▦ Core ▦▦▦▦▦▦
action                            ⇧
event
property
thing




⇦                          ⇨ ⬚
```

Figure 2. Enlighten's Core Index

event, property, and *thing.* Recall from Section 3.4.3 that these categories are the building blocks upon which an knowledge base is organized. Each category created in the system will be related (directly or indirectly) to one of these root categories via a subcategory, example, or fact relation.

1.2 Adding Categories

Select *Add* in the *Category* menu to add a new category to the knowledge base. Add will bring up the window shown in Figure 3. It has a field for describing a new category and a set of buttons for selecting its type.

Figure 3. Adding a New Category

Notice that the types are the four root categories discussed in the previous section. Every category created in the Add Category will be asserted as an *example* of the selected type. To verify this, try adding the category, 'engine' as an example of a 'Thing':

- Select the button labeled 'Thing', if it is not already selected

- Type "engine" into the Category box

- Click on the Add button.

This creates the 'engine' category and opens the Category window shown in Figure 4. Now, double click on the Context field which contains 'thing' to view its category information. Notice the category, 'engine' appears as an *example* of a 'thing', as originally asserted.

It is important to note that a multiple word category name (e.g. "car part") cannot be used to create a new category in the Add Category window. Entering a multiple word name using hyphenation is discouraged (e.g. "car-part"). The recommended method is to create the category by entering its definition (e.g. "part of a car") and then assign the multiple word name in the category's Names field. Enlighten allows the expert to then refer to this category with the multiple word name or definition.

```
┌─────────────────────────────────────────────────────┐
│ ▤□▤▤▤▤▤▤▤▤▤▤▤▤▤ engine ▤▤▤▤▤▤▤▤▤▤▤▤▤▤ │
├─────────────────────────────────────────────────────┤
│   Parent  ┌──────────────────────────────────┐      │
│           │                                  │      │
│           └──────────────────────────────────┘      │
│  Context  ┌──────────────────────────────────┐      │
│           │ thing                            │      │
│           └──────────────────────────────────┘      │
│ Definition ┌─────────────────────────────────┐      │
│            │                                 │      │
│            │                                 │      │
│            └─────────────────────────────────┘      │
├──────────────────────────────────────────────┬──────┤
│ Names                                        │  ⇧   │
│ engine                                       │      │
│                                              │      │
│                                              │  ⇩   │
├──────────────────────────────────────────────┼──────┤
│ Facts                                        │  ⇧   │
│ is a thing                                   │      │
│                                              │      │
│                                              │  ⇩   │
├──────────────────────────────────────────────┼──────┤
│ Examples                                     │  ⇧   │
│                                              │      │
│                                              │      │
│                                              │  ⇩   │
├──────────────────────────────────────────────┼──────┤
│ Subcategories                                │  ⇧   │
│                                              │      │
│                                              │      │
│                                              │  ⇩   │
└──────────────────────────────────────────────┴──────┘
```

Figure 4.   Category Window

- Go to the Add Category (⌘-A or select Add in the Category menu)

- Enter "a part of a car" in the Category box

- Click on the Add button. This will create the category and open the Category window shown in Figure 5.

Name the category, 'a part of a car' as "car part".

- Click in the Names field

- Type "car part"

- Press return.

After the return, the window's title will change from "a part of a car" to "car part" to indicate that the category can now be referred to by name or its definition.

```
┌─────────────────────────────────────────────────┐
│ ▤▢▦▦▦▦▦▦▦▦ a part of a car ▦▦▦▦▦▦▦▦▦ │
├─────────────────────────────────────────────────┤
│   Parent │ a part                              │ │
│          └─────────────────────────────────────┘ │
│  Context │ thing                               │ │
│          └─────────────────────────────────────┘ │
│ Definition│ a part of a car                    │ │
│          │                                      │ │
│          └─────────────────────────────────────┘ │
├─────────────────────────────────────────────┬──┤
│ Names                                       │ ⇧ │
│                                             ├──┤
│                                             │  │
│                                             ├──┤
│                                             │ ⇩ │
├─────────────────────────────────────────────┼──┤
│ Facts                                       │ ⇧ │
│                                             ├──┤
│                                             │  │
│                                             │  │
│                                             ├──┤
│                                             │ ⇩ │
├─────────────────────────────────────────────┼──┤
│ Examples                                    │ ⇧ │
│                                             ├──┤
│                                             │  │
│                                             │  │
│                                             ├──┤
│                                             │ ⇩ │
├─────────────────────────────────────────────┼──┤
│ Subcategories                               │ ⇧ │
│                                             ├──┤
│                                             │  │
│                                             ├──┤
│                                             │ ⇩ │
└─────────────────────────────────────────────┴──┘
```

Figure 5. Category Window for Part of a Car

Examine the new categories that were just asserted into the system as *examples* of 'thing'. Click on the window for 'thing' or double click on the Context field for 'a part of a car'. Notice that two new categories: 'a part' and 'a car' were asserted and added to the Examples field for 'thing'. Wondering what happened to 'a part of a car'? Double click on 'a part': 'a part of a car' is a *subcategory* of 'a part'. The category, 'a part of a car' does not appear in the Examples field for 'thing' because its an *implied* example. That is, since 'a part' is an example of a 'thing' and 'a part of a car' is a subcategory of a 'a part' then 'a part of a car' is an implied example of a 'thing'. As a general rule, Enlighten does not show implied examples in the Examples field of a Category window. However, if the user selects 'View All Examples' in the Category menu, Enlighten opens a complete index of examples for the current category. This includes those examples like 'car part' that are implied through a subcategory relation.

In summary, the Add windoow allows the user to enter the *name* or *definition* of a category. Categories are described using an option article (a, an, the), a subject word, and optional prepositional (<pp>) and restrictive (<rp>) phrases which restrict

the subject as shown in Figure 6. Multiple word names cannot be interpreted and will result in a syntax error.

```
<category> forms:        examples:
<subject>                engine
<article> <subject>      an engine
<subject> <pp>           part of an engine
<subject> <rp>           an engine that is part
                         of a car
```

Figure 6. Syntactic forms for categories

1.3 Getting Around

1.3.1 Hot Text

One of the key features of Enlighten's user interface is *hot text*. In general, hot text is text that has an *active link* to additional information. Thus, by double clicking on the text, a user can immediately view the information that the text is linked to. In Enlighten, each entry in the Index and Category windows is hot text. By double clicking on one of these entries, the user can quickly open for viewing the category that the entry references.

Examples:

Enlighten provides active 2-way links between subcategories and their parent.

- Double click on the entry, 'a part' in the Parent field of 'car part'

- Double click on the entry, 'car part' in the Subcategories field of 'a part'

The 'thing' category is accessible using the hot text in the Context field of either 'a part' or 'car part' as well as through the fact 'is a thing' in the category 'a part'.

- Double click on 'thing', in the Context field of 'car part'

- Double click on 'is a thing' in the Facts field of 'a part'

The hot text in the Definition field of 'car part' provides quick access to 'things of a car' which in turn provides access to 'a car'. See Section 3.4.3 for an explanation of how these categories are linked to each other.

- Double click on 'car part', in the Definition field of 'car part'

- Double click on 'a thing of a car' in the Definition field of 'a thing of a car'

1.3.2 Accessing Categories

A user can also access a category using the Add Category. Just follow the same steps that you used to create a category above. Since the category has already been asserted, Enlighten will *find* the category in the knowledge base and display its information in a Category window, rather than creating a new one of the same name. Try accessing the category 'a part of car' using the Add Category.

- Go to the Add Category (⌘-A or select Add in the Category menu)

- Enter "car part" in the Category field

- Click on the Add button.

1.4 Entering and Viewing Category Information

All of the fields in a Category window allow the user to enter information, except for the Parent and Context fields. The Parent field displays the category's parent and the Context field displays the context in which the category was asserted, neither of which may be altered. (Context will be discussed in more detail below.)

New information entered in a Category window field is asserted in the system when the user hits the return key. Before the return key is pressed, the information is completely editable by standard Macintosh methods. After the return, the information is locked and can only be deleted. It cannot be modified.

The user can move from field to field within a Category window in two ways. A field can be selected with the mouse, or, sequentially, selected with the tab key. Try selecting the fields in the Category window for 'car part' using both methods.

A cursor will appear in each of the selected fields except the Parent, Context, and Definition fields which are read-only. When one of these fields is selected, the whole field is highlighted to indicate that it cannot be modified. It is important to note that the Definition field is read-only when a definition has already been entered for the category. Otherwise, when creating a new category enlighten allows the user to define the category (see section 1.4.2).

1.4.1 Names

The Names field allows the user to enter and view the *names* and *synonyms* for a category. Note, that a name or synonym can consist of *multiple words* (e.g. car part).

As a rule, the first name in the field should be the primary or most common name for the category. This is the name that will be used whenever the category is referred to by other categories.

The other names in the Names field are important because they provide additional ways in which a user can reference the category. Therefore, it is a good idea to include both singular and plural forms of all names and synonyms, (e.g., 'car parts'). Enter this synonym into the Names field.

1.4.2 Definition

The Definition field allows the user to enter and view the *definition* of the category. Categories are defined using an article (a, an, the), a subject word, and prepositional or restrictive phrases which restrict the subject as shown in Figure 7.

```
<definition> forms:        examples:
<subject> <pp>             a part of an engine
<subject> <rp>             an engine that is part
                           of a car
```

Figure 7. Syntactic forms for definitions

Define the category 'engine' as "a machine that converts energy into motion that is mechanical".

- If the 'engine' Category window is closed, reopen it using the Add Category.

- Click on the Definition field

- Type "a machine that converts energy into motion that is mechanical"

- Press return.

The definition sounds a little awkward because one would prefer to say "mechanical motion" rather than "motion that is mechanical", but the former does not meet the syntactic requirements of Enlighten described in Section 3.4.2 and would therefore result in a syntactic error.

As a result of the definition, 'a machine' is recognized in the Parent field as the parent of 'engine'. Recall from Section 3.4.3 that this parent relationship is formed because the set of machines is being subdivided into those which 'convert energy into motion that is mechanical' and those which do not. The former category is called 'engine'.

1.4.3 Facts

Facts must always begin with a predicate containing a form of "is", an action verb or a combination of both. An action predicate may be followed by prepostional phases or a category. In the latter case, the category can be followed by prepositional phrases which modify the predicate. Each of these acceptable syntactic forms is shown in Figure 8.

```
<fact> forms:                          examples:
is <category>                          is a machine
is <action> <pp>..<pp>                 is made of metal
                                       is used to power other machines
<action> <category> (<pp>..<pp>)       requires energy to function
```

Figure 8. Syntactic forms for facts

Tab down to the Facts field so that you can enter some facts about an engine. Try typing in the following facts, making sure to terminate each one with a return.

- requires energy to function
- is made of metal
- is a motor
- is used to power a machine

Now, take a look at how Enlighten asserts a fact about a category. Double click on 'is made of metal'. This action brings up the Category window for 'thing that is made of metal'. Notice that 'engine' is an example for this category. Enlighten uses each fact entered about a category to subdivide one of the root categories and form a new category. In this case, the fact, "is made of metal", is used to subdivide the root category 'thing', forming the category 'thing that is made of metal'. Furthermore, the original category, 'engine', is an *example* of a 'thing that is made of metal'.

These new categories which are formed automatically by asserting facts can be very useful in building an knowledge base. Facts can often be asserted about them which in turn will be inherited by all of their examples. Consider the category, 'thing that is made of metal'. There may be facts that can be asserted about this category which in turn, are *implied* facts about an 'engine' as well as *all* things that are made of metal. For example, by entering the fact "is a conductor of electricity" for 'a thing that is made of metal', it is implied that an engine "is a conductor of electricity". Verify this.

- Open the category 'thing that is made of metal' by double clicking on the fact 'is made of metal' for the category 'engine'

- Click on the Facts field, enter "is a conductor of electricity", and press return

- Activate the Category window for 'engine'

- Select View All Facts in the Category menu

The fact 'is a conductor of electricity' is not displayed in the Facts field for 'engine' since it is an implied fact. The only way to view implied facts about a category is in a facts index (see Section 4.1.2).

A-8

1.4.4 Examples

A category's examples should consist of those things that are *grouped* into sets or subcategories based upon specific properties. The properties used to uniquely define an example of a category will vary based on application requirements. An engine manufacturer, for example, would define engine examples based on *model and year* (e.g. 303-1979.), but an engine repair person, would also include *make* as a defining property (e.g. Chevy-303-1979). A layman on the other hand, might only have engine *type* as a defining property (e.g. V6). Add the following examples to the 'engine' category.

- Chevy-303-1979
- Chevy-303-1980
- Chevy-356-1980
- Judd-V8-1980
- Judd-V8-1981

Examples are themselves categories and therefore can be described by name or by definition (Refer to Figure 6). As in the Add Category window, names can only be one word. If there is a need to refer to a category using multiple words, enter the category's definition, open the example category, and add the multiple word name in the Names field.

1.4.5 Subcategories

Subcategories are defined in order to group examples together. Those examples of a category which share one or more common properties can be grouped together in a subcategory. One subcategory of 'engine' is 'an engine that is manufactured by Chevrolet'. This subcategory is formed by restricting the parent category, 'engine'. based on a particular manufacturer. Here are some subcategories that can be added to 'engine'.

- an engine that is manufactured by Chevrolet
- an engine that is manufactured by Judd
- an engine that was manufactured in 1979
- an engine that was manufactured in 1980
- an engine that was manufactured in 1981
- V6
- V8

Subcategories can be described by name or by definition just like an example or category entered in the Add Category window. The first five subcategories shown above are described by definitions because multiple word names (e.g. Judd engine and 1980 engine) cannot be entered in the Subcategories field. A subcategory must be created before it can be assigned a multiple word name.

1.5 Editing Categories

Enlighten supports the editing operations: Cut, Copy, Paste, and Clear in the Facts and Examples fields, and Clear in the Names and Subcategories fields. An operation

is performed by selecting an entry and then the desired operation from the Edit menu. Only one entry in a field can be manipulated at a time.

The Clear function allows the user to delete a name, fact, example, or subcategory from a category. For example, try adding a dummy fact for the category 'engine' and then delete it.

- Activate the Category window for 'engine' if its not already active.

- Add the fact, "is a dummy"

- Select the fact that was just added with the mouse.

- Select Clear from the Edit menu.

The Cut, Copy, Paste operations allow the user to copy and move both facts and examples from one category to another. These operations are particularly valuable when a category's examples must be moved into the appropriate subcategories. To illustrate, the example category, 'Chevy-303-1979' in 'engine' belongs in the subcategories: 'an engine that is manufactured by Chevrolet', 'an engine that was manufactured in 1979', and 'V8'. Copy and Paste can be used to perform these operations as described below.

- Open the subcategory, 'an engine that is manufactured by Chevrolet'

- Open the subcategory, "an engine that was manufactured in 1979'

- Open the subcategory, 'V8'

- Activate the 'engine' category window and select the example, 'Chevy-303-1979'

- Select Copy in the Edit menu to move the example to Enlighten's clipboard

- Activate the subcategory window for 'an engine that is manufactured by Chevrolet' and click in the Examples field

- Select Paste in the Edit menu to move 'Chevy-303-1979' from the clipboard into the Examples field

- Repeat the last two steps for the other two subcategories.

Open 'Chevy-303-1979' to view the effects of performing these operations. The category has been restricted from an 'engine' to an 'engine that is a V8 that is manufactured by Chevy in 1979'.

1.6 Saving New Information (Save As, Save)

A user can save an knowledge base to disk by selecting one of the Save operations from the File menu. The Save operation will update the currently open file

with the recent changes made to the knowledge base. *Do not use Save when the Core file is open* unless the knowledge base is to be used as core knowledge for building all other knowledge bases.

The *Save As...* operation will create a new file where the knowledge base is stored. Use Save As... to store the knowledge base that was developed in this exercise.

- Select Save As... in the File menu.

- Enter the name of the file where the engine knowledge base is to be stored.

- Click on the Save button.

2.0 Creating Indexes

Index windows can be created by the user to display separate hierarchies of facts, examples, and subcategories for any category in the knowledge base.

2.1 View All Subcategories

An index window which displays a *subcategory* hierarchy for a category can be created using the View All Subcategories function in the Category menu. Try creating the subcategory index for the category 'machine' (Figure 9).

- Open the category 'machine' (double click on 'machine' in the Parent field of the 'engine' category window).

- Select View All Subcategories in the Category menu.


```
≡□≡≡≡ a machine - SubCategories ≡≡≡≡
An engine                                          ⬆
    An engine that is manufactured by Chevy
    An engine that is manufactured by Judd
    An engine that is manufactured in 1979
    An engine that is manufactured in 1980
    An engine that is manufactured in 1981

                                                   ⬇
⬅                                              ➡ 🗗
```
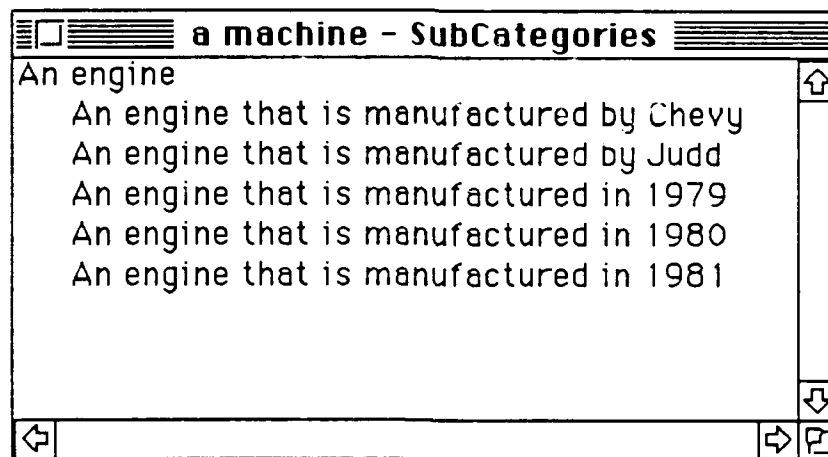
Figure 9.   New Subcategory Index

For convenience, a user can also create the subcategories index by double clicking on the *header* in the Subcategories field.

- Activate the Category window for 'machine'

- Double click on the "Subcategories" header in the Subcategories field.

A-11

2.2 View All Facts

An index window can also be created to display a comprehensive hierarchy of facts for any category. Recall from Section 4.1.2 that a fact index allows the user to view all of the facts about a category including those which are inherited from other categories. Try creating the fact index for the category 'Chevy-303-1979' shown in Figure 10.

- Open the category 'Chevy-303-1979' (double click on 'engine' in the Subcategories index window, then double click on 'Chevy-303-1979' in the Examples field of 'engine').

- Select View All Facts in the Category menu or double click on the "Facts" header in the 'Chevy-303-1979' Category window.

The index is indented to show the category level where each fact is asserted. For example, the fact 'is an engine' in not indented because it is asserted for the category 'Chevy-303-1979'. The facts, 'converts energy into motion that is mechanical', 'is made of metal', 'is a motor', 'is used to power a machine', and 'is a machine', are indented under 'is an engine' because they are asserted for the category 'engine'. Similarly, the fact, 'is a conductor of electricity' which is displayed incorrectly in the index because of a software glitch as intended under 'is made of metal'.

```
┌──────────────────── Chevy-303-1980 - Facts ────────────────────┐
│ is an engine                                                  ⇧ │
│     converts energy into motion that is mechanical              │
│         is a thing that converts energy into 'something'        │
│             is a thing that converts energy                     │
│                 is a thing that converts                        │
│                     is a thing                                  │
│     is made of metal                                            │
│         is a conductor is made of electricity                   │
│             is a conductor                                      │
│         is a thing that is made of 'something'                  │
│             is a thing that is made                             │
│     is a is motor                                               │
│     is used to power a machine                                  │
│         is a thing that is used to 'something'                  │
│             is a thing that is used                             │
│     is a machine                                              ⇩ │
│ ⇦ │                                                    │ ⇨ │ ⊡ │
└────────────────────────────────────────────────────────────────┘
```
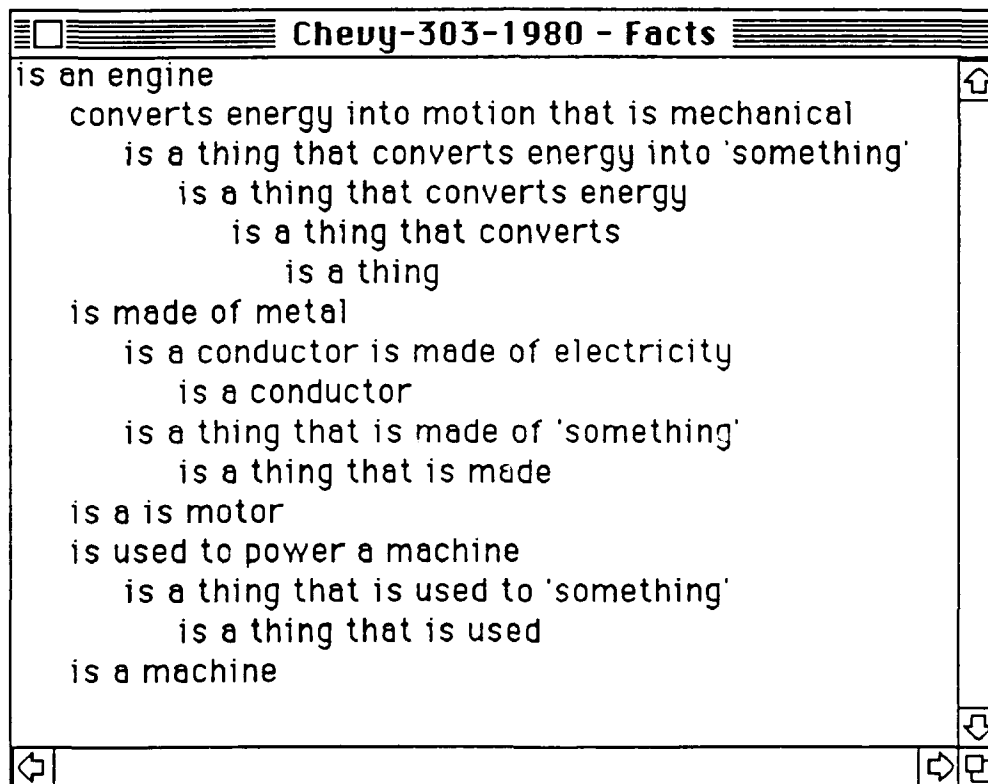
Figure 10. Chevy Facts Index

2.3 View All Examples

An index window can also be created to display a hierarchy of examples for any category. An example index allows the user to view all of the examples of a category including those which are inherited through subcategory relations. Try creating the examples index for the category 'machine' shown in Figure 11.
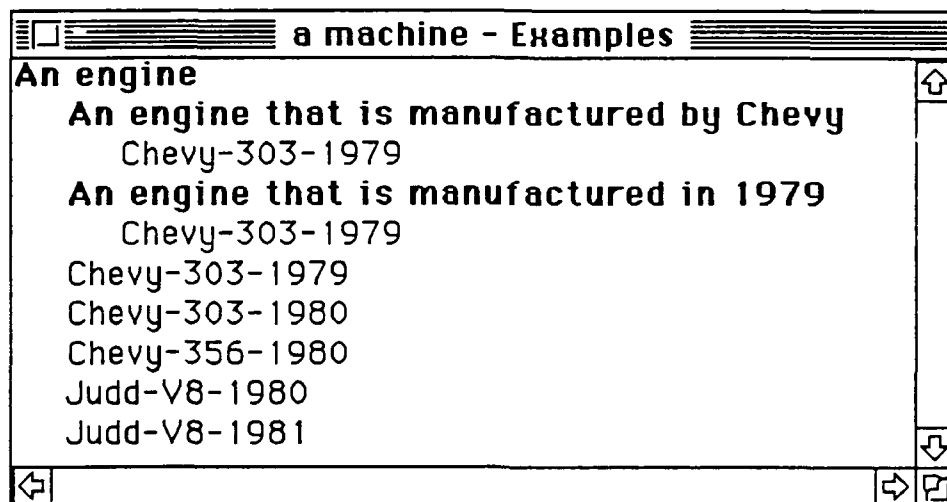
```
╔═▄▄▄▄▄▄▄▄▄▄▄ a machine - Examples ▄▄▄▄▄▄▄▄▄▄▄╗
An engine                                              ⇧
   An engine that is manufactured by Chevy
      Chevy-303-1979
   An engine that is manufactured in 1979
      Chevy-303-1979
   Chevy-303-1979
   Chevy-303-1980
   Chevy-356-1980
   Judd-V8-1980
   Judd-V8-1981                                         ⇩
╚═══════════════════════════════════════════════════╝
```

Figure 11. Index for Machine Examples

Enlighten uses indentation and contrasting text styles to show the category level where each example is asserted. Subcategories are displayed in **bold** and examples are displayed in a normal style. Therefore, 'engine' is a subcategory of 'machine' which has the five examples: 'Chevy-303-1979', 'Chevy-303-1980', 'Chevy-356-1980', 'Judd-V8-1980', and 'Judd-V8-1981'. Further, 'an engine that is manufactured by Chevrolet' is a subcategory of 'engine' which currently has one example (e.g. 'Chevy-303-1979') defined for it. Because 'Chevy-303-1979' was copied, instead of cut from the Example field of 'Engine' then pasted to the Example fields of 'an engine that is manufactured by Chevy' and 'an engine that is manufactured in 1979', it appears three times in the list.

3.0 Context

Enlighten allows a user to define multiple categories with the same name. For example, it is possible to have two categories named "Judd": Judd, the manufacturer and an engine that is manufactured by Judd.

To create multiple categories with the same name, the user must create each category within a different *context.* (see Section 3.4.1) or category. To illustrate, create the category 'manufacturer' and add 'Judd' as an example.

- Open the Add Category

- Enter "manufacturer" in the category field, select the category type, Thing, and click on Add

- Enter "Judd" in the Examples field of the 'manufacturer' Category window and hit return

- Double click on 'Judd' to verify that its context is 'manufacturer'.

The category 'Judd' is defined in the context of 'manufacturer' since it was added in the 'manufacturer' Category window.

The same rule applies when a subcategory is added to a Category window. Open the category 'an engine that is manufactured by Judd' from the category 'engine'.

- Open the category 'engine'.

- Open the category 'an engine that is manufactured by Judd'

- Enter the name "Judd" into the Names field and hit return.

Note, that the subcategory's context is set to 'engine' because it was added in the 'engine' Category window above. This subcategory is the other 'Judd' category that was described earlier. The user can name it, "Judd" and still have two distinct categories.

The context of a category created in the Add Category, is initially the root category type selected by the user. Recall that the category 'manufacturer' was created in the Add Category with the root category type 'thing' selected. Therefore, the context of 'manufacturer' is 'thing'. Verify this by opening the category, 'manufacturer'.

A category's context changes when a category is given a definition. To illustrate, define the category 'manufacturer' as "an organization that produces goods".

- Click on the Definition field for 'manufacturer'

- Enter "an organization that produces goods" and hit return.

The category 'organization' is the parent and context for 'manufacturer'. Thus, this 'manufacturer' category captures the *organizational* sense of a manufacturer and it's examples are organizations like GM and Chevrolet. If needed, the user could define another category called 'manufacturer' which is defined as "a person that produces goods".

Context is intended to help a user easily distinguish between categories with the same name when accessing information in a knowledge base. Thus, if a user asked a question about Judd, Enlighten would be able to determine which Judd category they were referring to by asking the question (Judd the manufacturer or the Judd the engine?). Unfortunately, this capability is not yet implemented in Enlighten.

4.0 Asking questions

Enlighten's Fact? dialog is intended to provide a convenient medium for asking *yes/no* type questions about a category's facts. For example, *a user can find out it if a thing that is made of metal is a conductor of electricity. Try this.*

- Select Fact? in the Category menu.

- Enter "a thing that is made of metal" into the category field of the Fact? dialog (Figure 12).

- Enter "is a conductor of electricity" into the fact field.

- Click on the Fact? button

```
▤▢▤▤▤▤▤▤▤▤▤▤▤ Fact? ▤▤▤▤▤▤▤▤▤▤▤

    Category:  ┌──────────────────────────────────┐
               │ a thing that is made of metal    │
               │                                  │
               └──────────────────────────────────┘

        Fact:  ┌──────────────────────────────────┐
               │ is a conductor of electricity    │
               │                                  │
               └──────────────────────────────────┘

      Answer:  ┌──────────┐        ┌──────────┐  ┌──────────┐
               │          │        │ Cancel   │  │ Fact?    │
               └──────────┘        └──────────┘  └──────────┘
```

Figure 12. Fact? Window

Enlighten responds with "yes" to this question because "is a conductor of electricity" has been asserted as a fact about the category 'engine'.

The current implementation of the Fact? dialog is very limited in that it can only correctly answer questions about categories that are defined in the context of 'thing'. For example, the user cannot ask a question about the category 'engine' since its context is 'machine'.

5.0 Summary

The knowledge base that has been developed through the examples presented here represents a very small chunk of what an engine repair person would need to troubleshoot car engines or that engine manufacturer would need to maintain information about different models and parts. After trying the examples presented here, a user should have the confidence and ability to expand on this knowledge base or create knowledge bases for other application areas.