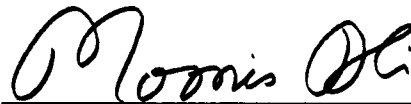


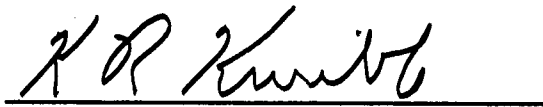
To the Graduate Council:

I am submitting herewith a thesis written by Harry Joseph Ferber II entitled "Automating the Knowledge Acquisition Process in Expert System Development." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.



Moonis Ali, Major Professor

We have read this thesis
and recommend its acceptance:



Accepted for the Council:



Vice Provost
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under the rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without written permission.

Signature Harry J. Ferber II
Date 11/28/88

AUTOMATING THE KNOWLEDGE ACQUISITION PROCESS
IN EXPERT SYSTEM DEVELOPMENT

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Harry Joseph Ferber II
December 1988

DEDICATION

This thesis is dedicated to my friends, relatives and most especially to my Mother. Because of the continuing encouragement from my friends and relatives and the never-ending love and encouragement of my Mother, the work which has created this thesis has all been worthwhile. I express my sincerest thanks to everyone. To my Mother, I say "Thanks for everything and I love you."

ACKNOWLEDGEMENTS

In the course of the development of this work, a number of people have been instrumental. In particular I would like to thank Mr. Robert S. Jones, Assistant Engineer, who tolerated many simple questions regarding electricity and electrical systems. His responses provided me with enough electrical background to understand what was necessary in the electrical domain. I would also like to thank Captain James C. Kinnard, a former airline pilot, whose expertise and articulate expression of same, made further learning about airplanes and their electrical systems much easier.

I am indebted to my thesis committee, Dr. Kenneth R. Kimble, Dr. Seung-Chul Lee and most especially Dr. Moonis Ali, my committee chairman. Without Dr. Ali's help and advice, this work would have not been possible. The comments and constructive criticism of my thesis committee are also appreciated. Whatever errors remain in this work, however, are solely my responsibility.

I am also indebted to the financial support provided in part by grants from NASA Langley Research Center under contract numbers NAG-1-513 and NGT-43-001-807 and to the UTSI administration for financial support and for permission to use the computer facilities. I would also like to thank the UTSI library staff, past and present, for their immeasurable help in procuring numerous technical works used as background for the development of this thesis.

I would also like to express my sincerest thanks to Larry Reynolds for his excellent work on some of the figures included in this thesis.

Finally, I would like to express my deepest and sincerest thanks to Mr. Patrick L. Sherley whose superhuman proofreading effort helped to greatly improve this thesis.

ABSTRACT

With the ever growing interest in the use of expert system technology for sophisticated problem solving, it becomes increasingly more important to expedite the traditional knowledge acquisition process from which these systems evolve. The traditional process—in which a knowledge engineer familiarizes himself with the domain, helps the domain expert formalize the required knowledge, and finally implements the representation of this knowledge in machine usable form—is slow, cumbersome, and time-consuming. In the research reported in this thesis, an attempt has been made to automate the knowledge acquisition process by employing a “learning by being told” methodology. The scope of acquired knowledge ranges from domain knowledge, including the information about objects and their relationships, to the procedural knowledge associated with the functionality of the mechanisms.

AKAS (Automatic Knowledge Acquisition System) is the constructed prototype for demonstrating proof of concept in which a domain expert directly interfaces with the knowledge acquisition system to ultimately construct the knowledge base for the particular application. In this prototype, the expert talks directly to the system using a natural language restricted only by the extent of the definitions in the analyzer dictionary containing a subset of concepts related to a given domain. In this case, the domain is the electrical system of the BOEING 737. AKAS uses a set of seed concepts and vocabulary to capture information. Using this information, the system extracts and differentiates the incoming knowledge into two fundamental classes - knowledge related to physical objects and their relationships to each other in the domain, and heuristic and functional knowledge about mechanisms. These two classes of knowledge are represented in the form of frames and functions, respectively, to create the knowledge base and to augment the control strategy of the expert system.

TABLE OF CONTENTS

CHAPTER	PAGE
1.0 INTRODUCTION	1
2.0 THE AKAS ARCHITECTURE	5
2.1 Fundamental Research Issues	5
2.2 System Structure Overview	6
2.2.1 The System Administration Module	8
2.2.2 The Concept Comprehension Module	8
2.2.3 The Knowledge Classification/Extraction Module	8
2.2.4 The Knowledge Incorporation Module	9
2.2.5 Explanation Facilities	9
2.3 System Implementation	10
3.0 THE SYSTEM ADMINISTRATION MODULE (SAM)	12
3.1 Module Purpose	12
3.2 Module Implementation	12
3.2.1 System Login	14
3.2.2 Set/Modify User Help Level	14

3.2.3 System Logout	17
4.0 THE CONCEPT COMPREHENSION MODULE (CCM)	19
4.1 Module Purpose	19
4.2 Module Implementation	19
4.2.1 The CCM Design Philosophy	21
4.2.2 The CCM Control Strategy	22
4.2.3 The Domain	23
4.2.4 Scope of the User-Defined System	24
4.2.4.1 Object-Related ERKS Definitions	28
4.2.4.2 Function-Related ERKS Definitions	30
4.2.5 Wordsense Definitions	35
4.2.6 Analyzer Dictionary Definitions	35
4.2.7 Examples of Functioning Sentences	37
4.2.8 CCM Vocabulary Learning	38
4.2.9 An Illustrative Example (Part One)	39
5.0 THE KNOWLEDGE CLASSIFICATION/ EXTRACTION MODULE (KCEM)	42
5.1 Module Purpose	42

5.2 Module Implementation	42
5.2.1 The Classification Strategy	44
5.2.2 The Extraction Process	44
5.2.3 The Domain Concept Dictionary	46
5.2.4 An Illustrative Example (Part Two)	49
6.0 THE KNOWLEDGE INCORPORATION MODULE (KIM)	53
6.1 Module Purpose	53
6.2 Module Implementation	53
6.2.1 The Domain Concept Dictionary	55
6.2.2 The Function Demons	58
6.2.3 Representative Frame Structures (RFS) and Frame Relationships	60
6.2.4 The LISP Function Dictionary	60
6.2.5 Explanation Tools	61
6.2.6 An Illustrative Example (Part Three)	61
7.0 FUTURE WORK AND CONCLUSIONS	65
7.1 Future Work	65
7.2 Conclusions	66

BIBLIOGRAPHY	68
VITA	72

LIST OF FIGURES

FIGURE	PAGE
1. AKAS Architecture	7
2. AKAS User Interface	11
3. The System Administration Module (detailed)	13
4. The User Table	15
5. The Expert Flavor	16
6. Example of AKAS Sample Session	18
7. The Concept Comprehension Module (detailed)	20
8. Knowledge Source Dictionary Examples	25
9. Wordsense Dictionary Examples	26
10. Analyzer Dictionary Example	27
11. Input Transformation by Concept Comprehension Module	40
12. The Knowledge Classification/Extraction Module (detailed)	43
13. Domain Concept Dictionary Entries	45
14. Input Transformation by Knowledge Classification/ Extraction Module	50

15. The Knowledge Incorporation Module (detailed)	54
16. An Example Function Demon	59
17. Input Transformation by Knowledge Incorporation Module	62

CHAPTER 1

INTRODUCTION

With the ever growing interest in the use of expert system technology for sophisticated problem-solving, it becomes increasingly more important to expedite the traditional knowledge acquisition process from which these systems evolve. The traditional process—in which a knowledge engineer familiarizes himself with the domain, helps the domain expert formalize the required knowledge, and finally implements this knowledge in machine usable form—is slow, cumbersome and time-consuming. In trying to capture the two basic forms of knowledge, facts and, most especially, heuristic and functional knowledge, there is often what Fiegenbaum [15] calls a “knowledge acquisition bottleneck.” This “bottleneck” is essentially a mismatch between the knowledge engineer and the expert with respect to the levels of understanding the problem domain.

The significance of this “bottleneck” is that it often prolongs the knowledge acquisition process which causes problems related to manpower availability, scheduling and delays in system development, possibly leading to premature termination of the project. If the expert is only a part-time one then he may not always be available when needed by the knowledge engineer. In trying to arrange interview meetings, there may be scheduling conflicts between the expert and the knowledge engineer slowing down the construction of the system. If the elapsed development time becomes protracted, management, who originally sanctioned the construction of the system, may become disenchanted with the rate of development and cancel the system.

Realizing the importance of reducing the “knowledge acquisition bottleneck”

in capturing both domain and heuristic knowledge, researchers have constantly strived to enhance this process through automation. One very popular automation approach is the use of a learning paradigm called "learning by being told." This approach has been used in such systems as TEIRESIAS by Randall Davis [11], KAS by Richard Duda [19], NANOKLAUS by Norman Haas and Gary G. Hendrix [17], and an automatic knowledge acquisition system for a flightline robot by William Clifford [9], Captain, USAF, as well as the system described in this thesis, AKAS (Automatic Knowledge Acquisition System).

Although these systems have used the same general learning approach, the objectives of each have varied. TEIRESIAS, using meta-knowledge and a restricted natural language vocabulary, attempted to capture rule-like knowledge for the medical diagnostic system, MYCIN. KAS, the knowledge-acquisition system for PROSPECTOR, was essentially the PROSPECTOR program without its domain-specific knowledge. Its objective was to allow application of the PROSPECTOR control strategy, "a consultation program developed for problem diagnosis in mineral exploration,"¹ on new problems representable in the KAS rule language. NANOKLAUS, by Haas and Hendrix, was a small prototype developed from a larger philosophy called KLAUS (Knowledge Learning and Usage Systems). The KLAUS philosophy advocated the use of a powerful, natural language system, the inclusion of seed concepts and seed vocabulary, the ability to capture linguistic constructs used in introducing new concepts, a methodology for integrating new concepts, and facilities allowing the user to undo previous information along with other powerful fundamental concepts. The objective of NANOKLAUS and KLAUS-based systems, in general, "was to collect and organize aggregations of individual facts for use in question-answering tasks."²

¹ Frederick Hayes-Roth, Donald A. Waterman, Douglas B. Lenat, eds., Building Expert Systems (Reading, MA: Addison-Wesley Publishing Company, Inc., 1983), p.291.

² Ryszard S. Michalski, Jaime G. Carbonell, Tom M. Mitchell, eds., Machine

In the system described by William Clifford, the objective was to teach a flightline robot about the refueling procedures related to an F-15 and an F-16 jet aircraft. Because the refueling manuals were in English, he first transformed the English sentences into a modified Conceptual Dependency form. This permitted him to add domain and world knowledge for later transformation into well-formed-formulas (wffs). The wffs were used in later planning steps for the robot's operation and for the comparison of operational scripts for later generalization. This is one of the few systems which has used Conceptual Dependency (CD) forms for knowledge acquisition. Although his eventual objectives are different from AKAS, the use of the CD forms reinforces the general utility of this knowledge representation and demonstrates its utility across various domains at least as an initial representation scheme.

Because the knowledge acquisition process for FLES[1-4], a diagnostic flight expert system and other diagnostic expert systems [6-8,12-14,16,21-22] developed at UTSI, were equally slow, tedious, and time-consuming, it was desirable to see if automation could also play a role. Before this possibility could be fully evaluated, however, it was necessary to analyze the domain under which the system would be developed.

Because of this author's previous experience in the creation of the BOEING 737 electrical system knowledge base for FLES, this domain was selected. Having previously determined the frames and heuristics necessary for the diagnostic system, it was now necessary to analyze the domain's language. With an understanding of the elements, relationships and syntactic constructs in the domain, it was determined that the system should be built upon a robust semantic-based natural language system to provide the most flexibility. Previous work with a system called NALI, a keyword-based system developed at UTSI [5], also showed

Learning, "Learning By Being Told: Acquiring Knowledge for Information Management", (Los Altos, CA: Morgan Kaufmann Publishers, Inc., 1983.), p. 407.

that a semantic-based system was needed. The natural language system chosen was the one created by Richard E. Cullingford and his colleagues. This system creates conceptual-dependency-like representations of the input sentences controlled by predefined definitions in an analyzer dictionary. Based on the output generated by this system, AKAS functions extract object knowledge and interrelationships, as well as domain-related heuristic and functional knowledge. Thus, the objective of the research creating AKAS was to develop a system to attempt to capture, through natural language interaction with the domain expert, knowledge about domain objects (e.g., generators, constant speed drives, busses, etc.) to be expressed in the form of frames and slot values, and specific problem-solving strategies (heuristics), to be expressed in the form of LISP functions (because FLES is a frame-based system which employs specific functions to perform specific domain tasks, such as fault detection, fault/sensor verification and fault diagnosis).

AKAS, consisting of four fundamental modules (discussed in Chapter 2), uses the conceptual-dependency based natural language system created by Cullingford to represent the information to be processed by the other AKAS modules. From the input information the system identifies, extracts and groups knowledge into two fundamental categories - facts and domain-dependent heuristics and functional knowledge. From this extracted knowledge, frames, representing the facts and interrelationships of facts, and functions, representing the domain-dependent heuristic and functional knowledge, are created to form the knowledge base and to augment the control strategy for the particular expert system.

The following chapter, Chapter 2, overviews the architecture of AKAS and Chapters 3 through 6 discuss each fundamental module in more detail. Chapter 7 will discuss further research and present conclusions.

CHAPTER 2

THE AKAS ARCHITECTURE

2.1 Fundamental Research Issues

In developing the architecture of AKAS, research problems, mentioned primarily in the design of KLAUS (Knowledge Learning and Using Systems), were seriously considered. The consideration of these problems explain some elements in the overall system design and the presence of some system features.

The KLAUS philosophy advocates the use of interactive, mixed-initiative dialogues in a system's design. The following list summarizes other general research problems for a KLAUS design. As stated above, some elements of this philosophy influenced the design of AKAS.

The research problems are summarized as follows: ³

- 1) A powerful natural language processing capability is required.
- 2) A structure for lexical entries must be specified so that the system can acquire new lexical information.
- 3) Seed concepts and seed vocabulary must be identified for inclusion in a core system.
- 4) A methodology is needed for integrating new concepts into the system's knowledge base.
- 5) Means are required for detecting and dealing with inconsistent data.
- 6) Linguistic constructs used in introducing new concepts must be identified and analyzed to be interpreted correctly by the natural language system.

³ Ryszard S. Michalski, Machine Learning " Learning By Being Told: Acquiring Knowledge for Information Management " pp. 406-407.

- 7) An efficient problem-solving capability is needed to answer questions and draw inferences for integrating newly-acquired information.
- 8) A set of readily understandable questions is needed for eliciting information from the user.
- 9) Facilities must be provided for allowing a user to change his mind about what he has told the system.

In the design of AKAS, points one through five have been incorporated into the design. The other points not fully developed offer opportunities for further research to be discussed in the future work and conclusion chapter, Chapter 7. The following sections, overviewing the system architecture and system implementation, describe how the above mentioned points have been incorporated into the overall system design.

2.2 System Structure Overview

As illustrated in Figure 1, AKAS consists of four basic modules. They are 1) the System Administration Module (SAM), 2) the Concept Comprehension Module (CCM), 3) the Knowledge Classification/Extraction Module (KCEM) and 4) the Knowledge Incorporation Module (KIM). The boxes indicated in Figure 1 as "Target Knowledge Base" and "Expert System Control Strategy" are considered to be the repository for the information created by the system. The box labeled "Explanation Capabilities" is simply there to indicate that AKAS has the ability to provide a simple explanation indicating what input was responsible for creating what output.

Chapters 3 through 6, respectively, will discuss each of the four basic modules, the SAM, the CCM, the KCEM and the KIM in more detail. However, for purposes of overviewing the basic architecture, the following sections briefly describe each module's purpose.

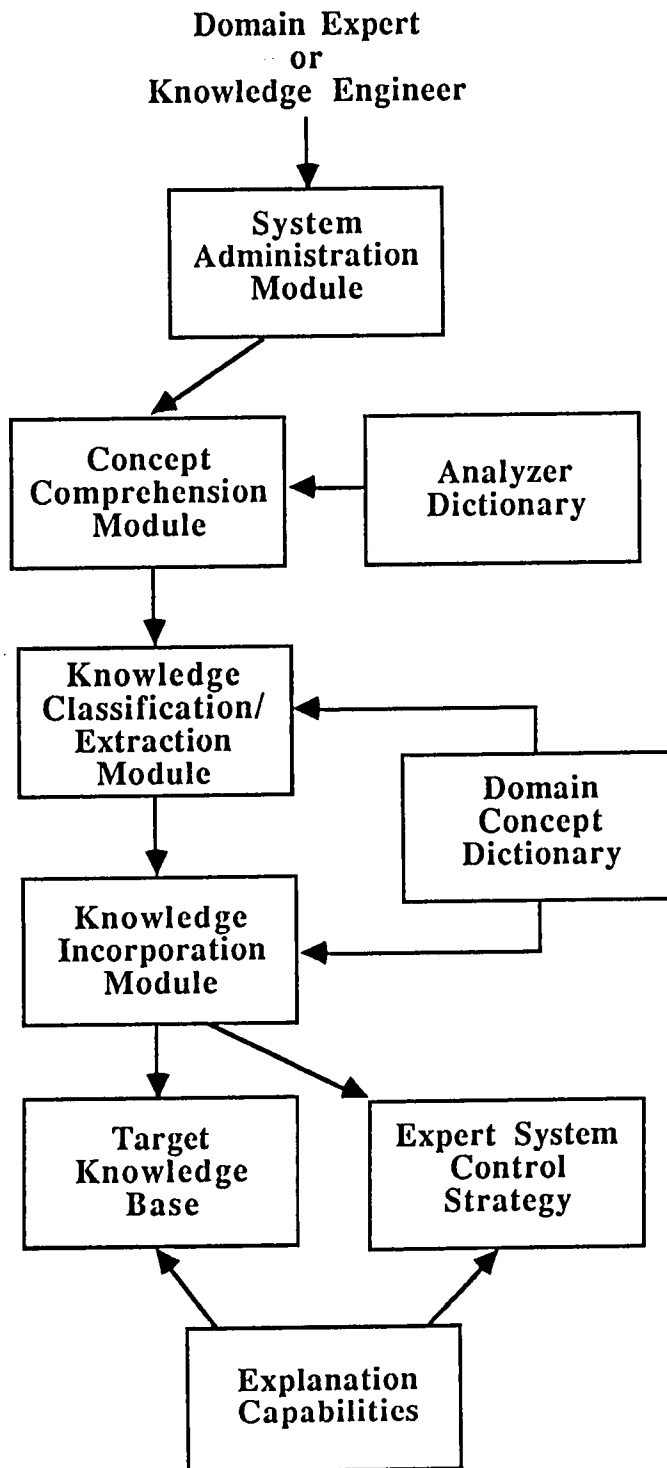


Figure 1. AKAS Architecture

2.2.1 The System Administration Module

The System Administration Module is designed to perform bookkeeping tasks for the system. The tasks provide the system a means of identifying the expert, allow the user to adjust the amount of help needed in the operation of the program, correlate the input with created structures and functions, relate the created structures and functions to the expert source, and save the information for later retrieval. Permitting the user to adjust help levels presently allows the user to control the amount of help he receives from the system. The size of the currently simple help message is decreased in relation to the user's help level. With further refinements in developing the help facility, this division of help requirements into three levels would permit users of varying experience levels to operate the system with greater ease.

2.2.2 The Concept Comprehension Module

The Concept Comprehension Module provides the means for the user to interact with the learning program. It is designed to support interactive mixed-initiative dialogues, comprehend currently understood concepts and learn both new concepts and the linguistic constructions used to express them. The fundamental approach used to comprehend the language is based on conceptual dependency theory. The system employed here was adapted from the NLP toolkit analyzer developed by Richard E. Cullingford and his colleagues. This module provides the powerful natural language facility mentioned as the first research issue. This module also provides the structure for lexical entries, and the seed concepts and seed vocabulary mentioned above as research issue points two and three, respectively.

2.2.3 The Knowledge Classification/Extraction Module

The major purpose of this module is to differentiate between the two fundamental forms of input, that dealing with physical objects in the domain and their interrelationships and the input dealing with fundamental domain heuristics,

domain-related concepts and functional knowledge. The classification is performed by predetermining which words are most likely to describe the classes of information. For example, "generator", "constant speed drive", and "bus" are physical domain objects and "connect" or "power" are verbs which may indicate relationships between two objects. For domain heuristics and functional knowledge, such words as "determine", "diagnose", "detect" or "verify" might be used to describe how some operation is performed. An example of a domain-related concept might be some specific concept or object, such as a "supporting-sensor" or a "bulletin-board".

2.2.4 The Knowledge Incorporation Module

The purpose of the Knowledge Incorporation Module is to transform the output of the Knowledge Classification/Extraction Module into its proper form - either objects (in this case, frames) with the appropriate relationships or functions related to functional knowledge. Producing the final output forms to be used in either the target knowledge base or the target control strategy, this module deals with the knowledge integration issue and provides a simple means for detecting and dealing with conflicting or inconsistent data. Using relationships expressed in the primitives and information in the domain concept dictionary, the system understands where new information is to be integrated. In the creation or modification of frame objects, if conflicting information is detected, the user is asked to decide if information should or should not be changed within a given frame slot.

2.2.5 Explanation Facilities

Inspired by the utility of explanation facilities in the traditional expert system architecture, it is the purpose of these explanation facilities to allow the user to understand the source material which created either a particular object or a particular function. These facilities currently provide only a simple explanation in the form of what sentence or sentences are responsible for the item in question.

2.3 System Implementation

AKAS has been implemented on a Symbolics 3670 LISP machine using the `defsystem` and `defpackage` facilities available under Genera 7.1. The current user interface was designed using the `Frame-Up` facility, a utility also available under Genera 7.1.

The system is divided into two fundamental packages, AKAS and NLP. The AKAS package contains the functions created for the implementation of the System Administration Module (SAM), the Knowledge Classification/Extraction Module (KCEM), and the Knowledge Incorporation Module (KIM). The NLP package contains the control strategy for the implementation of the NLP Toolkit designed by Richard E. Cullingford and his colleagues. It also contains the core vocabulary created by this thesis' author for demonstrating the overall AKAS system.

Figure 2 shows the layout of the user interface as designed by the `Frame-Up` facility. The first pane is, of course, the system title pane. The second pane is the command-menu pane showing the valid commands for the system. The third pane is the knowledge acquisition screen which is further subdivided into four smaller scrollable windows to be explained below. The final pane is the interactor pane which permits interaction between the system and the user.

As indicated above the knowledge acquisition screen is subdivided into four smaller scrollable windows. The top left window, the "Learning Window", is the window which carries the conversation between the user and the natural language system. The top right window, the "Intermediate Representation Window", displays the result of the natural language system. This display is useful for both system debugging and for demonstration purposes. The bottom left window, the "Dialogue Window", is used as a focal point for the user to carry on conversations with the system. It is in this window where explanations, for example, are placed. Finally the bottom right window, the "Electrical Model Window", is used to display a model of the electrical system as it grows in complexity. The model consists of a root value and all subsequent children frame objects that the AKAS currently knows.

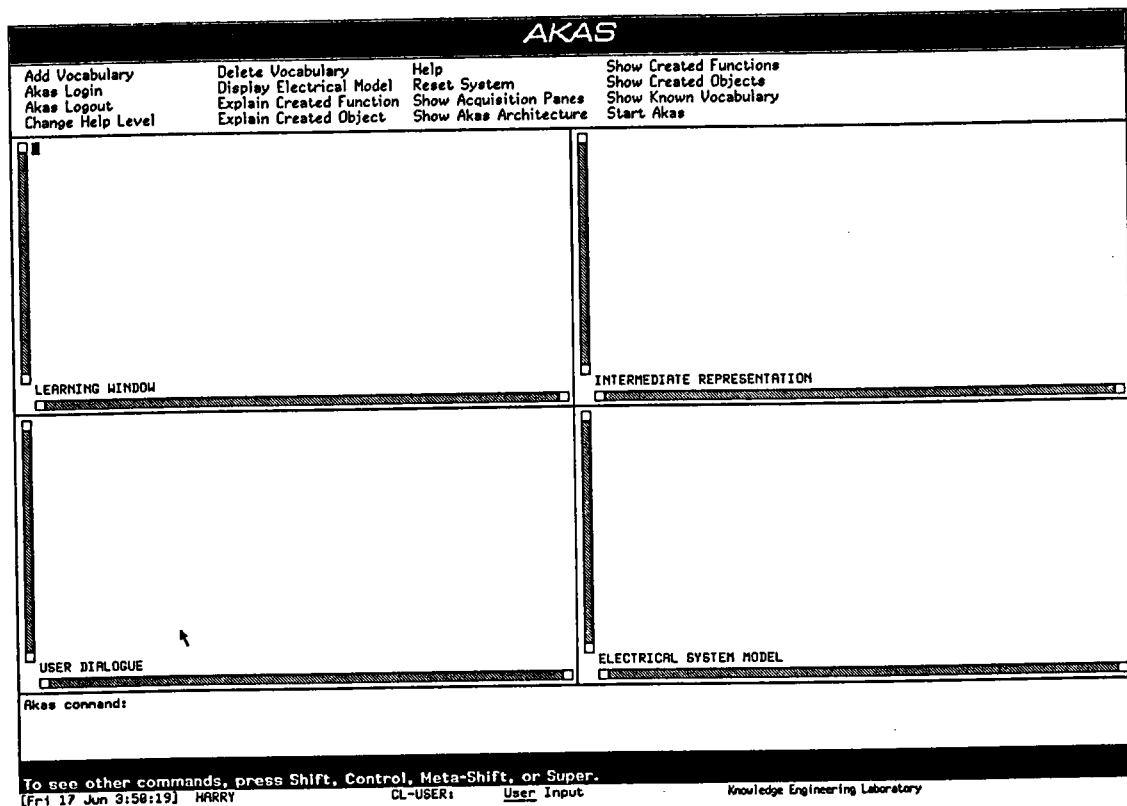


Figure 2. AKAS User Interface

CHAPTER 3

THE SYSTEM ADMINISTRATION MODULE (SAM)

3.1 Module Purpose

The purpose of the System Administration Module is to perform the necessary bookkeeping operations for the AKAS system. It initially requests the name of the expert, determines the user's experience level from a user-table, sets a help level accordingly, and allows the user to adjust this level of help, if desired. It also correlates the created structures and functions with the expert who initiated their creation, and saves this correlated information, the created frames and functions, to a file for later use.

3.2 Module Implementation

Figure 3 shows the architecture of the System Administration Module in more detail. The boxes surrounding the "System Administrator" illustrate the processes which are considered to be important to the central bookkeeping task performed by this module. As mentioned above, the processes, such as "System Login", "Set/ Modify User Help Level", "Save/Load Created Functions/Objects, and "System Logout", perform functions which are central to administering the activity of the AKAS system. The box labeled "System Administrator" is not an actual function or active process; it is used for illustrative purposes, showing that the other processes are considered to be important administrative functions.

All processes mentioned above, system login, adjustment of help level, and system logout, et cetera are implemented using standard LISP functions. The

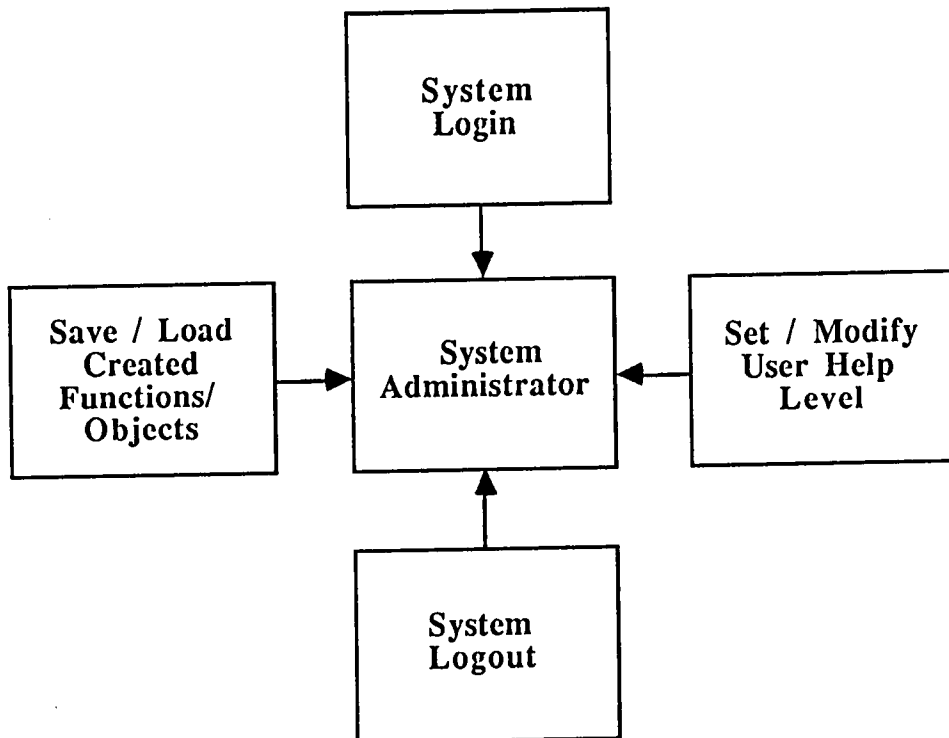


Figure 3. The System Administration Module (detailed)

following briefly details the duties of each of the above mentioned processes:

3.2.1 System Login

The system login process captures the user's name, reads a user table to determine the user's help requirements and proceeds with a brief message about the system. Figure 4 shows a sample of the user-table which stores entries representing previous users of the system. It is from this table the user help level is retrieved if the user has previous system experience. Otherwise, a new table entry is created with a default help level of novice.

It is necessary to have this user's name later for the logout process, because this process will use the user's name for creating an instance of the expert flavor and for generating a filename to store the created information. Figure 5 illustrates the structure of the expert flavor which will be discussed further in section 3.2.3.

3.2.2 Set/Modify User Help Level

The "Change Help Level" function allows the user to adjust his level of help dynamically throughout the execution of the program. There are three help levels designed to provide decreasing amounts of help :

- 1) Novice user - all help messages are provided,
- 2) Experienced user - fewer messages are provided and
- 3) Expert user - minimum amount of help is provided.

At any time the user may adjust his help level to better suit his needs.

The help facility is currently rather primitive in that it only prints a message of decreasing complexity as user experience with the system increases; that is, the message becomes shorter as the user's experience level increases. This use of levels could be expanded to create a more complex, yet manageable, help facility, because the system would not inundate an experienced user with as much information as it would a novice user or provide as much information for the expert user as for the experienced one.

```
(setf *user-table* '(  
    (JOHN (HELP-LEVEL NOVICE))  
    (EDDIE (HELP-LEVEL EXPERIENCED))  
    (FRED (HELP-LEVEL EXPERT))  
    (GEORGE (HELP-LEVEL EXPERIENCED))  
    (SAM (HELP-LEVEL NOVICE))  
    (HARRY (HELP-LEVEL EXPERT))  
))
```

Figure 4. The User Table

```

(defflavor expert ( (experts-name nil)
                    (login-time nil)
                    (logout-time nil)
                    (session-duration nil)
                    (system nil)
                    (help-level nil)
                    (created-function-instances nil)
                    (created-object-instances nil)
                    )
  ()
  :readable-instance-variables
  :writable-instance-variables
  :initable-instance-variables)

```

Figure 5. The Expert Flavor

3.2.3 System Logout

It is the system logout process which incorporates two of the processes illustrated in Figure 3. Not only does it handle the logout process, but it also handles the "Save Created Functions/Objects" process as well. The process of saving the information in machine readable form also enables the loading of this same information.

As mentioned above, the logout process uses the expert's name to create an instance of an expert flavor which will be used to capture the information created during the AKAS session. The information captured will provide values for the created instance of the expert flavor (see Figure 5). These values include the user's name, login time of the session, logout time of the session, a calculated session duration, a value for the type of system described (e.g., electrical or hydraulic), the user's help-level, and the symbolic representations of the physical objects and functions created as a result of the expert's input. The instance of the completed expert flavor and the results of the AKAS session are saved to a filename generated by using the expert's name. Figure 6 illustrates the contents of the expert's file from a sample session.

The saving of information is, of course, a practical consideration. If it is not saved then all work is lost. Capturing and associating information with a given expert provides the capability to detect and resolve information conflicts between experts. This is not implemented currently in AKAS but it was the intention of creating the expert instance to provide that capability. Because login-time, logout-time and a session-duration is calculated, it would be possible to use this information to evaluate the interaction between the system and the user based on the session duration and the amount of information captured. This might aid in improving the overall system.

```

(setf HARRY (make-instance 'expert
    :login-time '2792686975
    :logout-time '2792687733
    :session-duration '758
    :system 'ELECTRICAL
    :help-level 'experienced
    :created-function-instances '(verify)
    :created-object-instances'
    (ENGINE (ENGINE0 ENGINE1))
    (GENERATOR (GENERATOR0 GENERATOR1))
    (CSD (CSD0 CSD1))
    ))

(defun verify (current)
  (cond ((not (null (cdr supporting-sensor))) t )))

(form CSD0
isa component
with
(parents = ENGINE0)
(children = GENERATOR0)
(sensors = nil))

(form CSD1
isa component
with
(parents = ENGINE1)
(children = GENERATOR1)
(sensors = nil))

(form GENERATOR0
isa component
with
(parents = CSD0)
(children = nil)
(sensors = nil))

(form GENERATOR1
isa component
with
(parents = CSD1)
(children = nil)
(sensors = nil))

(form ENGINE0
isa component
with
(parents = nil)
(children = CSD0)
(sensors = nil))

(form ENGINE1
isa component
with
(parents = nil)
(children = CSD1)
(sensors = nil))

```

Figure 6. Example of AKAS Sample Session

CHAPTER 4

THE CONCEPT COMPREHENSION MODULE (CCM)

4.1 Module Purpose

It is the role of the Concept Comprehension Module (CCM) to act as the natural language interface between the user/expert and the system. After the initial login process is completed, this module, henceforth, becomes the main entry point for system/user interaction.

The system contains a user-defined core vocabulary and has a limited ability to capture new vocabulary using learning by simple analogy. The overall control strategy is driven by the contents of these semantic-based analyzer dictionary definitions. (The control strategy will be discussed in more detail in the next section.) As the system architecture indicates (Figure 1, page 7), this module also provides the output form which will be interpreted by the Knowledge Classification/Extraction Module to be discussed in the next chapter.

4.2 Module Implementation

Figure 7 illustrates the architecture of this module in more detail. The figure shows boxes representing the "Knowledge Source Dictionary", the "Wordsense Dictionary", and the "Analyzer Dictionary" all being used by the "Analyzer". The "Knowledge Source Dictionary" represents the repository for the fundamental structure of a concept, that is, the surface semantic categories. The "Wordsense Dictionary" represents the repository for words which might have more than one usage, as in "Take a trip" or "Take a pill". The "Analyzer Dictionary" represents

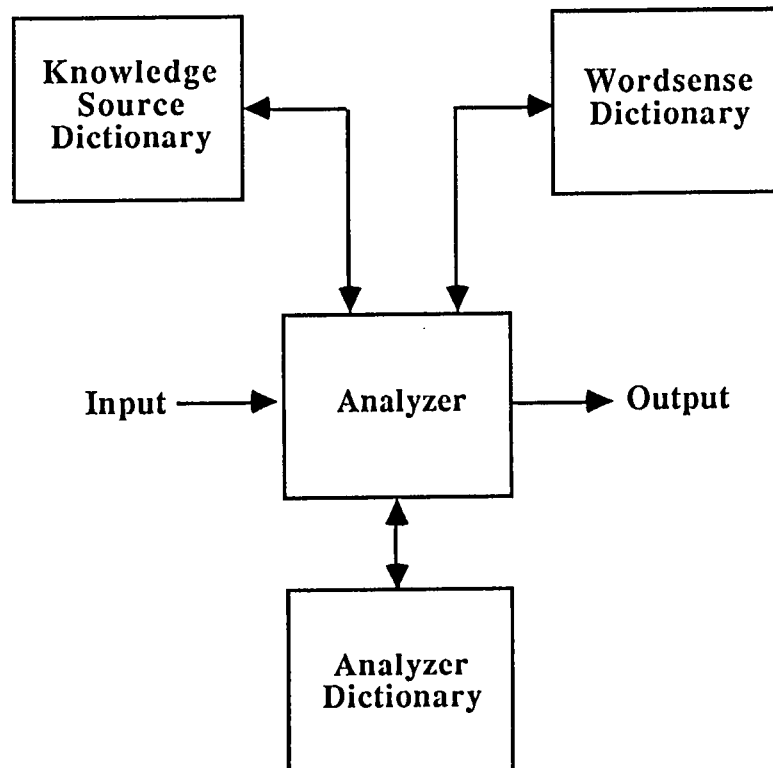


Figure 7. The Concept Comprehension Module (detailed)

the definitions which are directly manipulated by the module's control strategy to accomplish understanding.

The implementation for this module's control strategy has been directly adapted from the analyzer constructed by Richard E. Cullingford [10] and his colleagues. The vocabulary contained in the Analyzer Dictionary uses definitions designed by both Cullingford and his colleagues and by this author. The following sections discuss in more detail the implementation of the Concept Comprehension Module.

4.2.1 The CCM Design Philosophy

The basic design philosophy of the Concept Comprehension Module, directly adapted from Cullingford's implementation, is centered around the idea of "conceptual analysis or the mapping of natural language sentences into a semantic (meaning) representation. Thus, the program is technically a surface-semantic conceptual analyzer whose main job is to convert a surface form, an object encoding an English sentence or sentence fragment, into a meaning representation at the level of surface semantic categories or ERKS. "5 The ERKS (Eclectic Representations for Knowledge Structures), "an amalgam of ideas from Conceptual Dependency, Commonsense Algorithms, and Preference Semantics "6 , provide the method to represent concepts to be found in the user-defined system. In conjunction with the ERKS, two other fundamental tools are used; these are the use of wordsense definitions (different uses of the same word, such as take - in take a pill (ingest) or take a train to Bermuda (ptrans)), and the analyzer dictionary definitions. The analyzer dictionary definitions provide the overall driving force in the control strategy.

⁵ Richard E. Cullingford, Natural Language Processing: A Knowledge-Engineering Approach. (New Jersey, 1986), p. 84.

⁶ Ibid, p. 22.

4.2.2 The CCM Control Strategy

To accomplish the conversion process, from a surface form into a meaning representation, the control functions of the program focus on the lexicon as the locus of processing. That is, the analyzer dictionary definitions directly control the flow of processing. The overall design of the analyzer definitions takes its form from a production system because the definitions are in the form of test-action pairs. Thus, in this approach, there is no explicit grammar as in an ATN. This lack of a specific grammar allows processing of sentence fragments, ungrammatical inputs, as well as well-formed sentences.

The focus on the lexicon as the locus of processing control comes from the idea that "language understanding is an intensely *predictive* process. As a new sentence in a conversation starts people seem to operate in a bottom-up manner, piecing together fragments of meaning from the first few words until their conceptual *expectations* can come into play to drive the understanding process in a more top-down manner. These expectations come from the words themselves, or from the conversational context of preceding exchanges. "⁷ Thus the analyzer uses the idea of expectations associated with words to accomplish the understanding process.

"Expectations are associated with an analyzer dictionary definition in a special type of production rule called a request. Associated with a meaning structure built by a word is a set of expectations embodied in the requests indicating how the structure's characteristic roles should be filled. The special feature of these production-rule-like requests is that, in addition to modifying working memory, which contains the facts describing the current state of the understanding process, they can also add or delete requests. Requests, themselves, are activated when the associated word definition is loaded; that is, they are placed in a short-term memory (the C-LIST) of requests to be considered.

⁷ Ibid, p. 85.

In the algorithm used in this control strategy, the request consideration process repeatedly selects a request and evaluates its test part. If the test returns "non-nil", the request is said to have "fired", its action part is evaluated, and the request is removed from request memory. Requests are capable of checking semantic, lexical or contextual features of the run-time environment, and can create or interconnect conceptual structures. ⁸ This strategy continues until all requests have either fired, deleted themselves or are deleted by another request. ⁹

In summary, using these tools, namely, the analyzer dictionary definitions, the ERKS representation scheme, and wordsense definitions, this system allows the user to create Conceptual-Dependency-like representations of input sentences. Because the design of the analyzer dictionary definitions is in the form of the test-action pairs, the processing is driven until all subconcepts associated with the words in the sentence are either combined into one overall concept (the ideal case) or until all requests are exhausted.

4.2.3 The Domain

The domain chosen for this thesis was an airplane's electrical system. Based on previous experience through discussion with pilots, it was possible to identify the concepts related to both the domain objects and the function-related (heuristic) concepts.

Objects created in the CCM are related to the actual objects found in a typical electrical system - generators, generator busses, lighting, control circuits, etc. The verbs found in the object-related analyzer dictionary, also derived from the pilot conversations, describe the typical relationships between these objects. Examples of these verbs or core concepts are "connect", "power", "use", "convert", et cetera.

Concepts and verbs for the function-related heuristic information were also

⁸ Ibid, pp. 88-89.

⁹ Ibid, p. 93.

```
(DEF-ERKS-TYPE ELECTRICAL-COMPONENT
  SUPERSET (POBJECT)
  INTRINSIC-SLOTS
  ((COMP-NAME SYMBOL-P)
   (PARTOF COULDBE-ELECTRICAL-SYSTEM-P)))
```

```
(DEF-ERKS-TYPE POWER
  SUPERSET (KB-ACTIONS)
  INTRINSIC-SLOTS
  ((POWER-SOURCE COULDBE-ELECTRICAL-COMPONENT-P)
   (POWERED-OBJECT COULDBE-ELECTRICAL-COMPONENT-P)))
```

```
(DEF-ERKS-TYPE DETERMINE
  SUPERSET (CS-ACTIONS)
  INTRINSIC-SLOTS
  ((PURPOSE COULDBE-CS-ACTIONS-P)
   (METHOD COULDBE-CS-ACTIONS-P)))
```

Figure 8. Knowledge Source Dictionary Examples

```

(DEF-WORDSENSE WSVVERIFY1
  SURFACE-FORM (VERIFY)
  WS-STRUCTURE
  (VERIFY-F OBJECT-STATE (NIL) )
)

(DEF-WORDSENSE WSVVERIFY2
  SURFACE-FORM (VERIFIED)
  WS-STRUCTURE
  (CS-FAULT-STATE-F CSF-STATE-VALUE (VERIFIED)
    OBJECT-NAME (NIL) )
)

(DEF-WORDSENSE WSENG1
  SURFACE-FORM (ENGINE)
  WS-STRUCTURE
  (ELECTRICAL-COMPONENT-F COMP-NAME (ENGINE)
    PARTOF (ELECTRICAL-SYSTEM-F) )
)

(DEF-WORDSENSE WSCSD1
  SURFACE-FORM (CONSTANT SPEED DRIVE)
  WS-STRUCTURE
  (ELECTRICAL-COMPONENT-F COMP-NAME (CSD)
    PARTOF (ELECTRICAL-SYSTEM-F) )
)

(DEF-WORDSENSE WSPOWER1
  SURFACE-FORM (POWER)
  WS-STRUCTURE
  (POWER-F POWER-SOURCE (ELECTRICAL-COMPONENT-F)
    POWERED-OBJECT (ELECTRICAL-COMPONENT-F) )
)

```

Figure 9. Wordsense Dictionary Examples

```

(ADICTDEF VERIFY
  POS V
  TOPREQ
  ( ( TEST ( := (ADD-WORD-CON WSVERIFY1 WSVERIFY2) ) )
    (ACTIONS
      ;; ACCESS THE VERB CONCEPT FOR VERIFY
      (:= VERBCON (GRF '(V1) TOPCON) )

      ;; ACCESS THE CONCEPT RELATED TO
      ;; VERIFYING THE STATE OF A SENSOR
      (:= STATECON (GRF '(V2) TOPCON) )

      (ACTIVATE
        ;; CHOOSING THE USE OF THE VERB CONCEPT
        (VELREQ
          (TEST (:= PRECTOCON (IF-CLAVAIL TOPCON (G)
                                (AND (PREC G VERBCON)
                                     (PREPCONST G '(TO) ) ) )))

          (ACTIONS
            (MAKE-UNAVAIL PRECTOCON)
            (MAKE-UNAVAIL STATECON)
            (MAKE-UNAVAIL TOPCON)

            ;; PICK UP THE ITEM TO BE VERIFIED USING THE VERB CONCEPT
            ;; - THE VERB CONCEPT HAS A VERIFIED STATE
            (ACTIVATE
              (GAPREQ
                (TEST (:= OBJCON
                          (IF-CLAVAIL TOPCON (G)
                            (AND (IN-OBJ-SPOT G TOPCON)
                                 (SEMFEATURE G 'COULDBE-CS-STATE-P)
                                 ) )))

                (ACTIONS
                  (FILL-GAP '(VERIFIED-STATE) VERBCON OBJCON)
                  (PUTPROP VERBCON T '*AVAIL)
                )
              )
            )
            ;; USING THE STATE VALUE WORDSENSE
            ;; AS IN <SOMETHING> IS VERIFIED
            (GAPREQ
              (TEST (:= STATEVALCON
                        (IF-CLAVAIL TOPCON (G)
                          (AND (PREC G VERBCON)
                               (SEMFEATURE G 'COULDBE-CSTATE-P) ) ) ) )

              (ACTIONS
                (MAKE-UNAVAIL VERBCON)
                (MAKE-UNAVAIL TOPCON)
                (FILL-GAP '(M-STATE-VALUE) STATEVALCON STATECON)
              )
            )
          )
        )
      )
    )
  )

```

Figure 10. Analyzer Dictionary Example

4.2.4.1 Object-Related ERKS Definitions

The object-related ERKS definitions represent the core concepts for both the physical objects and the object interrelationships in the electrical system domain. Generators, constant speed drives (csd), and generator busses are examples of such physical objects which can be described using the fundamental representation of "electrical-component". The more complex object interrelationships are indicated using verbs, such as "connect" and "power", representing functional and other necessary interrelationships.

From discussions with pilots, many concepts were identified and used in the development of this system. This section, of course, in no way presents a complete list of all the possible concepts needed for extensive understanding, but is intended only to provide examples. The section which follows, 4.2.4.2, discusses the ERKS related to the task of function generation. Before enumerating the object-related ERKS, it will be useful to discuss one structure as an example.

Figure 8 illustrates the structure of the object-related ERKS for "electrical-component" and "power". The "superset" slot allows the user to create a representational hierarchy. In this case, the superset for "electrical-component" simply states that the "electrical-component" is a member of a "pobject" class. Likewise, the superset for the ERKS "power" is "kb-actions (knowledge-base actions)". The "intrinsic-slots" represent what is considered to be necessary information to represent the created concept. The information following the slot names "comp-name" and "partof" in "electrical-component" are predicates which restrict the kind of information which can fill the given slot. Similarly for "power" the words following the slot names, "power-source" and "powered-object" are also predicates restricting the value of the fillers that can fill these slots during processing, namely, only "electrical-component" structures. The discussion in Cullingford [10] provides additional details and gives further examples. Now follows the list of the object-related ERKS created for this system.

1. Electrical-system - used to define a prototypical electrical system which includes some possible components. This was used in conjunction with the electrical component ERKS. Roles con1 through con5 represent possible concepts in the electrical system.
2. Electrical-component - this is the representation for the primary object in the user-defined system. The role comp-name represents the name of the component, such as generator or generator-bus. The role partof always indicates that the object is part of the electrical system. This would be useful when the system is expanded to include other aircraft subsystems, such as the hydraulics.
3. Connect - represents a physical connection between two objects. The actor role is used to pick up the actor position in the sentence construct. The parent role represents the component which exists in the parent relationship. The child represents the component which exists in the child relationship in the hierarchy. That is, the parent component is physically above the child component in the circuit structure.
4. Power - represents what the component produces. The role power-source represents the electrical object which produces the power. The powered-object role represents the electrical object which is powered by the power-source.
5. Produce - represents the concept that an object produces something, in this case power(AC or DC). The power-source role represents the object which is producing the power. The produced-object role represents what has been produced by the above mentioned object.
6. Require - represents a state of need by an object or objects. The requiring-object role represents the object or objects that need something. The required-object represents what is needed by the objects.
7. Power-source - used to represent the two basic kinds of power, 115

volts AC and 28 volts DC. The source-value role indicates the source of the power, such as a generator. The power type role indicates either AC or DC power. The power value role indicates two possible values for power in the electrical system.

8. Convert - represents a change of something to something else, as in changing one power voltage to another power voltage. The from and the object roles represent what is to be changed. The to role represents the result of the change. The method role indicates how the change is accomplished. The purpose role captures the reason the conversion took place.
9. Extract - represents a state in which an object gets something from another object, as in a generator extracting energy from the csd. The actor role represents the object doing the extracting. The object role represents what is being extracted. The source role indicates the object from which the extraction is occurring. The source-location role tells the location of the source object.
10. Locate - indicates that an object is at a particular location. The actor role refers to the object in question. The role actor-location describes the object's location or relationship to another object.
11. Use - represents that something is being used or consumed. The actor role represents the electrical object which consumes. The purpose role represents the reason the consumption is occurring.

4.2.4.2 Function-Related ERKS Definitions

The function-related ERKS definitions represent concepts necessary for capturing the functionally-related information in the electrical system domain. Examples of functionally-related information are concepts such as fault detection, fault or sensor verification, and fault diagnosis. After analyzing discussions with pilots, it was determined that such operations as mentioned above

-fault detection, fault/sensor verification, and fault diagnosis- could be best represented by LISP functions for inclusion in a diagnostic control strategy.

In order to capture this information, the following concepts were identified and used in the development of this system. As they relate to function definition or domain-related heuristics, it was determined that three fundamental representations were necessary. These are "cs-actions", "cs-objects", and "cs-state". These concepts and others are discussed below in more detail and are used to represent control strategy objects- objects conceptually related to objects discussed in the control strategy, such as "component-slots"- and to represent concepts describing object states, and descriptive concepts, such as "malfunction" or "more-than-one" and "if-test". This representation scheme, of course, is not intended to be complete but only representative of ideas necessary to capture the more elusive information best represented by LISP functions. Before enumerating the function-related ERKS, it will be useful to discuss one structure as an example.

Figure 8, page 25, illustrates the structure of the function-related ERKS "determine". The "superset" slot allows the user to create a representational hierarchy. In this case, the superset for "determine" simply states that the ERKS "determine" is a member of a "cs-actions" class. The "intrinsic-slots" represent what is considered to be necessary information to represent the created concept. The information following the slot names "purpose" and "method" in "determine" are predicates which restrict the kind of information which can fill the given slot during processing, namely, only "could-be-cs-actions-p" structures. The discussion in Cullingford [10] provides additional details and gives further examples. The list of the function-related ERKS created for this system follows.

1. Cs-actions - a generic category for control strategy actions, such as diagnosis, verification, and detection
2. Cs-objects - a generic category for control strategy objects, especially referencing control strategy related variables,

such as the hypothesis-bulletin-board or the supporting-sensor list.

3. Cs-state - a generic category used to express a control strategy state
4. Cs-fault-state - a structure used to represent the possible states of a given fault, as described in the FLES strategy. The slot called csf-state-value can have three possible values—detected, verified, or diagnosed. The object-name slot is used to hold the name of the faulted component.
5. Cs-object-state - a structure used to represent the possible states of a particular faulted object. The cso-state-value slot can have three possible values – on, off, or bad. These three states are the ones assigned to system components in the FLES diagnostic strategy during the diagnostic process in trying to determine the overall global cause. The object-name slot holds the name of the particular component.
6. Sensor-slot - the structure referencing a sensor frame's slot structure. The slot, named sensor-slot-name, can contain the values of the possible slot names in a given sensor frame structure, namely, reading, enabled, range, reliability, and associated-fault.
7. Component-slot - the structure referencing a component's frame's slot structure. The slot named component-slot-name can contain the values of the possible slot names in a given component frame structure, namely , sensors, children and parents.
8. Determine - a primitive used to express the idea that something is to be ascertained, as in determine a cause. The purpose slot expresses what is to be ascertained, and the method slot is used to express how the value in the purpose slot is to be ascertained.

9. Verify - a primitive used to indicate confirmation of an object fault.
The slot object-state holds the state value of the particular object.
10. Set-Value - a primitive used to indicate that a new value is to be attributed to an object. The value of the object slot represents the object whose value is to be set. The new-value slot is used to represents the objects new value.
11. Malfunction - used to describe a domain object which is out of order. The slot named object-class is filled with a component's name or a component reference. This is the component which has malfunctioned. The slot named object-type is filled with either the value "component" or "sensor" to represent the type of malfunctioning object. The slot named m-state-value represents the possible object states-on, off, or bad-of the named object or the state of a fault - verified.
12. If-test - used to represent a conditional situation having both a possible antecedent and consequent state. The slots antecedent and consequent are self-explanatory.
13. More-than-one - used to represent the concept of more than one object. The value of the slot named object will be the object of which there is more than one.
14. Only-one - used to represent an object of which there is only one. This is, of course, the converse of the concept above. The object slot will hold the value of the single named object.
15. No-children - indicates the concept of no further elements in the fault tree path, or leaves in the fault tree. The slot named object will be filled with a fault in which this state, no children, is true.

16. Fault-children - represents the possible existence of descendants of a named fault. The slot named object will be the value of a fault which has children.
17. Supporting-sensor - this represents the name of a variable used in the diagnostic strategy to indicate a list of sensors which corroborate or support each other. The object slot will be filled with the name of an electrical sensor.
18. Inspect - this is used to represent the idea that an object is to be examined. The object slot names the object being inspected.
19. Alternative - this is used to indicate that some other option is possible. The slot named action will be filled with a value which delineates the possible option.
20. First-element - this is used to represent the car of a list (a list's first element). The object slot will be filled with the name of that list.
21. Disable - this concept represents one possible action which could be done to a sensor in the diagnostic strategy. The other possible action would be to enable. A sensor would be disabled if it is thought to be constantly erroneous. The object slot's value will be the object to be disabled. The method slot will describe how this disabling will be done.
22. Add-to-list - this represents the concept of increasing the size of a list by including or adding other elements into that list. The object slot will then contain the name of the object to be added. The list-name slot will contain the list which is to be modified.

It should be noted that the ERKS defined above represent more concepts than the ones used in example sentences in section 4.2.7. It is, however, ERKS such as these, mentioned in sections 4.2.4.1 and 4.2.4.2 and possibly more, which would

be necessary to properly express the complex ideas found in a domain's heuristics. The ease with which these ERKS structures can be created or modified readily permits the capturing of these complex heuristic concepts.

4.2.5 Wordsense Definitions

Wordsense definitions, representing the possible usage of the words in particular contexts, are the structures filled in or used as fillers as the processing progresses. Figure 9, page 26, shows examples of wordsenses for the word "verify" and other concepts. The process of disambiguating a word's usage is controlled by the Analyzer Dictionary definition for that particular word and will be discussed in the next section, namely 4.2.6. In the electrical system domain, the word "power", related to domain object concepts, had an ambiguous usage. And for the function-related concepts, the word "verify" was used in two ways, as a verb and as a slot value. Processing for both words is similarly controlled by the analyzer dictionary definition as stated above. Using multiple wordsenses expands this system's utility greatly. Attempts to use the disambiguating tools in this module achieved mixed results. For further discussion of wordsense usage, see the discussion in Cullingford [10].

4.2.6 Analyzer Dictionary Definitions

As previously described, the analyzer dictionary definitions drive the actual processing of each word. As discussed in the above section, the disambiguation process is handled by the analyzer dictionary definition. Before describing how the system's Analyzer Dictionary definition files are structured, an explanation of the disambiguation process will follow.

Figure 10, page 27, shows the analyzer dictionary definition for the word "verify". As previously described, "verify" could be used as a verb and as a slot value. For example, its usage as a slot value might be **The fault is verified** describing the state of a fault. Because the word follows a form of the verb

“be”, the word “verify” can be considered semantically to be in the object-spot of the sentence. In its usage as a verb, the word “verify” has been used as an infinitive, that is, it would follow the word “to”, such as **To verify a fault ...**. For each of these possible usages, there would be a different defined wordsense. In the examples in Figure 10, page 27, the wordsense `wsverify1` represents the verb concept and `wsverify2` represents the use as a slot value.

When the word “verify” is first processed, both wordsenses would be placed on the c-list as possible fillers for usage in the sentence representation. This is shown in the list as “(add-word-con `wsverify1` `wsverify2`)”. As the processing proceeds, the analyzer dictionary definition for `verify`, using a `velreq`, a special request designed for disambiguation, attempts to identify the usage of the word “verify” relying heavily on the input sentence’s context. Depending on this context of the particular input sentence, the CCM will be able to decide which usage is the appropriate one, thus disabling the other usage. From this point the usual `gapreq` (`gaprequests` - filling in the structural gaps) test-action pairs take over to use the selected form as a filler or as a structure to be filled. As stated above, disambiguation achieved mixed results. This is because the writing of these dictionary definitions is very tedious and must be handled with great care to succeed. A brief discussion of the overall system structure in regard to the Analyzer Dictionary definitions follows.

Again, because there are two primary types of knowledge handled in this system—domain objects and their relationships, and heuristic and functional knowledge expressed as LISP functions, the analyzer dictionary definitions have been segregated into two files. The file containing the object-related definitions is divided into sections according to their usage in the sentences. The sections are 1) definite/indefinite articles, 2) number definitions, 3) domain objects, 4) prepositions, 5) pronoun, 6) verbs, 7) other modifier concepts, and 8) the phrase dictionary. Likewise, the file containing the function-related definitions is divided into

five parts. The parts are 1) control strategy concepts, 2) prepositions, 3) verbs, 4) other modifier concepts, and 5) phrase dictionary (as described above). This division helped to maintain the modularity necessary for a system of this magnitude. Finally, a brief discussion of the phrase dictionary is necessary to complete the explanation of the analyzer dictionary structure.

The phrase dictionary is a list bound to a variable called `*phrases*`. This data structure allows complex input phrases to be converted to single atomic values. Using the dictionary lookup function, the first element of the matching sublist containing the phrase is returned. For example, if the `*phrases*` list looked like this:

((115v-ac-power one hundred fifteen volts ac power) ...

the atom `115v-ac-power` would be returned when the phrase `one hundred fifteen volts ac power` was found in the original input. Thus, this entire phrase is processed as a single unit facilitating the use of long phrases to describe a single concept. This concludes the discussion of the analyzer dictionary and the associated lookup functions. The discussion in Cullingford [10] will provide additional information. Section 4.2.7 will briefly outline some examples of functioning sentences.

4.2.7 Examples of Functioning Sentences

Because the CCM can be considered as a system independent of the other AKAS modules, it is possible for this module to understand a greater number of sentences with greater complexity than can be understood by the currently limited implementation of the Knowledge Classification/Extraction module. The following sentences are examples of sentences which the CCM can understand. In the following chapter, Chapter 5, there will be examples of sentences which the Knowledge Classification/Extraction module understands along with a brief discussion of how understanding can be increased by adding more information to

the Domain Concept Dictionary.

Because this module is lexicon-driven, several different combinations of the defined vocabulary may exist as viable sentences within the module's definition. This fact is demonstrated below in which generic forms are presented (4, 5, 6). Sentences 1, 2, and 3 demonstrate sentences with the greater complexity mentioned above which are not currently understood by the Knowledge Classification/Extraction module discussed in the following chapter.

1. For the control circuits, lighting and other devices that use DC power, the 115 volts AC power is converted to 28 volts DC by a single phase transformer.
2. Two fourteen volt batteries, which produce 28 volts DC, are used for emergency power.
3. The two engine driven generators extract their energy from the csd which is located between the engine and the generator.
4. The generator bus is connected to the generator.
or more generally <a component> is connected to <a component>
5. The generator powers the generator bus.
or more generally <a component> powers <a component>
or <a component> is powered by <a component>
6. The generator is connected.
or more generally <a component> is connected.
or <a component> is powered.
7. To verify a fault hypothesis see if there is more than one supporting sensor. If there is more than one supporting sensor then the fault is verified.

4.2.8 CCM Vocabulary Learning

Because of the importance of the analyzer dictionary vocabulary, it was determined that vocabulary learning should be incorporated into the system's current implementation. Since the noun-related concepts are the least complex in

implementation, a method for noun learning has been incorporated into the system. Using the process of analogy, the system can learn concepts similar to a preexisting concept (e.g., the generator component is analogous to the engine component.) However, because verb-related concepts are more complex in implementation, a verb-related learning capability has not yet been included.

4.2.9 An Illustrative Example (Part One)

Beginning with this section, there will be a section at the end of Chapters 4, 5, and 6, illustrating the transformation which occurs as the example input proceeds through each module. Beginning this illustrative process, Figures 11a and 11b contain example sentence inputs to the CCM and the CCM generated outputs. These same example sentences, to be continued throughout the chapters which follow, thus the designation Part One, illustrate information as it is transformed by each AKAS module. Thus, in these Part One examples, the English input sentences are converted by the CCM into the Conceptual-Dependency-like form.

Figure 11a shows two input sentences dealing with physical domain objects. The first sentence is in the active voice. The second sentence is the same information expressed in the passive voice. Analyzer dictionary definitions used in the CCM are designed to handle both of these forms. Using the defined words in the analyzer dictionary, the CCM produces the Conceptual-Dependency-like representation shown as CCM output. The overall structure, beginning with the word power, is associated with the verb power in the sentence. The structures beginning with electrical component are used to describe both the engine object and the csd object. Using the specific test-action pairs in the appropriate analyzer dictionary definitions, the CCM strategy integrates these representations into this final form.

In a like manner, the input shown in Figure 11b is transformed into the form shown as the CCM output. The fundamental structure beginning with determine is attached to the verb see in the first sentence. The remaining structure is generated by integrating the individual representations of the words in sentence one and sentence two into one overall representation.

CCM Input:

The engine powers the csd.
The csd is powered by the engine.

CCM Output/KCEM Input:

```
(POWER POWER-SOURCE
  (ELECTRICAL-COMPONENT
    REF (DEF)
    COMP-NAME (ENGINE)
    PARTOF (ELECTRICAL-SYSTEM))
POWERED-OBJECT
  (ELECTRICAL-COMPONENT
    REF (DEF)
    COMP-NAME (CSD)
    PARTOF (ELECTRICAL-SYSTEM)))
```

a. An Object-Related Example

Figure 11. Input Transformation by Concept Comprehension Module

CCM Input:

To verify a fault hypothesis see if there is more than one supporting sensor. If there is more than one supporting sensor then the fault is verified.

CCM Output/KCEM Input:

```
(DETERMINE
  PURPOSE
    (VERIFY
      VERIFIED-STATE
        (MALFUNCTION REF (INDEF)
          OBJECT-CLASS (CURRENT)
          M-STATE-VALUE (VERIFIED)))
  METHOD
    (ACTIONS
      ACTION-1
        (IF-TEST ANTECEDENT
          (MORE-THAN-ONE OBJECT (SUPPORTING-SENSOR))
          CONSEQUENT
            (MALFUNCTION OBJECT-CLASS (CURRENT)
              M-STATE-VALUE (VERIFIED))))))
```

b. A Function-Related Example

Figure 11. Continued

CHAPTER 5

THE KNOWLEDGE CLASSIFICATION/ EXTRACTION MODULE (KCEM)

5.1 Module Purpose

The third system module, the Knowledge Classification/Extraction Module (KCEM), performs two fundamental tasks. The first task is to classify the input into one of two basic groups, namely, cs-action (control-strategy-action) or kb-object (knowledge-base-object). The second task is to extract the necessary information required as input to the Knowledge Incorporation Module. The result from the classification step tells the system how to process the extracted information.

5.2 Module Implementation

Figure 12 illustrates the implementation of this module in its detailed form. The box labeled "Classification Knowledge" represents the strategy for classifying the module's input. As stated above, the classification process tells the system ultimately how to process the input, either as object-related knowledge or as function-related knowledge. The box labeled "Domain Concept Dictionary Extraction Meta-Knowledge" indicates that part of the Domain Concept Dictionary, discussed in section 5.2.3, used in the extraction process to generate the module's output. This output then becomes the input (see Figure 1, page 7) for the Knowledge Incorporation Module to be discussed in Chapter 6. The following sections detail both the classification and the extraction strategies.

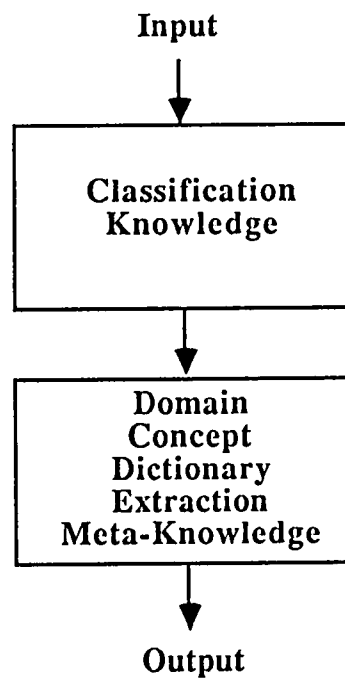


Figure 12. The Knowledge Classification/Extraction Module (detailed)

5.2.1 The Classification Strategy

Currently, the classification strategy is a simple one based on the central primitive of the input, that is, the first element of the input representation. For example, in the figure on page 51, the central primitive is "power". According to the classification knowledge, this would be classified as a "kb-object" indicating that this primitive should have physical objects associated with it. After classifying the input, the appropriate information is extracted using meta-knowledge contained in the Domain Concept Dictionary. At this point, the result of the extraction process is passed to the Knowledge Incorporation Module.

In a like manner, the classification strategy for function-related knowledge inspects the central primitive of the representation. In the figure on page 52, the central primitive is "determine." By using the classification knowledge, AKAS understands that this representation should be classified as a "cs-action" and be processed accordingly.

5.2.2 The Extraction Process

The extraction process is simply building the pathways to extract information from locations in the representation where the Domain Concept Dictionary meta-knowledge indicates information should be found. For example, with the concept "power", the Domain Concept Dictionary (see Figure 13) indicates the primary roles of importance are those roles named, "power-source" and "powered-object". The system knows, that using this information in conjunction with the slot name for a component, "comp-name", it will find the names of component objects. The building of the appropriate extraction pathways is then the concatenation of these role names. For example, to extract the filler value (engine) from the example in the figure on page 51, the pathway would be the list '(power-source comp-name). Using this constructed pathway as an argument to a function called "grf (get-role-filler)" will retrieve the desired value.

```
(setf power
  (make-instance 'dcd
    :role-list '(power-source powered-object)
    :path-frame-mapping
    '(((power-source parents)
      (powered-object children))))))
```

```
(setf determine
  (make-instance 'dcd
    :role-list '(purpose method)
    :path-function-mapping
    '(((purpose (name lambda-list))
      (method (function-body))))))
```

```
(setf more-than-one
  (make-instance 'dcd
    :role-list '(object)
    :functional-primitive
    '(not (null (cdr more-than-one-arg)))
    :functional-primitive-argument
    'more-than-one-arg
    :function-demon 'more-than-one-demon))
```

Figure 13. Domain Concept Dictionary Entries

In the more complex function-related information, the Domain Concept Dictionary also indicates the anticipated locations of desired information in the input representation. In the case of the example on page 52, the domain concept entry, (see Figure 13) "determine", tells the system that the name and lambda-list information will be found in the "purpose" pathway of the input. Likewise, the dictionary indicates that the function body will correspond to what is found in the "method" pathway.

5.2.3 The Domain Concept Dictionary

Critical to both the extraction process and the operationalization process (to be discussed in Chapter 6) is the information found in the Domain Concept Dictionary. This information indicates where the appropriate knowledge in the input representation can be found to accomplish both processes. In this section the Domain Concept Dictionary's role in the extraction process will be discussed in more detail. First, however, it is necessary to describe the overall structure of a dictionary entry.

Each dictionary entry is a flavor structure called "dcd" with instance variables representing meta-knowledge necessary for both the creation of LISP frames and LISP functions. The instance variables include the following:

- 1) role-list - a list of the role-names in the particular primitive,
- 2) path-frame-mapping - the relationship between a path in the CCM output structure and a slot-value in a created frame,
- 3) functional-primitive - a LISP expression representing the associated concept,
- 4) functional-primitive-argument - the argument atom found in the functional-primitive,
- 5) function-demon - the name of an associated intelligent function used to process information associated with the primitive concept, and
- 6) path-function-mapping - the relationship between a path in the CCM output and its use in a LISP function definition.

Figure 13 presents three example entries. In each specific entry the information necessary to perform both processes is included. These entries in Figure 13 will be used to illustrate briefly the utilization of this dictionary's meta-knowledge in the knowledge extraction process.

In the extraction process for domain objects, both the "role-list" and the "path-frame-mapping" instance variables are used. Briefly, the "role-list" is used to generate the access paths to the domain objects mentioned in the CCM output. This access permits creation of instances of the mentioned object with the number of instances being controlled by the type of aircraft determined at login time. For example, with the entry "power", the "role-list" instance variable (see Figure 13) indicates that the important roles are those named, "power-source" and "powered-object". Using this information in conjunction with the slot name for a component, "comp-name", the extraction process will find the names of component objects in the knowledge representation. The building of the appropriate extraction pathways is then the concatenation of these role names. For example, to extract the filler value, (engine), from the example on page 52, the pathway would be the list '(power-source comp-name). Using this constructed pathway as an argument to a function called "grf (get-role-filler)", the function would retrieve the desired value. After all component values are extracted the knowledge is converted into the form, "(frame-components (engine) (csd))", as shown in the example figure on page 52.

The path-frame-mapping instance variable tells the system how to establish the relationships between the extracted information and the LISP frame slots. In the case of "power" the "path-frame-mapping" value indicates that the "power-source" retrieved value will be the value of the "parents" slot. Likewise, the value extracted using the "powered-object" will be the value for the "children" slot in any given frame instance. In the example on page 52, the two relationship lists are created using the information described above. In this case, the two lists,

7. To verify a fault hypothesis see if there is more than one supporting sensor. If there is more than one supporting sensor then the fault is verified.

Currently the only object-related concepts KCEM can understand are representations related to "power" and "connect". In order to make it possible for the KCEM to understand more domain-related object concepts, it is only necessary to make additional entries in the Domain Concept Dictionary. Additional entries in the DCD, similar to the power entry (see Figure 13), would enable the system to understand other new concepts.

To enable the understanding process for function-related input, the process is a bit more complicated. In this case new entries must also be added to the DCD similar to the "determine" and "more-than-one entries" (see Figure 13). In addition to these entries, it is also necessary to handcraft a function demon which understands how to interpret the elements in the input. Further discussion about the function demons will occur in Chapter 6.

5.2.4 An Illustrative Example (Part Two)

Figures 14a and 14b contain example inputs to the KCEM and the KCEM generated outputs. These examples, continued from the previous chapter, have been used as examples illustrating points throughout the above text discussing the classification and extraction processes. However, for the sake of consistency the following paragraphs will briefly discuss each of these example figures again.

Both Figures 14a and 14b illustrate the before and after result of knowledge as it has been transformed by the KCEM module. Both figures at the top illustrate the input necessary for the KCEM to perform its task. The input to the KCEM is, of course, indicated to be the output of the CCM. Using the classification knowledge discussed above, the KCEM has classified each of these example representations as "kb-object" (Figure 14a) or "cs-object" (Figure 14b) depending on

CCM Output/KCEM Input:

```
(POWER POWER-SOURCE
  (ELECTRICAL-COMPONENT
    REF (DEF)
    COMP-NAME (ENGINE)
    PARTOF (ELECTRICAL-SYSTEM))
POWERED-OBJECT
  (ELECTRICAL-COMPONENT
    REF (DEF)
    COMP-NAME (CSD)
    PARTOF (ELECTRICAL-SYSTEM)))
```

KCEM Output/KIM Input:

Classification:

KB-OBJECT

Extracted Information:

```
(FRAME-COMPONENTS (ENGINE) (CSD))
(RELATIONSHIP (PARENTS (ENGINE) (CSD)))
(RELATIONSHIP (CHILDREN (CSD) (ENGINE)))
```

a. An Object-Related Example

**Figure 14. Input Transformation by Knowledge
Classification/Extraction Module**

CCM Output/KCEM Input:

```
(DETERMINE
  PURPOSE
    (VERIFY
      VERIFIED-STATE
        (MALFUNCTION REF (INDEF)
          OBJECT-CLASS (CURRENT)
          M-STATE-VALUE (VERIFIED)))
    METHOD
      (ACTIONS
        ACTION-1
          (IF-TEST ANTECEDENT
            (MORE-THAN-ONE OBJECT (SUPPORTING-SENSOR))
            CONSEQUENT
              (MALFUNCTION OBJECT-CLASS (CURRENT)
                M-STATE-VALUE (VERIFIED))))))
```

KCEM Output/KIM Input:

Classification:

CS-ACTION

Extracted Information:

```
(FUNCTION-NAME VERIFY)
(FUNCTION-LAMBDA-LIST (CURRENT))
(FUNCTION-BODY
  (ACTIONS
    ACTION-1
      (IF-TEST ANTECEDENT
        (MORE-THAN-ONE OBJECT (SUPPORTING-SENSOR))
        CONSEQUENT
          (MALFUNCTION OBJECT-CLASS (CURRENT)
            M-STATE-VALUE (VERIFIED))))))
```

b. A Function-Related Example

Figure 14. Continued

the classification of the central primitive (the first element of the representation).

Using the information contained in the Domain Concept Dictionary, the module has also created the outputs indicated as the extracted information in each figure. Using the extraction strategy detailed above, the KCEM has produced the frame-components and relationships lists for the object-related examples. In the case of the function-related knowledge, the similar extraction and construction process have produced lists showing the function-name, the function-lambda-list, and the function-body to be used by the next module for the actual creation step. Up to this point, parts one and two of the object-related and function-related examples have shown the transformation of sentences into Conceptual-Dependency-like representations and these representations into knowledge about frame components and relationships, or function-name, function-lambda-list and function-body.

CHAPTER 6

THE KNOWLEDGE INCORPORATION MODULE (KIM)

6.1 Module Purpose

It is the purpose of the Knowledge Incorporation Module (KIM), the final module in the AKAS architecture, to operationalize or convert its input into machine usable form, either actual LISP frames or LISP functions. In performing its intended purpose, the system handles the integration of knowledge into new or existing LISP frames, and provides a limited means for detecting and dealing with conflicting information within the frame representation.

As stated above, the operationalization process also handles the creation of LISP functions. This involves the use of the Domain Concept Dictionary, discussed in Chapter 5, and the use of function demons to complete the operationalization process. Currently, there is no capability to detect or deal with conflicting information pertaining to the structure of a LISP function. That is, once a function has been created, it cannot be modified. (A brief discussion of a not yet implemented portion of this module, the LISP Function Dictionary, may indicate methods to be used in the modification of a function.)

6.2 Module Implementation

Figure 15 shows a more detailed form of this module. The central operationalization process, using the module's input in conjunction with the Domain Concept Dictionary, the function demons, the representative frame structures, and frame relationships, is able to convert the input into either a frame for the

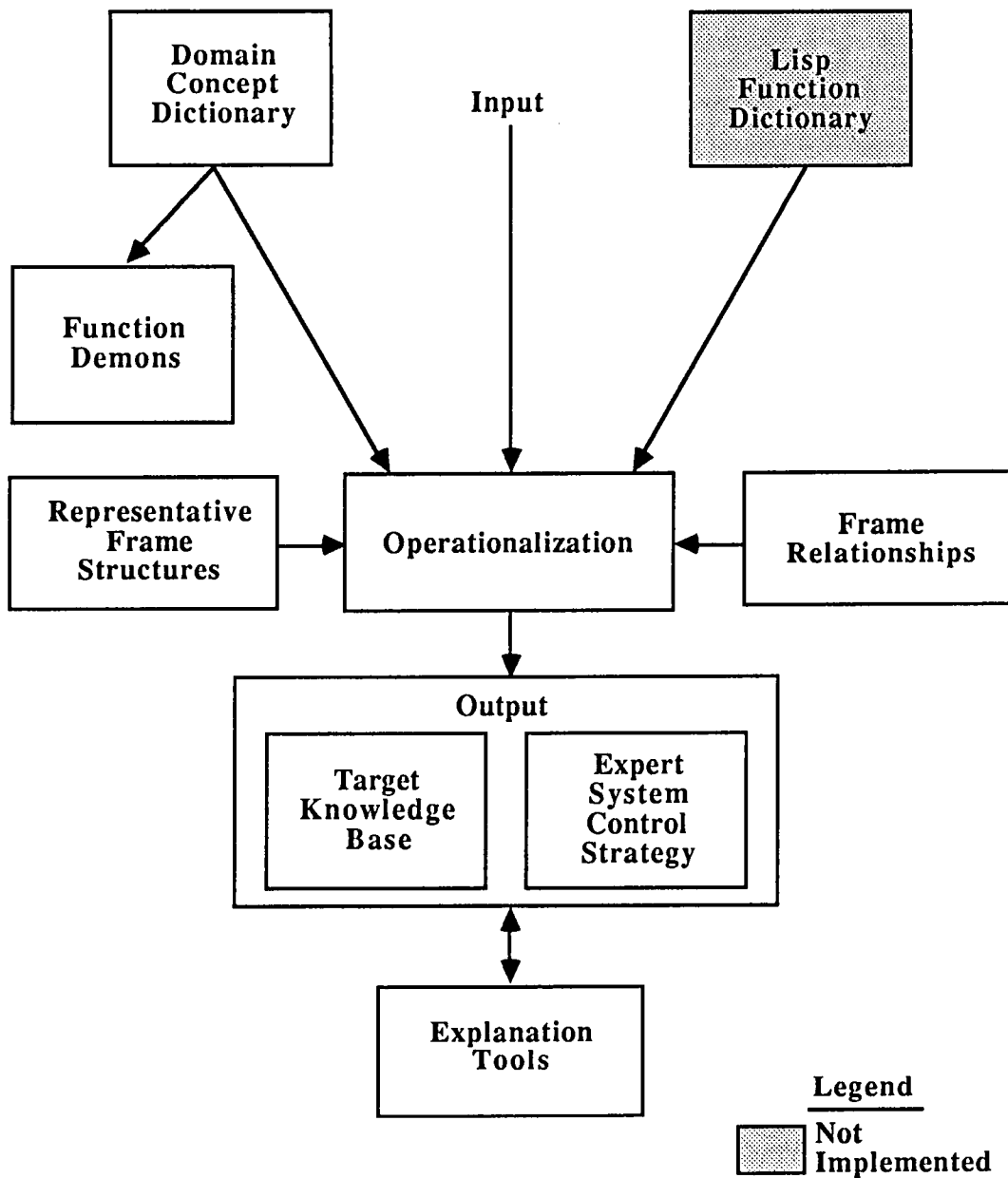


Figure 15. The Knowledge Incorporation Module (detailed)

target knowledge base or a function for the expert system's control strategy. The boxes marked "Target Knowledge Base" and "Expert System's Control Strategy" are intended to represent the repository for the knowledge generated by AKAS. The box marked "Explanation Tools" indicates the system's ability to provide a limited explanation of how a piece of information was treated.

Receiving information from the Knowledge Classification/Extraction Module, the Knowledge Incorporation Module, using the Domain Concept Dictionary, converts this information into the appropriate LISP forms. For LISP objects, knowledge integration is simply checking for the existence of a referenced object. If the object is unknown, it is created and the information is added to this new object. If the object already exists, only the new information is added in the appropriate place. In the case of conflicting information, currently implemented only for LISP objects, the user is given the option of deciding what information is to finally be included. For example, if the user first states that **The engine powers the csd** and then states that **The engine powers the generator**, this will be detected as a conflict and the user is given an opportunity to resolve it. For LISP functions, there is no current capability to add to the LISP expressions which compose the function after it is created.

6.2.1 The Domain Concept Dictionary

Previously discussed in Chapter 5, as indicated above, a discussion of the Domain Concept Dictionary is provided here to explain its role in the operationalization process. Again the operationalization process converts the extracted knowledge derived from the Knowledge Classification/Extraction Module into the appropriate form. Before further discussion, a review of the dictionary structure is necessary.

Each dictionary entry (see Figure 13, page 45, for example entries) is a flavor structure called "dcd" with instance variables representing meta-knowledge necessary for both the creation of LISP frames and LISP functions. The instance

variables include the following:

- 1) role-list - a list of the role-names in the particular primitive,
- 2) path-frame-mapping - the relationship between a path in the CCM output structure and a slot-value in a created frame,
- 3) functional-primitive - a LISP expression representing the associated concept,
- 4) functional-primitive-argument - the argument atom found in the functional-primitive,
- 5) function-demon - the name of an associated intelligent function used to process information associated with the primitive concept,
- 6) path-function-mapping - the relationship between a path in the CCM output and its use in a LISP function definition.

In the construction of domain objects, only the extracted information is used. The use of the Domain Concept Dictionary, as discussed in Chapter 5, has already determined the name of the frame components and the frame interrelationships. For example, on page 63, the extracted information contains three lists mentioning the frame-components and the component interrelationships. The first list indicates that the frame-components are the "(engine)" and the "(csd)". Using information regarding the type of aircraft determined at login time, the system is now able to create the appropriate number of instances for each frame-component mentioned in the extracted information. From the structure of the next two lists indicating relationship knowledge, the system can then place the appropriate knowledge into the appropriate frame slots. That is, the second list, "(relationship (parents (engine) (csd)))", saying that the "engine" is the parent of the "csd", tells the system to place instances of the engine into the "parents" slot of all instances of the "csd" object. Likewise, the information in the third list, "(relationship (children (csd)(engine)))" indicating that the "csd is the child of the engine", tells the system that symbols representing instances of the "csd" be placed as "children" slot values in each "engine" instance. The fact that this

has occurred is illustrated by the KIM output indicated also on page 63.

Constructing LISP functions is, however, more complex because it not only involves the use of the extracted knowledge but also information associated with instance variables in the DCD entries, namely, functional-primitive, functional-primitive-argument, and function-demon. Each of these values provides additional information to be used during the operationalization process producing the actual function. Briefly, the value of the "functional-primitive" slot for a given concept provides a template LISP expression representing that concept which can be used during the function creation process. The value of the "functional-primitive-argument " aids the associated function demon in the substitution process when the appropriate value found in the extracted information is substituted into the LISP expression. Lastly, the value of the "function-demon" provides a specific strategy to process information about a particular structure.

Using the extracted information as a guide, the operationalization process directly uses the function-name and function-lambda-list without further conversion. Looking at the figure on page 64, one sees that the function name and lambda-list are the same as the extracted information. It is the function-body information which uses the above mentioned instance variable values to convert the extracted information. Again looking at page 64 and also at the "more-than-one" example entry in Figure 13, page 45, it can be seen that the functional-primitive value was used in the actual function definition shown on page 64. (The "cond" in the function definition came from the "if-test" DCD entry.) It was the role of the particular function demon called "more-than-one-demon" to substitute the "supporting-sensor" value in place of the "functional-primitive-argument" value, namely, "more-than-one-arg" to create the LISP expression in the verify function. It was also the function demon which found the state-value "verified" and converted this into the "t" value at the end of the condition clause. The next section discusses the function demons in more detail.

6.2.2 The Function Demons

Used solely in the function creation process, function demons are designed specifically for a given concept. Their purpose is to convert a structure related to a given concept into a machine-usable form. Each concept's associated function demon is the value of the "function-demon" instance variable in the concept's Domain Concept Dictionary entry. Each demon is currently handcrafted to return desired values depending on what is contained in the input. Figure 16 illustrates the function demon designed for the concept "more-than-one".

As discussed above, it is the extracted function-body information which is converted by using the above mentioned instance variable values, namely, the "functional-primitive", the "functional-primitive-argument", and the "function-demon". Again looking at page 64 and also at the "more-than-one" example entry in Figure 13, page 45, it can be seen that the functional-primitive value was used in the actual function definition shown on page 64. (The "cond" in the function definition came from the "if-test" DCD entry.) It was the role of the particular function demon called "more-than-one-demon" to substitute the "supporting-sensor" value in place of the "functional-primitive-argument" value, namely "more-than-one-arg" to create the LISP expression in the verify function. It was also the function demon which found the state-value "verified" and converted this into the "t" value at the end of the condition clause. The repeated information found in the "object-class" of malfunction was discarded during the function creation. For more complex structures, the process of substituting the result of each function demon's operation associated with a given primitive concept continues until the entire function-body expression is processed.

```

(DEFUN MORE-THAN-ONE-DEMON (LIST)
  (LET* ( (OBJ (MEMBER 'OBJECT LIST) )
          (OBJ-FILLER (CADR OBJ) )
          (TEMP NIL)
        )
    (COND ( (EQUAL (TYPEP (CAR OBJ-FILLER) ) 'DCD)
            (FUNCALL (DCD-FUNCTION-DEMON (EVAL (CAR OBJ-FILLER) ) ) (CDR OBJ-FILLER) ) )
          (T (SETQ TEMP
                    (SUBST (CAR OBJ-FILLER) 'MORE-THAN-ONE)-ARG
                          (DCD-FUNCTIONAL-PRIMITIVE (EVAL MORE-THAN-ONE) ) ) )
            (FORMAT T "~%temp = ~a" TEMP)
            TEMP) ) ) )

```

Figure 16. An Example Function Demon

6.2.3 Representative Frame Structures (RFS) and Frame Relationships

Representative frame structure (RFS) is specifically the desired structure for a given object in the given domain. In the research reported here, both sensor and component frame structures were delineated, although no sensor frames were ever created. Currently, the values for the sensor and component frame structures are simple lists containing the desired slot values for each object type. The list for the component frame structure is "((sensors = ()) (children = ()) (parents = ()))". Similarly, the sensor frame structure is the list "((reading = ()) (enabled = 'ERR')(range = ())(reliability = 0.95)(associated-fault = ()))". For a different domain, similar structures could be created for each different class of objects.

The frame relationships, or more specifically the inter-frame relationships, are explicitly expressed in the object-related primitives found in the Domain Concept Dictionary. This information is expressed in the "path-frame-mapping" variable found in each entry. As explained before, this variable expresses the mapping from the input to the precise place in a created frame, a slot described in the frame structure, where information is to be placed.

6.2.4 The LISP Function Dictionary

Not currently implemented as a part of this module, the LISP Function Dictionary was envisioned as a method to check the structure of LISP functions created by the system. It would be used to check each LISP expression to determine if the expression was properly structured. If the expression is a function call, it would be checked for its validity as a defined function. It would also check the number, order and type of arguments to insure their validity before incorporation into an actual control strategy. It would be possible to make entries for not only predefined LISP language functions but also for functions specifically written for a given application.

This dictionary and associated functions could also be designed to modify

functions after initial creation. It might be possible to indicate to the system what has to be changed and for a new value to replace the old one. As this portion has not yet been implemented, the possibility of performing this modification function is yet unproven, but considered feasible by this author.

6.2.5 Explanation Tools

Because the system maintains a relationship between the given input and the produced output, it is possible to briefly explain the creation of domain objects and functions by showing which sentences invoked the creation of particular structures. This explanation facility grew as a result of the explanation facilities found in traditional knowledge-based system architectures.

This feature is implemented using two tables, a components-table and a function-table. During the course of the interaction between the system and the user, the system updates each table with information related to the given class of input. For the components-table, an entry in the table consists of three parts, the general name of the part, such as generator, the specific system generated instance names for each created frame, such as generator0 and generator1, and the sentence identifier which captures the specific knowledge, such as c20. The function-table entry consists of two parts, the function name and the identifier which captures the sentence or sentences expressing the information which is later converted into a function.

6.2.6 An Illustrative Example (Part Three)

Figures 17a and 17b contain example inputs to the KIM and the KIM generated outputs. These examples, continued from the previous chapter, illustrate information as it has been transformed by the KIM. The process transforming the extracted knowledge has been previously referenced in sections above, but will again be briefly described here.

The knowledge shown as extracted information in Figures 17a and 17b was

KCEM Output/KIM Input:

Classification:

KB-OBJECT

Extracted Information:

(FRAME-COMPONENTS (ENGINE) (CSD))
(RELATIONSHIP (PARENTS (ENGINE) (CSD)))
(RELATIONSHIP (CHILDREN (CSD) (ENGINE)))

KIM Output:

(form CSD1	(form CSD0
isa component	isa component
with	with
(parents = ENGINE1)	(parents = ENGINE0)
(children = GENERATOR1)	(children = GENERATOR0)
(sensors = nil))	(sensors = nil))
 (form ENGINE1	 (form ENGINE0
isa component	isa component
with	with
(parents = nil)	(parents = nil)
(children = CSD1)	(children = CSD0)
(sensors = nil))	(sensors = nil))

a. An Object-Related Example

Figure 17. Input Transformation by Knowledge Incorporation Module

KCEM Output/KIM Input:

Classification:

CS-ACTION

Extracted Information:

(FUNCTION-NAME VERIFY)

(FUNCTION-LAMBDA-LIST (CURRENT))

(FUNCTION-BODY

(ACTIONS

ACTION-1

(IF-TEST ANTECEDENT

(MORE-THAN-ONE OBJECT (SUPPORTING-SENSOR))

CONSEQUENT

(MALFUNCTION OBJECT-CLASS (CURRENT)

M-STATE-VALUE (VERIFIED))))))

KIM Output:

(DEFUN VERIFY (CURRENT)

(COND ((NOT (NULL (CDR SUPPORTING-SENSOR))) T)))

b. A Function-Related Example

Figure 17. Continued

generated as a result of the operation of the Knowledge Classification/Extraction Module. This information was then passed to the appropriate processing function determined by the information's classification category, either "kb-object" or "cs-action". At this point, depending on the nature of the information, either frames or a LISP function were created.

In Figure 17a the extracted information shows three lists, namely frame-components and two relationship lists. The frame-components list tells the system to create two classes of objects, namely, "engines" and "csds". The additional relationship information indicates how slots in the created instances of these objects should be filled. That is, the second list, "(relationship (parents (engine) (csd)))", saying that the "engine" is the parent of the "csd", tells the system to place instances of the engine into the "parents" slot of all instances of the "csd" object. Likewise, the information in the third list, "(relationship (children (csd)(engine)))" indicating that the "csd is the child of the engine", tells the system that symbols representing instances of the "csd" be placed as "children" slot values in each "engine" instance. The output portion of Figure 17a indicates what has been created from the extracted information from the operationalization process.

Figure 17b illustrates the operationalization of extracted information into a LISP function. As described above, the first two lists, namely, "(function-name verify)" and "(function-name-lambda-list (current))" have been used directly in the LISP function without any transformation. It is the function-body list (see Figure 17b) which is transformed using the values of the "functional-primitive", the "functional-primitive-argument", and the value of the "function-demon". It is the role of the appropriate function demon to use the values of these other two variables to create the LISP function in its final form. This output generated by the Knowledge Incorporation Module is the final form to be used by either the target knowledge base or the expert system's control strategy.

CHAPTER 7

FUTURE WORK AND CONCLUSIONS

7.1 Future Work

As with any system, work on the improvement of fundamental concepts and system features can be an ongoing process. In the case of AKAS, further improvements would be necessary to facilitate its use by a naive computer user.

One major area requiring improvement would be the communication between the system and the user. The mixed-initiative dialogue, as advocated in the KLAUS philosophy, needs further improvement in telling the user what sentences are understood and which ones are not. In the course of AKAS' development, the viable sentences were readily known by the authors because these sentences were the desired test cases. Because a naive user would not have this prior information, more guidance would be needed to direct the user in entering sentences that are understood.

Three possible solutions have been proposed. First, perhaps this guidance could be accomplished by some type of preprocessor in which the input sentence could be scanned with the system determining whether the word order of the sentence is understood or not and telling the user what is not understood. Two other alternatives are also possible solutions to this problem. One would be to use pull-down menus indicating the possible primitives. When a primitive is selected, another pull-down menu could delineate the acceptable arguments for the given primitive. For example, in the case of the object-related concept "power", it would be possible to indicate that only instances of electrical objects were acceptable. The final alternative, especially in the case of domain-related objects, would be to use an icon-based graphical representation to describe the relative arrangement of domain objects and their functional relationships.

Methods to improve user-system communication, with regard to function-related information, are not as clear. Perhaps the use of the pull-down menus would be acceptable for this mode also. Only implementation of these ideas would prove or disprove their utility. Also, the limited aspects of the vocabulary learning, mentioned above, could be expanded in order to lessen the amount of unknown concepts, thus easing the burden on the user.

Another important aspect of the design, not yet implemented, would be to allow the user to undo concepts he has previously created. This idea is advocated in the KLAUS philosophy and is worthy of inclusion into a viable system. This feature could be accomplished through the use of data dependencies. Establishing dependencies between concepts as they are referenced would permit the linking of information as it becomes available. If a piece of information is removed, then all elements dependent on that information could also be removed.

Because the vocabulary used in the concept comprehension module is critical and complex, it would be worthwhile to create a user interface to aid solely in the construction of the vocabulary. This tool would make it possible to create a seed vocabulary for use in the domain of one's choice. Coupled with an expanded ability to learn domain concepts, the concept comprehension module could become a very powerful system in any domain context.

7.2 Conclusions

The purpose of this work has been to demonstrate that it is possible to capture both domain and procedural types of knowledge. Within the framework of this architecture containing a rich natural language interface, this author believes that the possibility of capturing this information has been demonstrated. It should be pointed out, however, that this author does not believe that this system, as currently configured, can completely replace the duties of the knowledge engineer but can act only as an augmentation tool.

The greatest success of expert system technology has been to work within limited domains. Likewise, the knowledge acquisition systems have been restricted to a given domain and specific knowledge representation schemes. The fact that there have been limitations is illustrated by the example systems mentioned in the introduction, namely, TEIRESIAS, KAS, NANOKLAUS, and William Clifford's flightline robot system. Like these other systems, AKAS is also limited because it has been developed for the aircraft diagnostic domain with special emphasis on a frame representation and heuristics based on diagnostic strategies and functional relationships. Unlike the systems mentioned in the thesis introduction, the AKAS system architecture permits some versatility in the substitution of other domain vocabularies as long as fundamental concepts are included and the system deals with domain-related objects and control strategy concepts. It is this author's belief that, given the appropriate information required by the domain concept dictionary, it is possible to capture not only domain-related (object) information and relationships, but also the more complex function-related information. With only minor modification the system could be designed to create either a frame representation or an object-oriented one. Of course, this system is also not without its weaknesses and drawbacks.

A major drawback is that the demons which created the functional definitions must be hand-crafted in the present system. An area of improvement in the system would be to automate the demon creation process as well as the initial creation of the seed vocabulary.

As indicated above and in the section on further research, there are areas in which this system could be strengthened. This research, along with the other knowledge acquisition research efforts, points out that the knowledge acquisition process continues to be a difficult one to readily and easily capture. This thesis has offered yet another approach to this problem of capturing knowledge.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [1.] Ali, M., D. A. Scharnhorst, and H. J. Ferber, "Automatic Fault Diagnosis in the Electrical System of an Airplane," The University of Tennessee Space Institute, Technical Report 85/02, May 1985.
- [2.] Ali, M., D. A. Scharnhorst, Ai, C.-S. and H. J. Ferber, "Expert Systems for Real-Time Fault Monitoring and Diagnosis in Space Vehicles," Conference on Artificial Intelligence: From Outer Space ... Down to Earth, Huntsville, AL, Oct. 1985.
- [3.] Ali, M. and D. A. Scharnhorst, "Sensor-Based Fault Diagnosis in a Flight Expert System," Proceedings of the Second Conference on Artificial Intelligence Applications, Miami Beach, FL, December 11-13, 1985.
- [4.] Ali, M., D. A. Scharnhorst, Ai, C. -S. and H. J. Ferber, "A Flight Expert System (FLES) for On-board Fault Monitoring and Diagnosis," Proceedings of SPIE Technical Symposium on The Applications of Artificial Intelligence III, Orlando, FL, April 1-3, 1986.
- [5.] Ali, M., C-S. Ai, and H. J. Ferber, "Semantic Based Man-Machine Interface for Real Time Communication", 1986 IEEE Military Communications Conference, Monterey, CA, October 5-9, 1986.
- [6.] Ali, M. and Eddie Washington, "A Plant Intelligent Supervisory Control Expert System," ACM-IEEE Fall Joint Computer Conference, Dallas, TX, November 2-6, 1986.
- [7.] Ali, M., C-S. Ai, and H. Ferber, "A Natural Language Interface for Real-Time Communication," Artificial Intelligence for Space Applications, Huntsville, AL, November 13-14, 1986.
- [8.] Ali, Moonis and Roger Crawford, "Rocket Engine Control and Monitoring Expert Systems (RECMES)," AIAA/ASME/ASEE 24th Joint Propulsion Conference, Boston, MA, July 11-13, 1988.

- [9.] Clifford, William M., Captain, USAF Automating Knowledge Acquisition in a Flightline Robot, Defense Technical Information Center Technical Report AD A163829, Air Force Institute of Technology Report No. AFIT/GE/ENG/85D-7, December 1985.
- [10.] Cullingford, Richard E., Natural Language Processing: A Knowledge Engineering Approach, Totowa, NJ: Rowman and Littlefield, 1986.
- [11.] Davis, Randall and Douglas B. Lenat, Knowledge-Based Systems in Artificial Intelligence, New York: McGraw-Hill International Book Company, 1982.
- [12.] Dietz, W.E. and Moonis Ali, "Qualitative and Temporal Reasoning in Engine Behavior Analysis and Fault Diagnosis," The Third Conference on Artificial Intelligence for Space Applications, Huntsville, AL, Nov 2-3, 1987.
- [13.] Dietz, W.E., E.L. Kiech, and Moonis Ali, "Pattern Based Fault Diagnosis Using Neural Networks," The First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, The University of Tennessee Space Institute, Tullahoma, TN, June 1-3, 1988, pp. 13-23.
- [14.] Ferber, H. and Moonis Ali, "Automatic Acquisition of Domain and Procedural Knowledge," The First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, The University of Tennessee Space Institute, Tullahoma, TN, June 1-3, 1988, pp. 762-771.
- [15.] Fiegenbaum, Edward A., Knowledge Engineering: The Applied Side of Artificial Intelligence, Defense Technical Information Center Technical Report AD A092574, Stanford Heuristics Programming Project Report No. HPP-80-21, Department of Computer Science, Stanford University Report No. STAN-CS-80-812, September 1980.
- [16.] Gupta, U. and Moonis Ali, "Hierarchical Representation and Machine Learning from Faulty Jet Engine Behavioral Examples to Detect Real Time Abnormal Conditions," The First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, The University of Tennessee Space Institute, Tullahoma, TN, June 1-3, 1988, pp. 710-720.

- [17.] Haas, Norman and Gary G. Hendrix, "Learning By Being Told: Acquiring Knowledge for Information Management." Machine Learning: An Artificial Intelligence Approach. Michalski, Ryszard S., Jaime G. Carbonell, and Tom M. Mitchell, eds., Los Altos, CA: Morgan Kaufmann Publishers, Inc., 1983.
- [18.] Hayes-Roth, Frederick, Philip Klahr, and David J. Mostow, "Knowledge Acquisition, Knowledge Programming, and Knowledge Refinement ", book unknown, pp.310-349. (Originally appeared as R-2540-NSF, The Rand Corporation, May 1980.)
- [19.] Hayes-Roth, Frederick, Donald A. Waterman, and Douglas B. Lenat, eds., Building Expert Systems, Reading, MA: Addison-Wesley Publishing Company, Inc., 1983.
- [20.] Mostow, Jack and Frederick Hayes-Roth, "Machine-aided Heuristic Programming : A Paradigm for Knowledge Engineering" Rand: Santa Monica, CA, Report number N-1007-NSF, February 1979.
- [21.] Subramanian, C. and Moonis Ali, "Detecting Abnormal Situations from Real Time Power Plant Data Using Machine Learning," The First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, The University of Tennessee Space Institute, Tullahoma, TN, June 1-3, 1988, pp. 738-746.
- [22.] Washington, E. and Moonis Ali, "PISCES: An Expert System for Coal Fired Power Plant Monitoring and Diagnostics," The First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, The University of Tennessee Space Institute, Tullahoma, TN, June 1-3, 1988, pp. 87-93.

VITA

Harry Joseph Ferber II was born in Hampton, Virginia on August 15, 1948 to Harry Joseph and Velma Turnage Ferber. He attended elementary and junior high schools and was graduated from Kecoughtan High School in June 1966. The following September he entered Wake Forest University. In July 1974, after a four-year tour of duty in the United States Army, he reentered Wake Forest University completing a B.A. in Psychology in June 1975.

After working approximately six years, he returned to school at Old Dominion University in September 1981. In September 1984, after graduating in May with a B.S.C.S. from Old Dominion University, he entered the University of Tennessee Space Institute to begin work on a Master's degree in Computer Science with emphasis in the area of Artificial Intelligence.

The author is a member of The American Association for Artificial Intelligence (AAAI), the Association for Computer Machinery(ACM), and The Institute of Electrical and Electronics Engineers, Inc. Computer Society (IEEE Computer Society).