To the Graduate Council:

I am submitting a thesis written by Patrick Leroy Sherley entitled "Design of a Knowledge Based Interface for a Voice Actuated Computer Graphics Imaging System." 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.

Seung C. Lee, Major Professor

We have read this thesis
and recommend its acceptance:

Bruce Whitehead

Alfonso Pujol, Jr.

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 rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Request for permission for extensive quotation from or reproduction of this thesis in whole or in parts may be granted by the copyright holder.

Signature Patrick Shelby

Date April 20, 1990

DESIGN OF A KNOWLEDGE BASED INTERFACE FOR A VOICE ACTUATED COMPUTER GRAPHICS IMAGING SYSTEM

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Patrick Leroy Sherley

May 1990

Copyright © Patrick Leroy Sherley, 1990
All rights reserved

DEDICATION

I would like to dedicate this thesis to my wife, Toya L. Sherley, who urged and motivated me to attend graduate school, and to my parents, Rev. James L. Sherley and Mrs. Ruby L. Sherley, for their love, encouragement, and unfailing support.

ACKNOWLEDGEMETS

I would like to thank my thesis advisor, Dr. Seung C. Lee for the patience and understanding he continuously displayed during the preparation of this thesis. I would like to also thank Dr. Bruce Whitehead for serving on my thesis committe, and Mr. John Meadows for his numerous insights and suggestions.

I would like to give special thanks to my third committee member, Dr. Alfonso Pujol, for the many grueling hours he spent aiding me in the preparation of this thesis, to ensure it was written in a coherent, scholarly style. I am also grateful to Dr. Pujol for allowing me to work on the research project, on which the thesis was based.

Lastly, but most importantly, I would like to thank God for making this thesis, and everything else, possible.

ABSTRACT

Research was conducted to develop a voice actuated computer graphics imaging system to provide a means of rendering complex computer languages and input/output modalities transparent to experienced and inexperienced users. The system is used for reconstructing and displaying computed tomography and magnetic resonance imaging scan data. A knowledge based artificial intelligence control strategy module was developed and implemented to interface the voice components and support software of the Texas Instruments 955 workstation to the computer graphics functions implemented on the Sun Microsystems 4/280 color graphics workstation. Based on converted renditions of verbal utterances by the user, the control strategy determines the user's intent and develops and validates a plan. The program type and parameters within the plan are used as input to the graphics system for reconstructing and displaying medical imaging data corresponding to the perceived intent. If the plan is not valid, the control strategy queries the user for additional information. The control strategy operates in a conversation mode and vocally provides system status reports. The design of the knowledge based control strategy is presented with emphasis on the artificial intelligence techniques and the compositional heuristics incorporated in the strategy.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION	1
II. SIGNIFICANCE	3
III. METHOD	7
3.1. AI Module Overview	8
3.2 Initialization Submodule	12
3.3 Utterance Loader Submodule	12
3.4 Translator Submodule	13
3.5 Associator Submodule	14
3.6 AI Inference Submodule	14
3.7 Graphics Driver Submodule	15
3.8 Responder Submodule	15
IV. RESULTS	16
V. CONCLUSIONS AND RECOMMENDATIONS	23
LIST OF REFERENCES	25
APPENDICES	28
APPENDIX A. PAPER DESCRIBING VOICE ACTUATED COMPUTER	29
APPENDIX B. PAPER DESCRIBING AI MODULE	40
APPENDIX C. AI MODULE SOFTWARE	49
VITA	126

LIST OF FIGURES

FIGURE PAGE

1. AI module block diagram 9

2. Data words 10

3. Multiple CT slices 19

4. MRI wireframe display of human head 19

5. MRI reconstructed human head 21

CHAPTER I

INTRODUCTION

Research was conducted at The University of Tennessee Space Institute and Vanderbilt University Medical Center to develop a voice actuated computer graphics system that responds interactively with the user (Appendices A and B). In conjunction with this development, a knowledge based artificial intelligence control strategy module (AI module) was created to interface the various hardware components of the voice recognition and synthesis system (Speech module) to a collection of graphics functions (Graphics module) residing on the Sun-4 system. The AI module, consisting of eight major submodules, accepts textual renditions (in ASCII strings) of the user's utterance from the Speech module and determines the intent of the user. Based on the intent, the strategy constructes and validates a plan and instructs the Graphics module to display the corresponding image. If the plan is not valid, the AI module queries the user for additional information. The AI module enables the system to accommodate the user's natural speech and eliminates the need for the user to remember complex graphics entries, commands, codes, and application syntaxes.

Although the hardware aspects of the system are of significance, the major emphasis of this thesis is to present

the AI module developed for the voice actuated computer graphics system. A detailed description of the hardware aspects is presented in Appendix A. Chapter II presents the significance of the artificial intelligence control strategy developed for this research. Chapter III presents a discussion of the method used for implementing the knowledge based artificial intelligence control strategy module. The chapter presents an Overview and the Initialization, Utterance Loader, Translator, Associator, AI Inference, Graphics Driver, and Responder Submodules that comprise the AI module. Chapter IV presents a discussion of the results of this project. Chapter V presents the conclusions and recommendations for future research in the project's area.

CHAPTER II

SIGNIFICANCE

For decades, speech has been recognized by the scientific community as the preferred man-machine computer interface modality that would provide a natural means of accessing and controlling the computer and its control devices (1). With the early developments of voice recognition systems, it became possible for humans to "talk" information directly into the computer with no intermediate keying or other unnatural manual steps involved. With the continuing advancements in the field of speech recognition, speech applications have begun to appear in areas of industrial production and management information services, office automation, aids for the disabled persons, medical and surgical applications, aviation and space applications, voice I/O and information services, and many others (2). In addition, several commercially available speech recognition and synthesis boards, utilizing speech technologies as the computer interface modality, have begun to appear on the market (3,4).

However, as noted by Phillips, Bristow, et. al., within the area of man-machine communications by voice, there are three avenues of research: speech synthesis (voice output), speaker recognition (speaker identification and verification), and speech recognition (voice input). Although the speech

synthesis and speaker recognition modalities are relatively well developed, speech recognition technology is still in its infancy. The recognition task is complicated by the fact that speech signal characteristics differ from person to person and in the speech of an individual on different occasions. Speech recognition is also divided into four smaller categories, based on whether the systems are speaker dependent or independent, and able to recognize isolated word or continuous speech. Continuous speech recognition systems provide a more flexible communication modality for sophisticated applications; systems without continuous speech require a pause between each spoken word. Many commercially available speech recognition systems are speaker dependent continuous systems (3).

However, there is a distinct difference between continuous speech recognition and continuous speech understanding systems. Continuous speech recognition requires an exact verbatim transcription of the speaker's utterance, whereas, continuous speech understanding requires only that enough words in the utterance be correctly interpreted for an appropriate response to be elicited from the computer. The continuous speech recognition systems discussed above base their information retrieval on matching certain words uttered by the user to templates (keywords) stored within the systems. However, template matching provides very little syntactic coverage, no semantic understanding, no reasoning ability, no

easy way to get information into the system, and no way for the user to find out if the information returned is what was requested (5). On the other hand, as stated by Schutzer and et. al. (6,7,8), continuous speech understanding requires the implementation of a voice recognition element and a natural language interface that understands English and can interpret the conceptual and contextual elements of English input. For this study, the voice recognition element is the Speech module (Appendix A) and the natural language interface is the AI module. There are several commercially available interfaces, e.g., Intellect, LanguageCraft, NLI Datatalker, et. al. (6,9), that can also be used for natural language interfaces. However, because of the grammatical parsing algorithms they use to resolve valid input (identify, nouns, verbs, adjectives, and etc.) into sentences, these interfaces have difficulty of resolving unusual syntactical sentence structures and sentence fragments. Therefore, these interfaces are not well suited for voice recognition systems (10,11). In contrast, the AI module developed for this study does not rely on grammatical parsing, but uses heuristics (rules of thumb) and knowledge based artificial intelligence inference techniques to resolve valid input. This approach is more similar to the plan-based reasoning systems that are founded on Shank and Riesber's conceptual dependency semantic-network based representation (12). The AI module accommodates unusual sentence syntax structures and fragments that are difficult

for currently available interfaces. It can also accommodate unknown words and the absence of user's trivial words (words not essential to understanding the sentence's meaning) that are not recognized by the Speech module.

CHAPTER III

METHOD

The following chapter presents the underlying principles used to develop the AI module for interfacing the Speech module's voice recognition and synthesis functions implemented on the Texas Instruments 955 workstation to the Graphics module's computer graphics functions implemented on the Sun Microsystems 4/280 color graphics workstation. Referring to Figure 2, in Appendix B, the module is composed of seven submodules; the Initialization, Translator, Associator, AI Inference, Graphics Driver, and Responder. The module was written in Sun Common Lisp 3.0, a functional programming language especially suited for large complex programs, and executed on the Sun workstation in the UNIX environment. The language allowed for UNIX shell command calls, manipulating structures as objects, and foreign function calling. To facilitate the presentation of the AI module, the following chapter will consist of eight sections: (3.1) Overview, (3.2) Initialization Submodule, (3.3) Utterance Loader Submodule, (3.4) Translator Submodule, (3.5) Associator Submodule, (3.6) AI Inference Submodule, (3.7) Graphics Driver Submodule, and (3.8) Responder Submodule.

3.1 AI Module Overview

Referring to Figure 1, the AI module receives an ASCII character string from the Speech module or keyboard. The string is comprised of words, which fall into one of three categories: data, filler, or action words. Referring to Figure 2A, data words (Section 3.5) will consist of words and numbers used to select parameters, e.g., individual axial, sagittal, and coronal cross sections (320x320x18) from multiple Computed Tomography (CT) or Magnetic Resonance Imaging (MRI) cross sections (256x256x61) acquired from Vanderbilt University Medical Center (Appendix A). Referring to Figure 2B, words will also be used to select parameters, e.g., image perspectives, angular degrees of rotation in three dimensions, threshold levels, pseudocolors, and other parameters essential for displaying the image. All selected parameters must lie within the maximum and minimum domain defined for that parameter. Referring to Figure 2C, data words are used to select appropriate graphics programs for displaying graphics images. Filler words, e.g., and, the, to, again, and etc., are "helper" words that are normally used by individuals to verbally express their thoughts in a more natural and comprehensible manner. They bind action and data thoughts into complete sentences (action, filler, and data words) for natural language expressions. For this research, filler words did not provide pertinent information and were ignored.

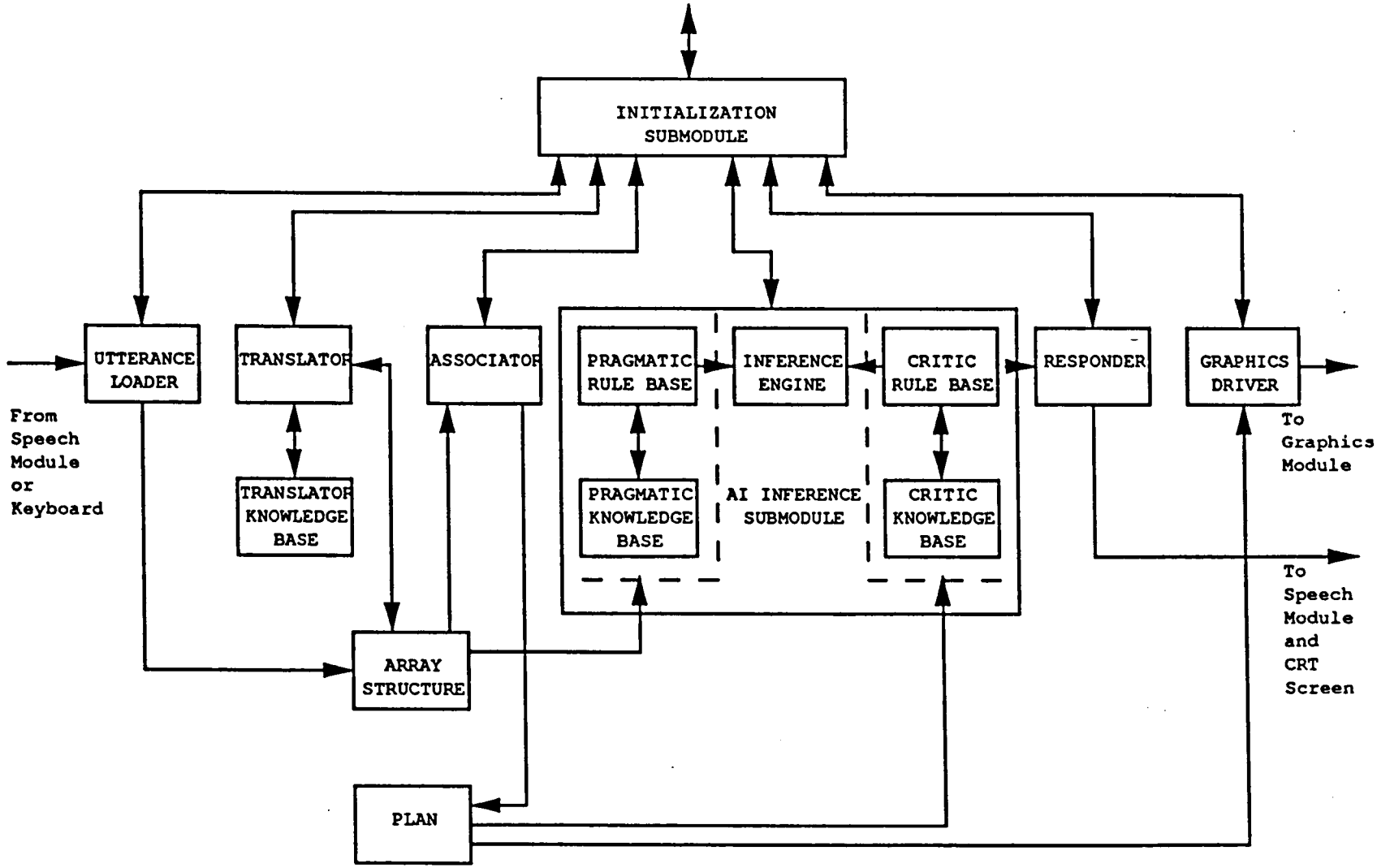

Figure 1. AI module block diagram.

PARAMETERS WORDS

Parameters	Function
CT	- selects the appropriate 320x320 CT cross section from the 18 available cross sections in the axial plane
sagittal	- selects the appropriate 256x256 MRI cross section from the 61 available cross sections in the sagittal plane
axial	- selects the appropriate 256x61 MRI cross section from the 256 available cross sections in the axial plane
coronal	- selects the appropriate 256x61 MRI cross section from the 256 available cross sections in the coronal plane

A. Data words (parameters) used for selecting CT or MRI cross sections for the graphics programs.

PARAMETERS WORDS

Parameters	Function
maximum	- selects the upper threshold value (CT or MRI number)
minimum	- selects the lower threshold value (CT or MRI number)
colorscale	- selects the type of color map used (grayscale, pseudocolor, or etc.)
background	- selects the background color to be used (black, gray, beige, or etc.)
shade	- selects the type of shading (phong, wireframe, gouraud, or etc.)
distance	- distance selected for determining the perspective (size) of the displayed image
theta	- selects the degree of longitudinal rotation
phi	- selects the degree of latitudinal rotation

B. Data words (parameters) essential for displaying the images of the graphics programs.

Figure 2. Data words.

APPROPRIATE GRAPHICS PROGRAMS

Program Name Words	Programs	Function
CT	CT_slice	- draws a CT axial slice full screen
sagittal	CT_slices	- draws a series of 18 axial slices
axial	sagittal_slice	- draws a MRI sagittal slice full screen
coronal	sagittal_slices	- draws a series of 18 sagittal slices
slice	axial_slice	- draws a MRI axial slice full screen
slices	axial_slices	- draws a series of 18 axial slices
all	coronal_slice	- draws a MRI coronal slice full screen
contours	coronal_slices	- draws a series of 18 coronal slices
face	all	- draws a MRI sagittal, axial, and coronal slice simultaneously
head	contours	- draws a three dimensional head using slice edges
normal	face	- draws a reconstructed three dimensional face without using smoothing techniques
	head	- draws a reconstructed three dimensional head without using smoothing techniques
	normal_face	- draws a smoothed three dimensional face
	normal_head	- draws a smoothed three dimensional head

C. Data words (program name words) used to select appropriate graphics programs , and the graphic programs used for the research project.

Figure 2. (Cont.)

Action words (Section 3.6), e.g., draw, rotate, cut, and etc., are words used to select appropriate graphics commands using the data words discussed above. When an ASCII character string is received, based on the intent of the action words, the AI module develops a plan, and if indicated, selects the appropriate graphics program and parameters for displaying an image. When the image is completed, the AI module informs the Speech module, and if indicated, the process is repeated.

3.2 Initialization Submodule

Referring to the block diagram in Figure 1, and the source program in Appendix C, the Initialization Submodule is responsible for loading the user help files, utility files, and foreign files. It initializes the various parameters, and loads the knowledge and rule bases, and other pertinent information necessary for program execution. It builds the various pragmatic and critic rule bases, and the translator, pragmatic, and critic knowledge bases for the Translator (Section 3.4) and AI Inference (Section 3.6) Submodules discussed below.

3.3 Utterance Loader Submodule

Again, referring to the block diagram and source program, the Utterance Loader uses Lisp (13,14), Array Reference Calls (aref call), for defining, accessing, and clearing an array structure. The submodule receives utterances (in an ASCII character string) from the Speech module or keyboard, grammar-

character string) from the Speech module or keyboard, grammarlessly parses (8) the ASCII words within the string, and stores the words at position index locations within the defined array structure. The procedure allows the Translator, Associator, and the AI Inference Submodules to effectively and efficiently access and manipulate each word in the manner discussed below. After the words have been processed and the user's intent determined and executed (Sections 3.4-3.8), if indicated, a new ASCII character string is acquired and the process is repeated.

3.4 Translator Submodule

Referring to the diagram and source program, the Translator Submodule consists of a knowledge base that contains action and data words (Section 3.1) and their internal language equivalents (Appendices B and C). The submodule initializes the position index (Section 3.3) to the first location of the defined array structure and then performs idiom recognition and canonical form techniques (15) on the words addressed by the incremented position index within the array structure. Using these techniques, the Translator reads the array structure one word at a time and each word is compared to the words within the knowledge base. If the word is within the knowledge base, the defined internal language equivalent of that word is substituted into the array in place of the original word. If the word is not found, the original word is substituted back into the array. For numbers such as

"one" or "three", their base 10 BCD (binary coded decimal) equivalent are substituted back into the array.

3.5 Associator Submodule

Referring to the block diagram and program, the Associator submodule uses heuristic convergence, proximity, and ghost rules that groups the data words, and based on the word groupings, develops a plan that selects the appropriate graphics programs and associated parameters (Section 3.1). For example, for equivalent medical domain words and numbers within the array structure, e.g., "slice", "axial", 1, 4, and 5, are grouped by the Associator to form a program name "axial_slice" and number 145, respectively. This plan is later used by the AI Inference Submodule to determine if the user's uttered words lie within the maximum and minimum domain defined during program initialization.

3.6 AI Inference Submodule

The AI Inference submodule is composed of a pragmatic rule base, a critic rule base, and an inference engine. The pragmatic rule base with its knowledge base contains if-then rules and support definitions. When executed by the inference engine, it determines the user's intent. The rule base checks the array structure in Section 3.4, for specific action words within the array and provide particular instructions to be executed. The critic rule base with its knowledge base also contains if-then rules and support definitions. When executed

by the inference engine, it determines if the plan developed in Section 3.5, is valid. The rule base checks the plan to determine if the associated parameters lie within the maximum and minimum domain defined for them. If the plan is not valid, an error message will be sent to the Responder Submodule (Section 3.8) discussed below. The inference engine uses forward chaining techniques (8) to process both the pragmatic and critic rule bases.

3.7 Graphics Driver Submodule

The Graphics Driver is responsible for transferring the valid plan to the Graphics module. The Graphics Driver interrupts the Graphics module and causes it to read and execute the plan to generate and display an image (Appendix A).

3.8 Responder Submodule

The Responder Submodule is responsible for sending error (Section 3.6) and image completion (Section 3.1) messages to the Speech module and to the TI and Sun-4 computer displays (Appendix B).

CHAPTER IV

RESULTS

Chapter IV, presents the results generated by the methodology discussed in Chapter III. These results were generated by participating in a verbal dialogue with the voice actuated computer graphics imaging system (Appendices A and B). To present the dialogue used to generate these results, the following chapter will present verbal sessions (Table 1) used to displaythe various graphics images (Figures 3-5) illustrated in this chapter. Additional dialogue and images are presented in the "System Performance" section of the referenced Appendices.

To demonstrate the AI module's semantic analysis of a verbal dialogue, Session 1, in Table 1, illustrates the verbal dialogue used to generate the image in Figure 3. The dialogue consisted of: the action word "Draw", filler words "with" and "a", and the data words "CT","slices", "black", and "background" (Section 3.1). The word "end" was used to indicate to the Speech module (Appendix A) the end of the spoken sentence. For the dialogue, the AI module loaded the sentence (ASCII character string) into the array structure (Section 3.3) as words, converted each word into its defined internal language equivalent (Section 3.4), grouped the data words, and developed a plan (Section 3.5). It checked the action word and

Table 1. Sessions with verbal dialogue.

Session 1, for Figure 3
USER: "Draw CT slices with a black background end"
SPEECH MODULE: "CT SLICES IMAGE COMPLETED"
Session 2, for Figure 3
USER: "With background black, draw CT slices end"
SPEECH MODULE: "CT SLICES IMAGE COMPLETED"
Session 3, for Figure 3
USER: "CT slices draw, background black end"
SPEECH MODULE: "CT SLICES IMAGE COMPLETED"
Session 4, for Figure 4
USER: "With a gray background, draw a wireframe head end "
SPEECH MODULE: "HEAD IMAGE COMPLETED"
Session 5, for Figure 5
USER: "Use theta one three five and phi three zero; draw head with cuts right four zero and top one nine five end"
SPEECH MODULE: "HEAD IMAGE COMPLETED"

determined the user's intent (Section 3.6), checked the data words within the plan and determined if the plan was valid (Section 3.6), and transferred the valid plan to Graphics module (Section 3.7) to graphically display an image on the Sun-4 color monitor. When the image was completed, the Speech module verbally responded with the message illustrated in Table 1, provided by AI module (Section 3.8). Figure 3, illustrates the image generated by the verbal dialogue. The image shows a series of eighteen successive two dimensional CT slices of a human head (from the top to bottom) in the axial plane. The slices, displayed in pseudocolors (warm body temperature color scale (16)), were drawn on a black background as specified by the dialogue. With additional dialogue, individual slices can also be drawn with various backgrounds, in grayscale, and full screen. For three dimensional images with cutting planes, psuedocolors were assigned by system defaults.

Sessions 2, and 3, demonstrated the ability of the AI module to accommodate unusual word order (sentence syntax). Although the words in Session 1, were spoken in a different order, the AI module still identified, generated, and displayed the same image as in Figure 3. These results demonstrated that regardless of word order, the module still could properly determine the intent of the dialogue demonstrated in Session 1.

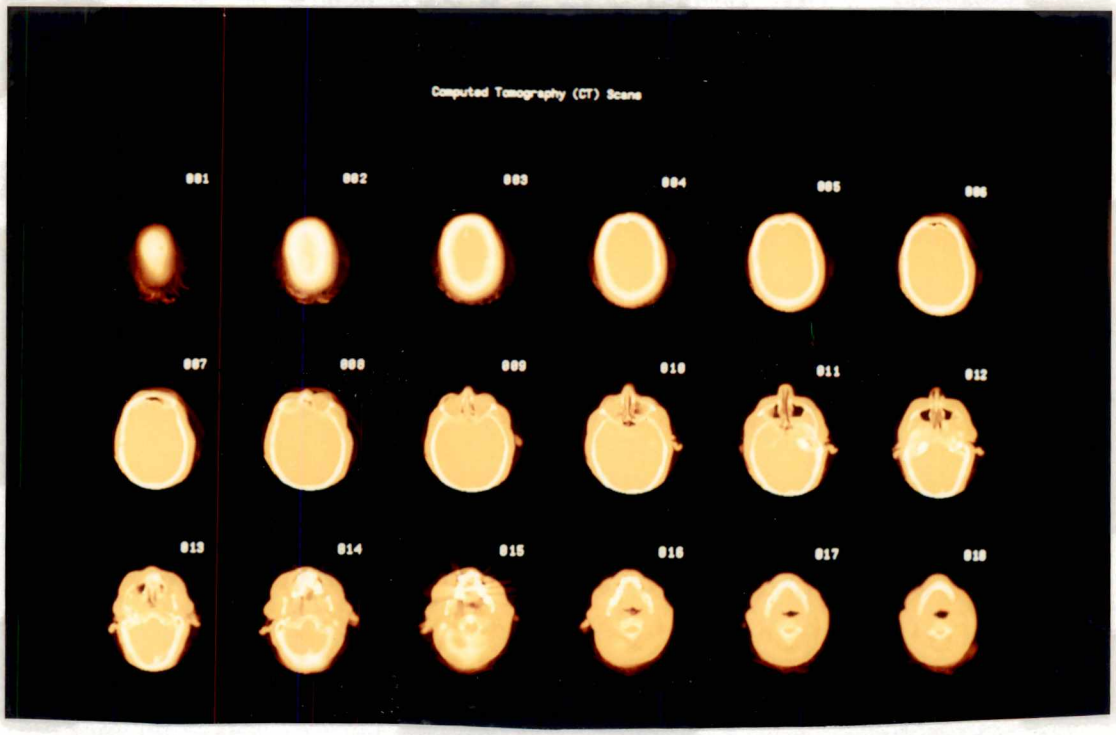

Figure 3. Multiple CT slices.

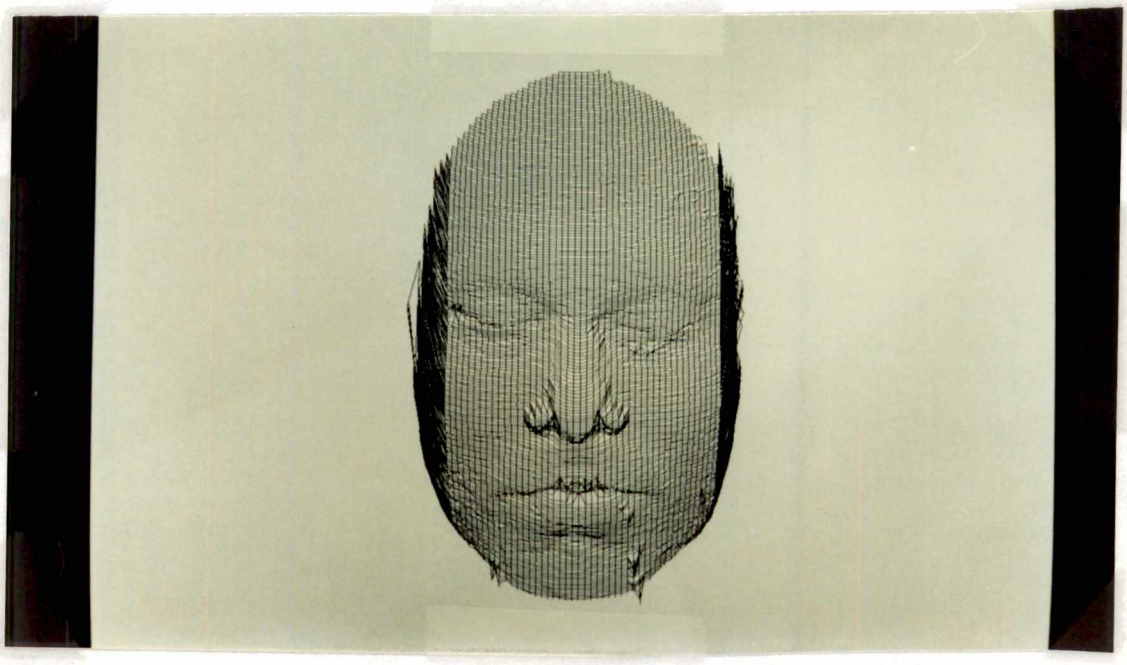

Figure 4. MRI wireframe human head.

The verbal dialogue in Session 4, was used to generate and display a three dimensional, reconstructed head of the human anatomy, as shown in Figure 4. The figure illustrates a wireframe image with a gray background. The wireframe image also provided discernable facial features of the scanned cadaver. The session also showed that the message sent depended on the type of image displayed (Section 3.8). For this particular session, the Speech module responded "Head Image Completed."

Session 5, illustrates a complex dialogue incorporating the use of three dimensional rotations and cutting planes. The rotation allows images to be rotated longitudinally (theta degrees) and latitudinally (phi degrees) in both clock and counter clockwise directions. The cutting planes allow images to have sections removed in the sagittal (from right to left), axial (from top to bottom), and coronal (front to back) planes. These planes (slices) allow external surface and internal anatomy to be simultaneously viewed from a single three dimensionally rotated image. Figure 5, shows a three dimensional reconstructed head, rotated longitudinally, one hundred and thirty five degress, and latitudinally, thirty degrees. These rotations were generated by the words "theta one three five" and "phi three zero" within the verbal dialogue. The figure also shows twenty one of the sixty one sagittal slices and sixty one of the two hundred fifty six axial slices removed by the cutting planes. These removals



Figure 5. MRI reconstructed human head.

were generated by the words "left two one" and "top six one" within the verbal dialogue. The AI module displayed the image using psuedocolor by default since no color was specified in the dialogue.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

The goal of this research is to allow experienced and inexperienced users to converse with the computer in a completely natural and unrestricted fashion. As part of this research, a voice actuated computer graphics imaging system was developed to respond interactively with the user. In conjunction with this development, an AI module was developed to interface the various hardware and software of the Speech module to the Graphics module residing on the Sun-4 system. The AI module received and loaded sentences into an array structure as words, converted words into their defined internal language equivalent, grouped data words, and developed a plan. It checked action words to determine intent, checked the plan to determine if the plan was valid, and transferred the valid plan to the Graphics module to graphically display an image. When the image was completed, the Speech module verbally responded with a message provided by the AI module. The AI module provided a natural language interface that allowed the user to vocally instruct the imaging system to display CT or MRI imaging data acquired from Vanderbilt University Medical Center. This module performed semantic analysis and interpreted intent of verbal dialogue to display the two and three dimensional images. It also accommo-

dated complex dialogue incorporating three dimensional rotations and cutting planes. The images were displayed with various colors and backgrounds, and full screen. An area for further research is to expand present capabilities of the AI module to accommodate speech understanding in more than one domain, e.g., UNIX, graphics, medical, and others (domain switching). Another area for research is to develop a toolkit for updating the module's rule and knowledge bases. The toolkit would also simplify module modification for other domains.

LIST OF REFERENCES

LIST OF REFERENCES

1. Scruggs, J. L., "Voice Control of Complex Workstations," 2nd Annual Workshop on Space Operations Automation and Robotics (SOAR), Report No: NASA CP-3019, Houston, TX, 1988.

2. Philip, G., Young, E.S., "Man-Machine Interaction by Voice: Development in Speech Technology. Part A: General Applications, and Potential Applications in Libraries and Information Services," Journal of Information Science, vol. 13, no. 1, pp. 15-23, February 1987.

3. Bristow, G.ed., Electronic Speech Recognition Techniques, Technology, & Applications, McGraw-Hill Book Co., 1986.

4. Holmes, J.N., Speech Synthesis and Recognition, Van Nostrand Reinhold (UK) Co. Ltd., 1988.

5. VoiceScribe-100 Speech Applications Designer's Handbook, Release 2.10, Dragon Systems Inc., 1986.

6. Schutzer, D., Artificial Intelligence: An applications-oriented Approach, Van Nostrand Reinhold Co., Ltd., pp. 146-163, 1987.

7. Fischler, M. A., Intelligence: The Eye, the Brain, and the Computer, Addison Wesley Publishing Co., Menlo Park, CA, pp. 165-188, 1987.

8. Barr, A., The Handbook of Artificial Intelligence, Heuristech Press, Stanford, CA, pp. 225-361, 1981.

9. Rettig, M., "How to Choose Natural Language Software," AI Expert, vol. 4, no. 2, pp. 44-49, July 1988.

10. Winston, T. W., "Natural Language Query Parsing," AI Expert, vol. 4, no. 2, pp. 44-49, February 1989.

11. Bobrow, D. G., Kaplan, R. M., et. al., "Gus, a Frame-Driven Dialog System," Artificial Intelligence, vol. 8, pp. 155-173, 1977.

12. Allen, J., Natural Language Understanding, Benjamin/Cummings Publishing Company, Inc., Menlo Park, pg. 214, 1987.

13. Sun Common Lisp 3.0 Advanced User's Guide, Sun Microsystems, August, 1988.

14. Tanimoto, S. L., The Elements of Artificial Intelligence, Computer Science Press, Inc., Rockville, MA, 1987.

15. Winograd, T., Language as a Cognitive Process, Volume 1: Syntax, Addison-Wesley Publishing Company, Menlo Park, CA pg. 400, 1983.

16. Prather, C., "Computer Generation of Two-Dimensional Pseudocolor Images of Monte carlo Input/Output Data," Thesis, The University of Tennessee Space Institute, Tullahoma, TN, 1985.

APPENDICES

APPENDIX A

PAPER DESCRIBING VOICE ACTUATED COMPUTER

Application of a voice actuated computer system
for reconstructing and displaying magnetic resonance imaging data

John S. Meadow, Alfonso Pujol Jr., and Patrick L. Sherley

University of Tennessee Space Institute
Tullahoma, Tennessee 37388

ABSTRACT

With the integration of computer graphics to presently available computed tomography, magnetic resonance imaging, and positron emission tomography scan devices, processing and displaying three dimensional data arrays is rapidly becoming important. Present systems can model full color, complex two or three dimensional objects, and manipulate their perspective displayed appearance in real time. However, even with the many hardware and software tools available for providing enhanced versions of two or three dimensional information, the rapid developments within these areas continues to outpace the ability of users to learn and use them efficiently. At the University of Tennessee Space Institute, Department of Electrical Engineering, and Vanderbilt University Medical Center, Department of Radiological Sciences, research is being performed to develop a voice actuated computer graphics system that responds interactively with the human voice. The research uses a Texas Instruments voice recognition and speech synthesis board, a Texas Instruments 955 computer workstation, and a Sun Microsystems 4/280 color graphics workstation. Verbal commands are used to control the graphics system programs, set parameters for displaying two and three dimensional views, and instruct the system for additional cross-sections, viewing references, display formats, etc. Verbal responses from the computer inform the user when additional information is needed or error conditions have occurred. The system provides a means for users to conduct an interactive verbal dialogue with the computer for rendering and displaying complex medical imaging data.

1. INTRODUCTION

This paper describes the development of a voice actuated computer graphics system that responds interactively with the human voice. The research uses a TI speech hardware board and application tool kit, a TI 955 computer workstation, and a Sun Microsystems 4/280 color graphics workstation. A microphone and speaker are connected directly to the speech board and provide voice input and speech output for the TI workstation. A serial interface is used to transfer speech data between the TI and Sun workstations. An Esprit Executive 10/102 terminal is used as the Sun controller console allowing the color monitor to be used continually for full screen color graphics. Anatomic data was obtained from computed tomography (CT) and magnetic resonance imaging (MRI) scans of a cadaver performed on the imaging systems in the Department of Radiological Sciences, Vanderbilt University School of Medicine. The series of two dimensional scans represent a composite set of cross sectional arrays that describe the particular anatomical area of interest. The arrays are reconstructed to numerically describe anatomy as a three dimensional rectangular array of elements.

2. SIGNIFICANCE

In today's world, computers are used in almost every field of human endeavor and are being sold from department stores to Hi-Tech OEM's (original equipment manufacturers). However, many computer users soon become frustrated with the incomprehensible hardware and software environments and the bewildering number of commands to control them. Good typing skills and a knowledge of the keyboard are also essential for speedy and accurate input of data to the computer.[1] For decades, there has been increasing interest among the scientific, industrial and medical disciplines for a suitable man-machine computer interface that would alleviate the user burden of having to learn different computer methodologies, e.g., operating systems, data structures, languages, interfaces, and others. For years, speech has been recognized as one of the preferred man-machine computer interfaces.[2] This interface provides a more natural interface modality for accessing and controlling the computer and associated peripheral devices. Within the last decade, there have been significant advances in the field of speech recognition, and applications of this technology have begun to appear in areas of industrial production and management information services, office

automation, aids to disabled persons, medical and surgical applications, aviation and space applications, libraries and information services, and others.[3]

Within the radiological sciences, the integration of computer graphics to CT and MRI scan devices is rapidly becoming important. Present systems can model full color, complex, two or three dimensional objects, and manipulate their perspective displayed appearance in real-time. However, even with the many hardware and software tools available for providing enhanced versions of two or three dimensional information, the rapid developments within these areas continues to outpace the ability of users to learn and utilize them efficiently. This paper presents the research being performed in the development of a voice actuated computer graphics system that responds interactively with the human voice. The system will translate utterances into proper application syntaxes and eliminate the need for remembering complex CT or MRI graphics entries and commands. Although this project will not provide the ultimate goal in this technology, it will provide an attempt for establishing a basis for achieving this goal.

3. SYSTEM HARDWARE

Figure 1, illustrates a block diagram of the voice actuated computer system that was developed and used for this research. The figure shows a TI voice recognition and speech synthesis board, a TI 955 workstation and monitor, a Sun Microsystems 4/280 graphics workstation and monitor, and a Esprit Executive 10/102 terminal. A serial interface is used to transfer voice data between the TI and Sun workstations. The TI speech board was used because of TI's advanced hardware and software support utilities as discussed by Scruggs and TI.[2,4] The board is a single board option for IBM compatible personnel computers and offers a wide range of voice recognition and speech synthesis functions. The board provides speaker dependent/connected speech (continuous speech) recognition and allows the speaker to talk in a more natural manner. Many other commercially available systems are isolated word systems that require pauses between each spoken word.[5,6] The board contains a 32 bit TMS320C10 microprocessor, 32K bytes of dual port random access on-board memory, a CODEC analog/digital converter, and other hardware features to support the research. The software support utilities for the board provides text to speech and continuous speaker dependent language recognition software which converts ASCII text into smooth, intelligible voice output and enables the computer to recognize words based on the context in which they are used, respectively. The TI 955 is an Intel 80286 microprocessor based IBM PC/AT compatible workstation with a 50 Megabyte hard disk drive, 1 Megabyte of RAM, and a RS-232 serial port for communications. The workstation's RS-232 serial port is used to transfer voice data to and from the Sun workstation. A TI color graphics adapter (CGA) monitor was used with the TI workstation. The Sun workstation has a 19 inch color monitor with a screen resolution of 1152 x 900 pixels and was used for displaying CT or MRI scans. The Sun system includes 892 Megabytes of hard disk storage, 32 Megabytes of RAM, a Graphics Processor board, 16 serial ports, and a 1/2 magnetic tape which was used to load the medical image data onto the system. The Esprit terminal was used as the Sun controller console to allow the color monitor to be used continually for color graphics.

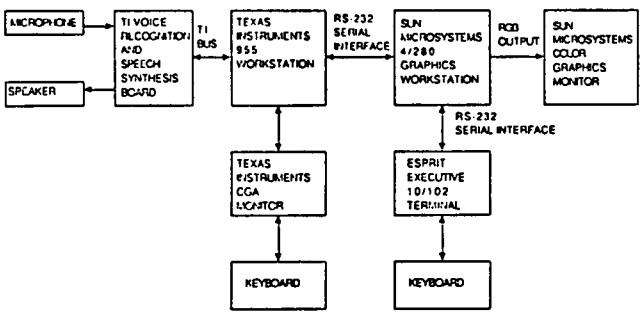

Figure 1. Block diagram of the voice actuated computer system.

4. SYSTEM SOFTWARE

Figure 2, illustrates the system software which consists of a speech module, an artificial intelligence (AI) module, and a graphics module. The speech module is executed by the TI workstation and the AI and graphics modules are executed by the Sun workstation. The speech module software was developed to control the voice recognition and speech synthesis functions of the voice actuated system. The AI module controls the dialogue for the voice actuated system and issues commands to the graphics module when indicated by the user's dialogue. An in-depth discussion of the AI module used in this research is available in a related paper authored by Patrick L. Sherley.[7] The graphics module consists of two and three dimensional graphics programs which outputs the desired images to the Sun color monitor when instructed by the AI module. To present the underlying principles of the software, the following sections present a detailed discussion of these modules.

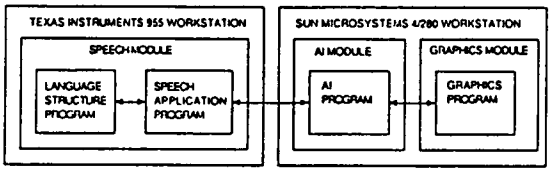

Figure 2. System software modules.

4.1 Speech module

Referring to Figure 2, the speech module consists of two programs: a language structure program and a speech application program. Both programs were developed on the TI workstation running under the MS-DOS operating system. The language structure program was developed using the TI LR2000 continuous language recognition software and allows the speech system to recognize words based on the context in which they are spoken. The program defines the specific word vocabulary as well as grammars (user defined word sets and language rules) that determines the order in which words can be spoken. Depending on the size of the word, the maximum number of words that can be stored in one grammar is 50. The TI speech system can support up to 100 grammars, however, only one grammar can be loaded on the speech board at any one time. The vocabulary words within the loaded grammar can be used to invoke a new grammar. New grammars may be loaded as different word sets are needed. The voice recognition process implemented by TI, uses a word hypothesizer which compares initially enrolled word templates with the real-time speech input. As the user speaks, the input speech is compared against the word templates and when a match is found, the hypothesizer outputs the result to the sentence recognizer. The sentence recognizer compares the output of the word hypothesizer with the defined language rules. The process is repeated for each spoken word until the sentence is completed. Each completed sentence is then output to the speech application program.

Referring to Figure 3, for this project, one grammar containing 43 words with language rules was defined within the language structure program. The language rules defined for this research allowed words to be spoken in any order and construct sentences using any possible word combinations. The 43 words were chosen to describe as general as possible, a vocabulary of words used within the medical imaging paradigm. The word "end" was used in the grammar to signal the sentence completion. The defined rules do not place sentence constraints on the user as previous methods did.[2] The LR2000 utility was used to compile, enroll, update, and test the language structures for this project.

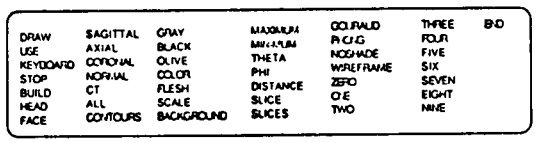

Figure 3. Vocabulary for language structure program.

Figure 4, presents a block diagram of the speech application program. The program was written in the C programming language and uses function calls to access the LR2000, run time routines and libraries. Using the serial interface, the program waits for a message from the AI module. When the message is received by the speech application program, the program outputs the message to the user through the TI speaker system (using the text to speech utility) and monitor, and then begins to receive and recognize voice data (words) through the microphone. When the voice data is completed (a sentence), the data is converted into an ASCII character string and sent to the AI module and printed on the TI monitor. The process is repeated until it is aborted by the TI keyboard or verbally instructed to stop by the system user.

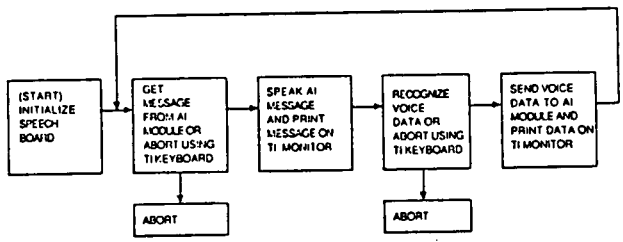

Figure 4. Block diagram of the speech application program.

4.2 AI module

The AI module developed by Sherley, was written in Lisp and executed on the Sun workstation in the UNIX environment. Using the serial interface, the AI module sends a message to the speech module and waits for the voice data character string. When the string is received, the AI module checks the string for intent, validates the intent, and formulates a response. If the intent is not valid, the program sends the appropriate message back to the speech module and waits for a new character string. If the string contains a valid graphics command, the AI module places the corresponding program number and associated parameters into a common file and sends an interrupt to the graphics module to read the file and display the selected image. Figure 5, shows an example of the conversion process performed by the AI module. After the image is displayed, the graphics module alerts the AI module that the image is completed and the process is repeated.

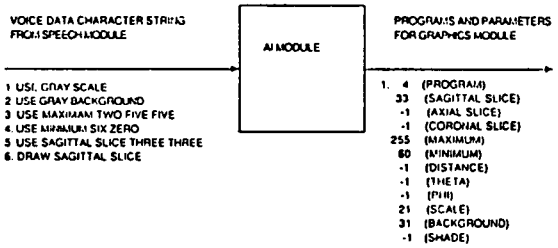

Figure 5. AI module conversion process.

4.3 Graphics module

The graphics module is written in the C programming language using SunCore interactive graphics primitives to perform the two and three dimensional graphics functions.[8] The program is executed on the Sun workstation in the UNIX environment. The graphics images are displayed in either the Sun window environment[9] or on the single display screen. Referring to Figure 6, the graphics module waits for an interrupt from the AI module and opens the common file to read the selected program and parameters mentioned above. Once the command data is verified, the graphics module executes the selected program to display the CT or MRI image. After the image is completed, the graphics module alerts the AI module that the program is completed and the process is repeated. Figure 7, illustrates the graphics module programs and parameters used for this research. The build program shown in the figure was used to create the reconstructed data files for the two and three dimensional programs. The two dimensional parameters select the slice numbers and the maximum and minimum CT or MRI numbers displayed. The three dimensional parameters select the three dimensional space (in spherical coordinates) and types of object shading used. Additional two and three dimensional parameters also select the colormap and background colors for the medical images.

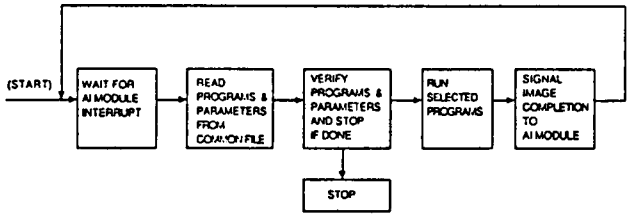

Figure 6. Block diagram of the graphics module.

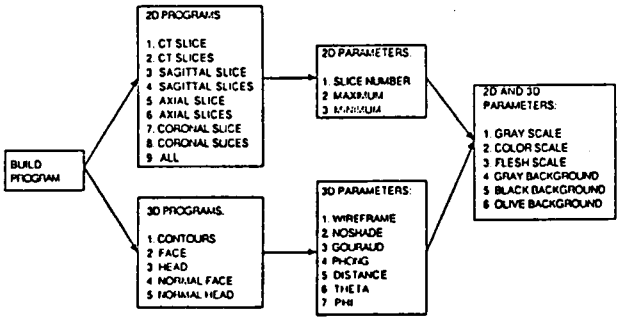

Figure 7. Graphics module programs and parameters.

5. SYSTEM PERFORMANCE

Figure 8, illustrates an example of a verbal dialogue between the user and the developed voice actuated computer system. The number preceding the sentences indicate the order in which the sentences were spoken by the user and responded to by the computer system. Figure 9, shows MRI sagittal slice 33, against a gray background, and a gray scale legend with labels, maximum 255, and minimum 60, displayed after the verbal dialogue was completed. Since the system contains initial default settings and also retains the last parameter settings, the maximum and minimum number values were not included in the dialogue. To demonstrate the error handling capability of the system, sentence number 1, used a non-existent sagittal slice number to create a system error. The system immediately informed the user that the command sentence was invalid by responding: "70 is outside the acceptable range for the sagittal slice plan." Sentence number 3, was then spoken with a valid sagittal slice number. Figure 10, illustrates simultaneous MRI axial, sagittal, and coronal views displayed with the sentence: "Draw all views using axial slice one one five and coronal slice one five zero end." The sagittal slice number was retained from the previous slice number as discussed. Figure 11, shows simultaneously displayed, MRI multiple coronal images, slice numbers 130-147, displayed with the sentence: "Draw coronal slices starting at one three zero end." Figure 12, shows CT slice 15, with a minimum CT number of 15, and a black background, created with the sentence: "Draw CT slice one five with a minimum value of one five and a black background end." Figure 13, shows a MRI three dimensional reconstructed, isocontour display of a human head, created with the dialogue: "Draw contours using a theta value of nine zero end." Figure 14, shows a MRI three dimensional reconstructed, gouraud shaded display of a human head with a gray background, created with the following sentence: "Draw normal head with gouraud shading, a theta value of one eight zero and a gray background end."

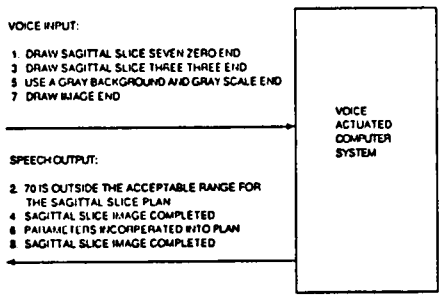

Figure 8. Example verbal dialogue.

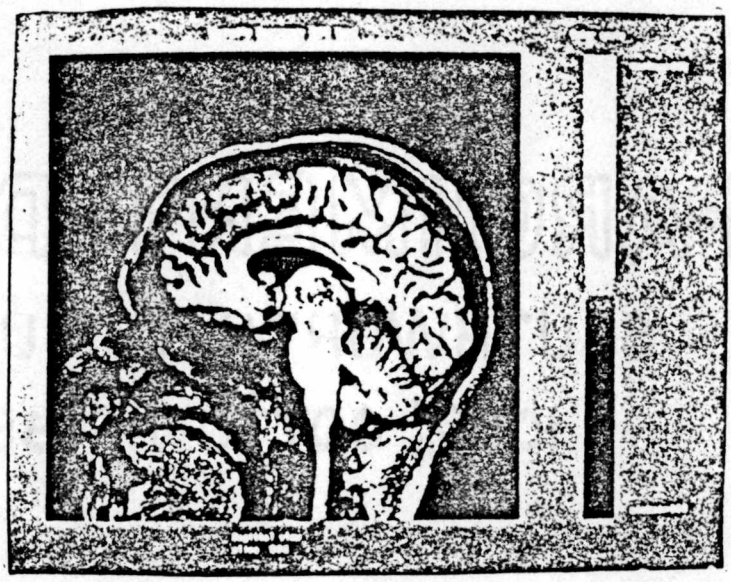

Figure 9. MRI sagittal slice 33.

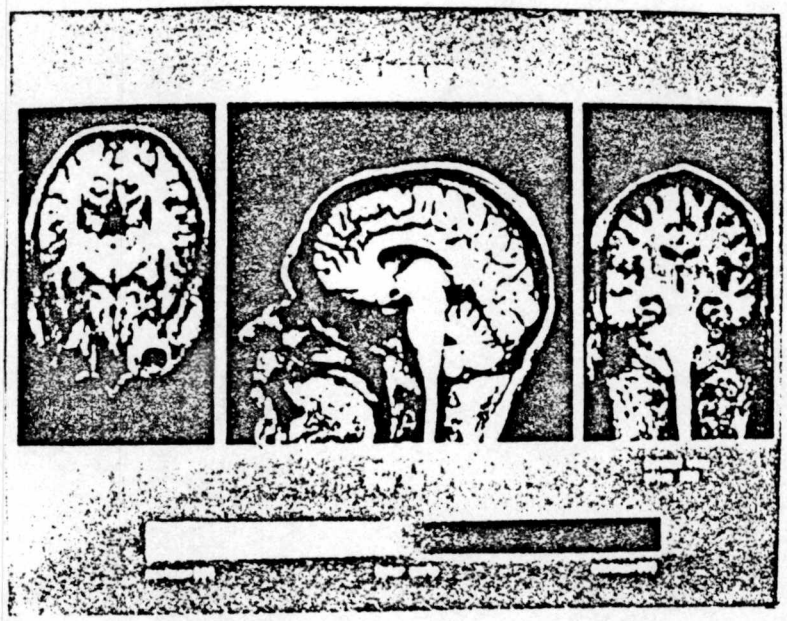

Figure 10. MRI axial, sagittal, and coronal views.

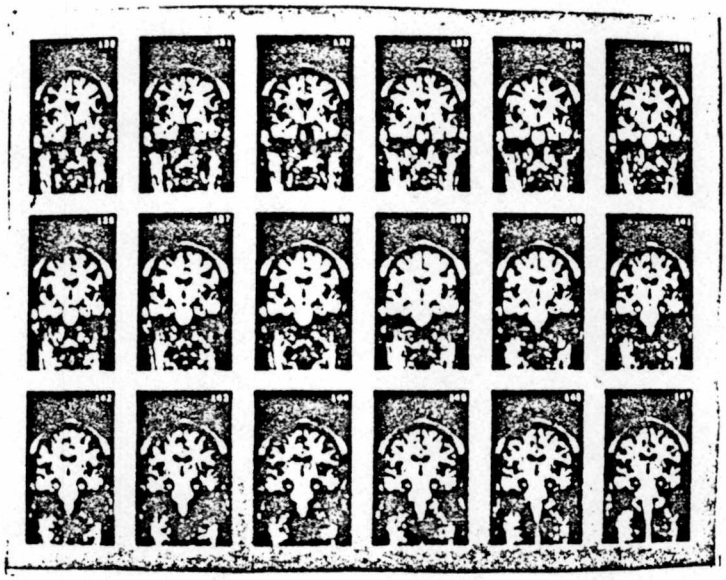

Figure 11. MRI multiple coronal slices.

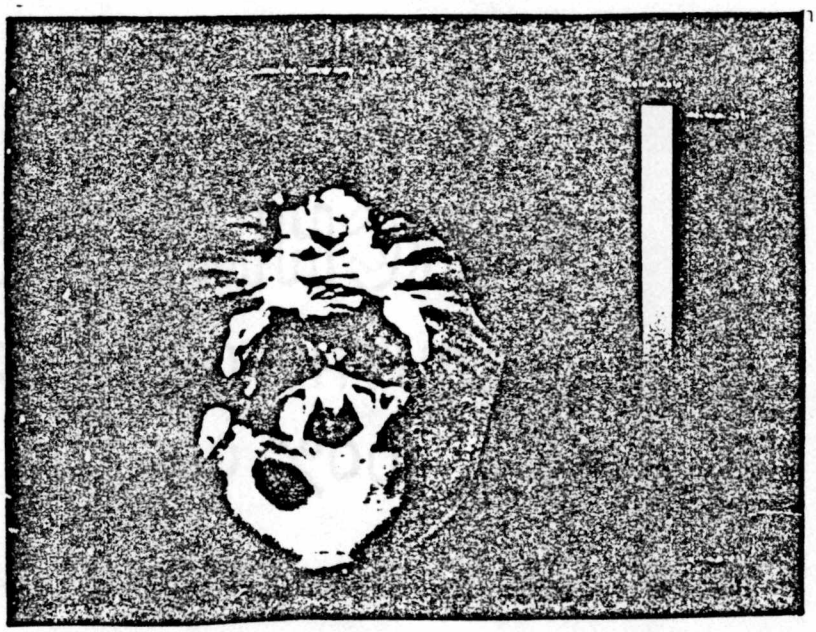

Figure 12. CT slice 15.

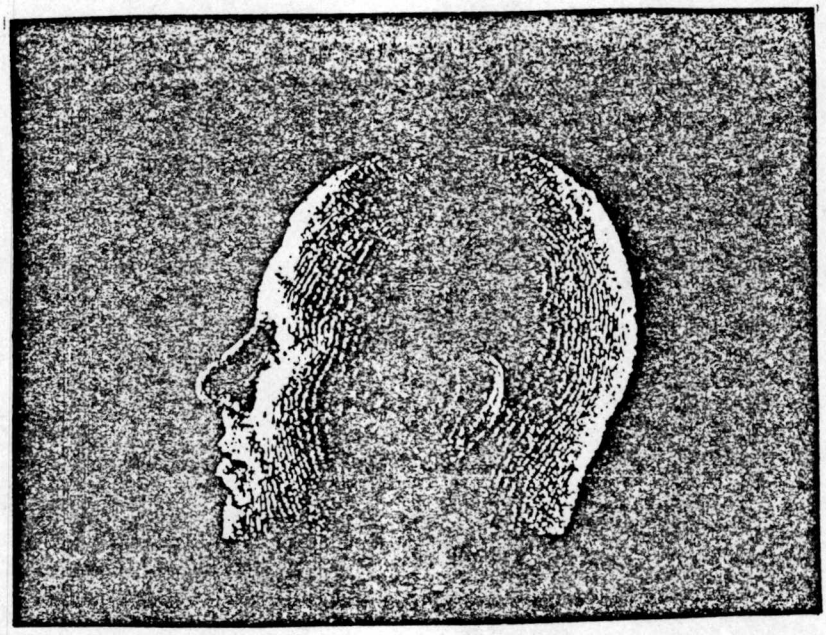

Figure 13. MRI isocontour display of human head.

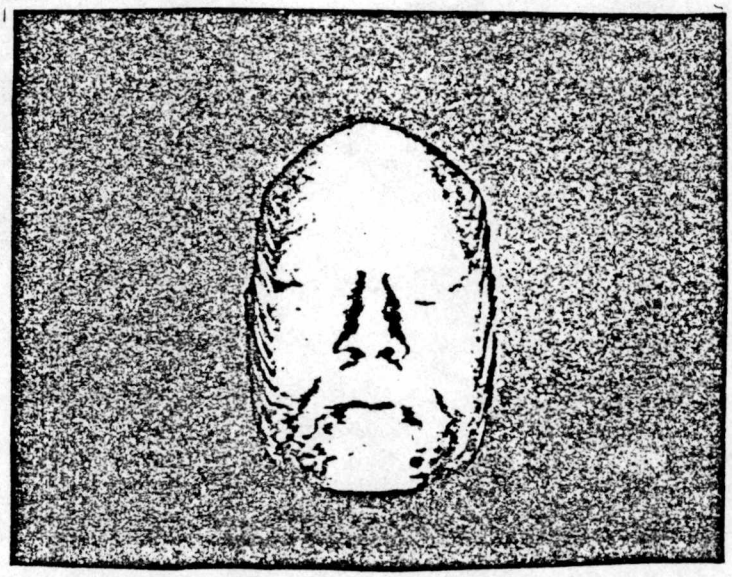

Figure 14. MRI reconstructed human head.

6. SUMMARY AND CONCLUSIONS

This paper described the use of a Texas Instruments voice recognition and speech synthesis board to provide a voice interface to a Sun Microsystems Sun-4/280, color graphics workstation for reconstructing and displaying CT and MRI images. The imaging data was acquired from Vanderbilt University Medical Center, Department of Radiological Sciences on 1/2 inch magnetic tape. Verbal commands spoken into the microphone controlled the graphics system programs and parameters for displaying the two and three dimensional views and instructing the system for additional cross-sections, view references, viewing formats, etc. Verbal responses from the speech synthesis board informed the user when additional information was needed or when error conditions occurred. The system completely controlled the graphics programs, and while no quantitative study was performed the system performed well under normal conditions. However, one problem encountered with the Texas Instruments voice recognition system was the limited number of words storable in one grammar. As previously mentioned, this project used one grammar with 43 words and allowed these words to be spoken in any order. The limited number of words greatly restricted the vocabulary size and word combinations that was used to describe the graphics systems programs and parameters. In addition, because the TI voice recognition system was speaker dependent, the vocabulary words had to be enrolled for each individual speaker who used the system. This process was time consuming and laborious. With the rapid advances in speech technology, an area for future study is to use the next generation of speech systems to repeat and expand the research endeavor. As these future advancements take place, the ultimate goal will be to allow experienced and inexperienced computer users to converse with the computer in a completely natural and unrestricted fashion.

7. ACKNOWLEDGEMENTS

The authors wish to acknowledge Texas Instruments, Inc., Martin Ras, and Robert E. Hollman, and Sun Microsystems, Inc., and Dr. Emil Sarpa, for their support in this endeavor.

8. REFERENCES

1. G. Philip and E.S. Young, "Man-Machine Interaction by Voice: Development in Speech Technology. Part I: The State-of-the-Art," Journal of Information Science, vol. 13, no. 1, pp. 3-13, February 1987.

2. J.L. Scruggs, "Voice Control of Complex Workstations," 2nd Annual Workshop on Space Operations Automation and Robotics (SOAR), Report No: NASA CP-3019, Houston, TX, 1988.

3. G. Philip and E.S. Young, "Man-Machine Interaction by Voice: Development in Speech Technology. Part II: General Applications, and Potential Applications in Libraries and Information Services," Journal of Information Science, vol. 13, no. 1, pp. 15-23, February 1987.

4. TI Speech Application Toolkit Guide, Part number 2232384-0001, Revision C, Texas Instruments, June 1989.

5. A.C. Staugaard, Jr., Robotics and AI: An introduction to applied machine intelligence, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1987.

6. G. Bristow ed., Electronic Speech Recognition Techniques, Technology, & Applications, McGraw-Hill Book Co., 1986.

7. P. L. Sherley, "Application of artificial intelligence techniques to a voice actuated computer system for reconstruction and display of magnetic resonance imaging data," presented at the SPIE-Society of Photo-Optical Instrumentation Engineers, Medical Imaging IV, Newport Beach, California, February 4-9, 1990.

8. SunCore Reference Manual, Part number 800-1787-10, Revision A, Sun Microsystems, May 1988.

9. Sunview1 Programmers Guide, Part number 800-1784-10, Revision A, Sun Microsystems, May 1988.

APPENDIX B

PAPER DESCRIBING AI MODULE

Application of AI techniques to a voice actuated computer system for reconstructing and displaying magnetic resonance imaging data

Patrick L. Sherley, Alfonso Pujol, Jr., John S. Meadow

The University of Tennessee Space Institute
Tullahoma, Tennessee 37388

ABSTRACT

To provide a means of rendering complex computer architectures, languages, and input/output modalities transparent to experienced and inexperienced users, research is being conducted to develop a voice driven/voice response computer graphics imaging system. The system will be used for reconstructing and displaying computed tomography and magnetic resonance imaging scan data. In conjunction with this study, an artificial intelligence (AI) control strategy was developed to interface the voice components and support software to the computer graphics functions implemented on the Sun Microsystems 4/280 color graphics workstation. Based on generated text and converted renditions of verbal utterances by the user, the AI control strategy determines the user's intent and develops and validates a plan. The program type and parameters within the plan are used as input to the graphics system for reconstructing and displaying medical image data corresponding to that perceived intent. If the plan is not valid, the control strategy queries the user for additional information. The control strategy operates in a conversation mode and vocally provides system status reports. A detailed examination of the various AI techniques is presented with major emphasis being placed on their specific roles within the total control strategy structure.

1. INTRODUCTION

Research is being conducted at The University of Tennessee Space Institute and Vanderbilt University Medical Center to develop a voice actuated computer graphics system that responds interactively with the user. In conjunction with this development, an artificial intelligence (AI) control strategy (AI module) was developed to interface the various components of the voice recognition and synthesis system (speech module) to a collection of graphics functions (graphics module) residing on the Sun-4 system. The AI control strategy, consisting of seven submodules, accepts textual renditions (in ASCII strings) of the user's utterance from the voice module and determines the intent of the user. Based on the intent, the strategy develops and validates a plan and instructs the graphics module to display the corresponding image. If the plan is not valid, the control strategy queries the user for additional information. The control strategy allows the system to translate voice input into proper application syntaxes and eliminate the need for remembering complex graphics entries, commands, codes, or syntaxes. Although the hardware aspects of the system are of significance, the major focus of this paper will be to present the AI techniques used to achieve the objectives of the research project.

2. SIGNIFICANCE

For decades, speech has been recognized by the scientific community as the preferred man-machine computer interface modality that would provide a natural means of accessing and controlling the computer and its control devices.[1] With the early developments of voice recognition systems, it became possible for humans to "talk" information directly into the computer with no intermediate keying or other manual steps involved. The numerical computations, storage, and manipulation of information became available to computer users with little or no computer knowledge.[2] With continuing advancements in the field of speech recognition, applications of speech technologies have begun to appear in areas of industrial production and management information services, office automation, aids to the disabled persons, medical and surgical applications, aviation and space applications, voice I/O and information services, and many others.[3] In addition, several commercially available speech recognition and synthesis boards, utilizing speech technologies as the interface to computers and PC's (personal computer), have begun to appear on the market.[4,5]

However, as mentioned by Phillips, Bristow, et al., within the area of man-machine communications by voice,

there are three avenues of research: speech synthesis (voice output), speaker recognition (speaker identification and verification), and speech recognition (voice input). The techniques in speech synthesis and speaker recognition are relatively well developed. Speech recognition technology, however, is still in its infancy. The recognition task is complicated by the fact that characteristics of speech signals differ from person to person, and even in the speech of one individual on different occasions. Therefore, it is more practical to divide speech recognition into four smaller categories, according to whether the systems are speaker dependent or independent, and able to recognize isolated words or continuous speech. Continuous speech (or connected speech) recognition systems provide a more flexible means for sophisticated applications. Systems without continuous speech require a pause between each spoken word. Many commercially available speech recognition systems are speaker dependent continuous systems.[4]

However, there is a distinct difference between continuous speech recognition and continuous speech understanding. Continuous speech recognition requires an exact verbatim transcription of the speaker's utterance, whereas, continuous speech understanding requires only that enough of the words in the utterance be correctly interpreted for an appropriate response to be elicited from the computer. The speech recognition systems discussed above are continuous speech systems that base their information retrieval on matching certain words uttered by the user to templates (keywords) stored within the systems. However, template matching provides very little syntactic coverage, no semantic understanding, no reasoning ability, no easy way to get information into the system, and no way for the user to find out if the information returned is what was requested.[6] On the other hand, as stated by Schutzer and et al., continuous speech understanding requires the implementation of a voice recognition element and a natural language interface that understands English and can interpret the conceptual and contextual elements of English input .[7,8,9] The voice element used for this study is a TI speech recognition board with support utilities and the natural language interface is an in-house developed AI control strategy module. There are several commercially available natural language interfaces, e.g., Intellect, LanguageCraft, NLI Datatalker, et al., that use grammatical structure parsing for database query type applications[7,10] and resolve valid input (identify nouns, verbs, adjectives, etc.) into sentences. However, they have difficulty resolving unusual syntactical sentence structures and sentence fragments and are not suited as an interface to a voice recognition system.[11,12] Consequently, natural language interfaces are not presently structured for voice recognition systems but are restricted to the keyboard for input.[13,14] The developed AI control strategy module for this study does not use grammatical structure parsing, but uses knowledge based inferencing to resolve valid input. The technique allows the interface to resolve unusual syntactical sentence structures and sentence fragments that present commercially available interfaces cannot resolve. It allows English input to be very informal and gives the user the freedom to converse in a more natural manner. To eliminate dialogue redundancy, the technique allows the user to perform ellipsis (omit information from present dialogue that can be inferred from the previous dialogue). The technique can also accommodate unknown words and the absence of user's trivial words (words not essential to the sentence's meaning) that are not recognized by the voice recognition system. Furthermore, the technique does not require a keyboard for input. To present the underlying principles of this proposed study, the following chapters will describe various procedures used to achieve the project's objectives.

3. SYSTEM HARDWARE

The system hardware used in this project consists of a Texas Instruments voice recognition and speech synthesis board and application tool kit, a Texas Instruments 955 computer workstation, and a Sun Microsystems 4/280 color graphics workstation. A microphone and speaker are connected directly to the speech board and provide voice input and speech output for the TI workstation. A serial interface is used to transfer data between the workstations, as illustrated in Figure 1. An in-depth discussion of the system hardware and speech and graphics module software used in this study is available in a related paper authored by John S. Meadow.[15]

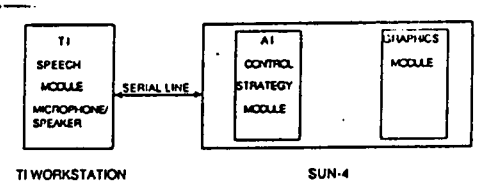

Figure 1. System block diagram.

4. AI CONTROL STRATEGY MODULE

Referring to Figure 2, the following sections present a detailed discussion of each of the developed software submodules that comprises the AI control strategy module. The module was written in Sun Common Lisp 3.0, a functional programming language especially suited for large complex programs .[16] The language allows for Unix shell command calls, manipulating structures as objects, and foreign function calling.

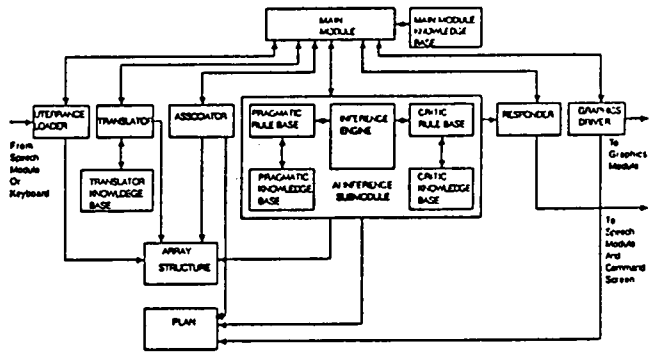

Figure 2. AI control strategy module.

4.1 Main submodule

Referring to Figure 2, the Main submodule is the main program that performs the overall function calls to the subordinate submodules to be discussed. Associated with this submodule is a knowledge base that contains global initialization data. This submodule is responsible for initializing and terminating the graphics module and itself.[15] It also implicitly selects inferencing modalities (pragmatic or critic mode) for the inference engine, and coordinates and synchronizes the activities of the other submodules.

4.2 Utterance loader

The Utterance loader is responsible for receiving utterances (in ASCII character strings) from the speech module developed by Meadow, or from the keyboard. Once the string is received, the submodule selects words from the

continuous string and loads the words (in ASCII) into an array structure. This procedure simplifies word access by other submodules.

4.3 Translator submodule

The Translator submodule is used to perform idiom recognition and canonical form conversions on the ASCII words within the array structure defined above. The Translator's associated knowledge base has an internal language dictionary that contains words which are relevant (within the paradigm) to this study. Figure 3, shows the internal language dictionary with words and synonyms, and their defined equivalent. The Translator scans the array structure, and each word within the array is compared to words (and their synonyms) within the internal language dictionary. If a match (or synonym) is found, the defined equivalent of that word is substituted back into the array. If a match is not found, the word is substituted back. If the word represents a numerical value, its defined equivalent is converted to a base 10, BCD (binary coded decimal) numerical value and substituted back into the array.

INTERNAL LANGUAGE DICTIONARY

VOCABULARY LIST

Word (Synonym)	Defined Equivalent
draw drawn display routine routines see	plan
sag sagittal side	sag
cor coronal horizontal	cor
max maximum top upper	max
bottom lower min minimum	min
dist distance s	dist
theta y yaw	yaw
phi pitch slant tilt z	pitch
colors cscale scale	cscale
background screen	background
shade shaded shading	shade
assist assistance explain help know list options	help
rotate rotated rotating rotation spin spun turn turned turning	rotate
home front	home
advance forward increment next	next
backward decrement last previous	last
select selected selecting use used using with	use
face	face
head cranium	head
wire	51
without	52
gour gouraud	53
phong phong-diffuse	54
gray grey	21
spectrum red reddish color	22
flesh natural	23
black dark	32
olive	33
zero	0
one	1
two	2
.	.
.	.
eighteen	18
nineteen	19
twenty	20
thirty	30
.	.
.	.

Figure 3. Internal language dictionary.

4.4 Associator submodule

The Associator submodule uses developed compositional (heuristic) rules (refer to Figure 4) to determine how numbers or words are grouped within the above redefined array structure. For example, the numerical values "1 2 5," "1 20 5" or "100 20 5" are grouped "125" and the words "sagittal slice" or "slice sagittal" are grouped "sagittal-slice." In addition, the Associator submodule also develops a plan that consists of a program number and parameters to be used by the graphics module. Figure 5, illustrates the program numbers and parameters used to display selected two and three dimensional images.[15]

1. Proximity Rule:

(x)(y){[{[(Ax)(Vy)(Piyx)]=(Lyx)}v
{(~[(Ax)(Vy)(Piyx)][(Ax)(Vy)(Fiyx)]=(Lyx)}v
{{[{~[(Ax)(Vy)(Piyx)]}]}v(~[(Ax)(Vy)(Fiyx)])]
[(Ax)(Vy)(Fyx)]}=(Lyx)}]

2. Ghost Rule:

a. (x){(Ax)[~({(y)[L(Vy)x]})]=(Rvx)}

b. (y){(Vy)[~({(x)[Ly(Ax)]})]=(Lyal)}

3. Convergence Rule:

(x)(y){[(Nx)(Ny)(Fxy)]=[z=(Cxy)]}

4. Blank Rule:

(x){[~(Rx)]=(Bx)}

al = last activated attribute
A = attribute predicate
B = blank predicate
C = combination linkage
F = follows predicate
Fi = follows immediately predicate
L = linkage predicate
N = number predicate
Pi = precedes immediately predicate
R = recognition predicate
Rv = retained value linkage
V = value predicate
x = a word in the utterance
y = a word in the utterance
zl = a composite number to be inserted

Figure 4. Heuristic compositional rules.

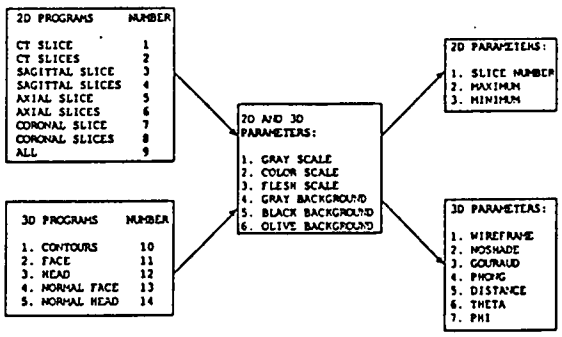

Figure 5. System programs and parameters.

4.5 AI inference submodule

Referring to Figure 2, the AI inference submodule developed for this study is a dual purpose inference submodule that contains a pragmatic and a critic rule base and supporting knowledge bases. The pragmatic and critic rule bases

consist of a collection of hierarchical structured subrule bases, as discussed by Tanimoto and Fischler.[8,17] The inference engine within the AI inference submodule performs the rule processing of both rule bases. The pragmatic rule and knowledge bases and the inference engine use the Associator submodule's redefined array structure to determine the user's intent. If the intent is not determinable from information within the array structure, a message (in ASCII) is sent to the Responder submodule to query the user for additional information. If the intent is determinable, the inference engine invokes additional functions within the pragmatic knowledge base to fulfill this intent. Program control is then passed to the Main module. If the determined intent is to display an image, program control is passed directly to the Main module. The critic rule and knowledge bases and the inference engine critique the Associator submodule formed plan for its validity. If the plan is not valid, a message (in ASCII) is sent to the Responder submodule to query the user for additional information. If the plan is valid, the Main module sends the plan directly to the Graphics driver submodule.

4.6 Responder submodule

The Responder submodule is responsible for sending error, status, query, and other system messages to the speech module and to the TI and Sun-4 computer displays.

4.7 Graphics driver submodule

The Graphics driver submodule places the plan (program number and parameters) critiqued by the AI inference submodule into a common file. It also sends an interrupt to the graphics module developed by Meadow to read the file and display the selected image.

5. SYSTEM PERFORMANCE

Figure 6, illustrates a verbal dialogue between a user and the voice actuated system. The dialogue in Session 1, displayed sagittal slice 33, shown in Figure 7. The figure shows sagittal slice 33, against a gray background and a gray scale legend with labels, maximum 255, and minimum 60. To demonstrate ellipsis, Session 2, shows the dialogue used for displaying slice 27. To intentionally create an error, the verbal dialogue in Session 3, contains an incorrect axial slice number that exceeds the actual number of available slices (256 slices). The system immediately responded "three hundred is outside the acceptable range for the axial slice plan." A valid axial value was then provided by the user. Session 4, shows the dialogue used to simultaneously display the axial, sagittal, and coronal views shown in Figure 8.

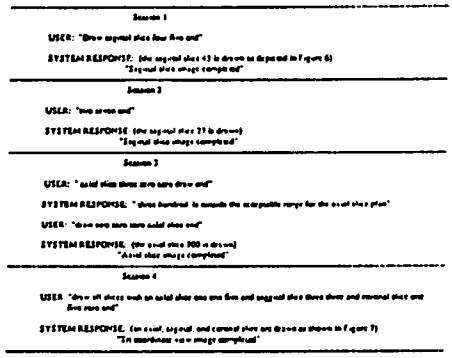

Figure 6. Example verbal dialogue.

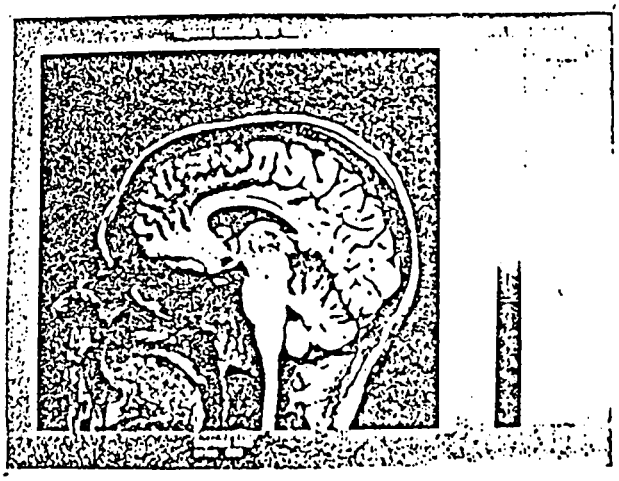

Figure 7. Sagittal slice 33.

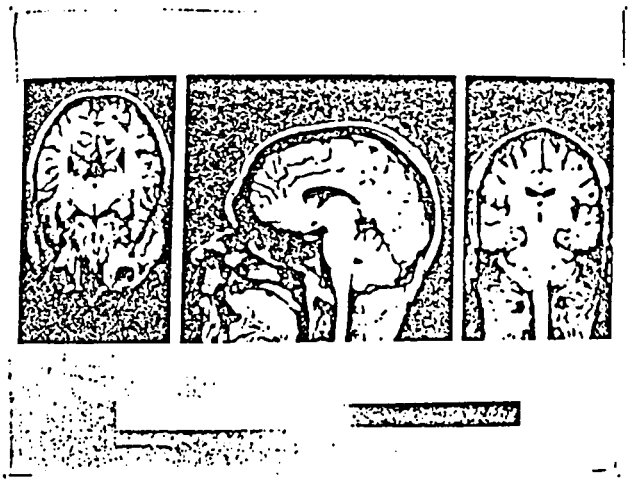

Figure 8. Axial, sagittal and coronal views.

6. SUMMARY AND CONCLUSIONS

This paper described the development and application of an AI control strategy module to a voice actuated computer graphics system. The interface did not require grammatical structure parsing, but used knowledge based inferencing to resolve valid input. The technique allowed the interface to resolve unusual syntactical sentence structures and sentence fragments that present commercially available interfaces cannot resolve. The technique also

allowed English input to be very informal and gave the user the freedom to converse in a more natural manner. To eliminate dialogue redundancy, it allowed the user to perform ellipsis. The technique also accommodated unknown words and the absence of user's trivial words that were not recognized by the system. Also, the technique does not require the use of a keyboard. The voice actuated system provided a means by which a user could verbally render complex medical imaging data into two or three dimensional displays and query the system for additional cross-sections, fields of view, projections, and etc. The control strategy made autonomous decisions from the user's utterances and participated in dialogue to resolve perceived inconsistencies between the user and its knowledge bases. The software provided: contextual focusing, an object-oriented approach to plan validation, dynamically interchangeable speech and keyboard modes, a dialogue modality, a high degree of modularity and data abstraction, and unrestricted verbal syntax. However, one problem encountered was that the vocabulary capacity of the speech module was not large enough to adequately accommodate the full capacity of the AI interface module. An area for further study is to repeat the project when a more sophisticated voice recognition and synthesis system becomes available.

7. ACKNOWLEDGEMENTS

The authors wish to acknowledge Texas Instruments, Inc., Martin Ras, and Robert E. Hollman, and Sun Microsystems, Inc., and Dr. Emil Sarpa, for their support in this endeavor.

8. REFERENCES

1. J. L. Scruggs, "Voice Control of Complex Workstations," 2nd Annual Workshop on Space Operations Automation and Robotics (SOAR), Report No: NASA CP-3019, Houston, TX, 1988.

2. T. B. Martin, "Practical Applications of Voice Input to Machines," Proceedings of the IEEE, vol. 13, no. 1, pp. 15-23, February 1987.

3. G. Philip, E. S. Young, "Man-Machine Interaction by Voice: Development in Speech Technology. Part II: General Applications, and Potential Applications in Libraries and Information Services," Journal of Information Science, vol. 13, no. 1, pp. 15-23, February 1987.

4. G. Bristow ed., Electronic Speech Recognition Techniques, Technology, & Applications, McGraw-Hill Book Co., 1986.

5. J. N. Holmes, Speech Synthesis and Recognition, Van Nostrand Reinhold (UK) Co. Ltd, 1988.

6. VoiceScribe-100 Speech Application Designer's Handbook, Release 2.10, Dragon Systems Inc., 1986.

7. D. Schutzer, Artificial Intelligence: An Applications-oriented Approach,Van Nostrand Reinhold Co., Inc., pp. 146-163, 1987.

8. M. A. Fischler, Intelligence: The Eye, the Brain, and the Computer, Addison-Wesley Publishing Co., Menlo Park, CA, pp. 165-188, 1987.

9. A. Barr, The Handbook of Artificial Intelligence, Heuristech Press, Stanford CA, pp. 225-361, 1981.

10. M. Rettig, "How to Choose Natural Language Software," AI Expert, vol. 3, no. 7, pp. 40-49, July 1988.

11. T. W. Winston, "Natural Language Query Parsing," AI Expert, vol. 4, no. 2, pp. 44-49, February 1989.

12. D. G. Bobrow, R. M. Kaplan, et al.,"GUS, a Frame-Driven Dialog System," Artificial Intelligence, vol. 8, pp. 155-173, 1977.

13. TI Speech Application Tool Kit Guide, Part number 2232384-0001, Revision C, Texas Instruments, June 1989.

14. J. L. Manferdelli, "Natural Languages," Sun Technology, vol. 2, no. 3, 1989.

15. J. S. Meadow, "Application of a Voice Actuated Computer System for Reconstruction and Display of Magnetic Resonance Imaging Data," presented at the SPIE-Society of Photo-Optical Instrumentation Engineers, Medical Imaging IV, Newport Beach, CA, February 4-9, 1990.

16. Sun Common Lisp 3.0 Advanced User's Guide, Sun Microsystems, August, 1988.

17. S. L. Tanimoto, The Elements of Artificial Intelligence, Computer Science Press, Inc., Rockville, MA, 1987.

APPENDIX C

AI MODULE SOFTWARE

```
;*****************************************************************************
;*****************************************************************************
;****                                                                     ****
;****                              AI MODULE                              ****
;****                                                                     ****
;**** Purpose: This module is responsible for interfacing the voice compon- ****
;****          ents of the Speech module to the graphics programs of the  ****
;****          Graphics module.                                           ****
;****                                                                     ****
;**** Developement: This software was developed at The University of Tenn. ****
;****              Space Institute by Patrick Leroy Sherley.              ****
;****                                                                     ****
;*****************************************************************************
;*****************************************************************************
```

```
;*****************************************************************************
;*                                                                           *
;*                          INITIALIZATION SUBMODULE                         *
;*                                                                           *
;*Purpose: This submodule loads the user help files, utility files, and foreign*
;*         files. It also loads the other submodules and other pertinent     *
;*         information. This submodule also builds the AI module's rule and  *
;*         knowledge bases and starts the Graphics module with a UNIX shell  *
;*         call.                                                             *
;*                                                                           *
;*****************************************************************************

;------------------- Load help files -----------------------------

(load "~/AI/screens.lisp")          ;Help and Parameter Screens maker

;------------------- Load utility files --------------------------------

(load "~/AI/Build-prog.lisp")       ;Flavors for building program objects
(load "~/AI/Build-environ.lisp")    ;Methods to bind general variable names
(load "~/AI/Util-prog.lisp")


;------------------ Load foreign files --------------------------------

(load-foreign-files "~/AI/lsig.o") ;C foreign functions

;----------------- Load Uterrance Loader Submodule --------------------

(load "~/AI/UL.lisp")               ;Utterance Loader Submodule

;----------- Load Translator Submodule (with its knowledge base)  -------

(load "~/AI/Translator.lisp")       ;Translator Submodule
(load "~/AI/Translator-kb.lisp")    ;Translator Knowledge Base

;------------------ Load Associator Submodule --------------------------

(load "~/AI/Associator.lisp")       ;Associator Submodule
```

```
;------------------ Load Responder Submodule ----------------------------

(load "~/AI/Responder.lisp")        ;Responder Submodule

;------------------ Load Graphics Driver Submodule ----------------------

(load "~/AI/Graphics-dvr.lisp")     ;Graphics Driver Submodule

;------------------ Load AI Inference Submodule -------------------------

(load "~/AI/Resolver.lisp")         ;Inference Engine

(load "~/AI/C-rules.lisp")          ;Critic rule base
(load "~/AI/C-kb.lisp")             ;Critic knowledge base

(load "~/AI/S-rules.lisp")          ;Pragmatic rule base
(load "~/AI/S-kb.lisp")             ;Pragmatic knowledge base

;----------------------- LOAD OTHER PERTINENT INFORMATION ------------------

(load "~/AI/Initialization.lisp")   ;default plan values and parameter
                                    ; definitions

;------------------- BUILD THE KNOWLEDGE STRUCTURES ----------------------

(Build_Translator_KB)               ;Build the knowledge bases
(Build_Critic_KB)
(Build_Pragmatic_KB)

(Build_Critic_RB)                   ;Build the rule bases
(Build_Pragmatic_RB)

;----------------------- Other Pertinent Information ----------------------

(setq slice-types '( sag axial cor))     ;types of slices

(setq plans '(ct-axial ct-incremental     ;available graphics programs
              sagittal-slice
              axial-slice
```

```
                coronal-slice
                incremental-sagittal
                incremental-axial
                incremental-coronal
                tri-coordinate-slice-view
                pixel-imaging
                face
                head
                help-screen
                exit
                volume))
(setq plan_list '( (ct-axial 2)           ;graphis programs plan numbers
                   (ct-incremental 3)
                   (sagittal-slice 4)
                   (axial-slice 5)
                   (coronal-slice 6)
                   (incremental-sagittal 7)
                   (incremental-axial 8)
                   (incremental-coronal 9)
                   (tri-coordinate-slice-view 10)
                   (pixel-imaging 12)
                   (face 14)
                   (head 15)
                   (help-screen 11)
                   (exit 1)
                   (volume 13)
))
```

```
(setq pos_list '(                          ;positions of parameters in plan
                 (plan 0)
                 (sag 1)
                 (axial 2)
                 (cor 3)
                 (max 4)
                 (min 5)
                 (dist 6)
                 (yaw 7)
                 (pitch 8)
                 (cscale 9)
                 (background 10)
                 (sag1cut 11)
                 (sag2cut 12)
                 (ax1cut 13)
                 (ax2cut 14)
                 (cor1cut 15)
                 (cor2cut 16)
))

(setq scale '(grayscale 21                 ;types of colormaps
              colorscale 22
              fleshtone 23
              yellowscale 24
))


(setq cut_list '(sag1cut sag2cut ax1cut    ; available cutting planes
                 ax2cut cor1cut cor2cut))
```

```
(setq plan '(                          ;initialize plan
            (plan 15)
            (sag 37)
            (axial 125)
            (cor 125)
            (max 255)
            (min 60)
            (dist 5000)
            (yaw 180)
            (pitch 90)
            (cscale 21)
            (background 31)
            (sag1cut 1)
            (sag2cut 60)
            (ax1cut 1)
            (ax2cut 255)
            (cor1cut 1)
            (cor2cut 255)
)))

;----------------------- START THE GRAPHICS MODULE -------------------------

(shell "/micro/john/scans/p_mrct_ &")
```

```
;*****************************************************************************
;*                                                                           *
;*                          Translator Submodule                             *
;*                                                                           *
;*Purpose: This submodule perform the module's idiom recognition and         *
;*         canonical conversions on the words. It checks a dictionary for    *
;*         the words in an array, and if the words are found in the          *
;*         dictionary, then they are converted to their internal language    *
;*         equivalent.                                                       *
;*                                                                           *
;*Major variables: sentence - the array containing the words to be converted*
;*                 dictionary - the internal language dictionary stored in   *
;*                              associate translator knowledge base          *
;*                                                                           *
;*****************************************************************************
(in-package 'user)
;---------------------------------------------------------------------------
;
;    Translator/Translate Functions
;    These functions converts the sentence into the internal format
;
;---------------------------------------------------------------------------
(defun translator ()
 (setq position_index 0)
  (loop while (not(eq (setf word (aref sentence count)) nil))
    do
    (setf (aref sentence position_index) (translate word))))


(defun translate (word)
  (loop for i in dictionary
    do
    (if (not(eq (member word i) nil))
    (setq word (car i)))
    while (eq (member word i) nil)
    finally
    (return word)))
```

```
;-------------------------------------------------------------------------
;
;     Translator Knowledge Base
;
;Purpose: This knowledge base contains the internal language dictionary used
;         by the Translator submodule
;
;-------------------------------------------------------------------------
(in-package 'user)
(defun Build_Translator_KB ()
(setq dictionary '(
              (plan routine routines plans draw drawn display see)
              (sag sagittal side)
              (axial vertical )
              (cor coronal horizontal)
              (max maximum  upper)
              (min minimum  lower)
              (dist x radius distance)
              (yaw y theta  )
              (pitch z tilt slant phi )
              (cscale scale colors)
              (background screen)
              (sag1cut right)
              (sag2cut left)
              (ax1cut top)
              (ax2cut bottom)
              (cor2cut back)
              (cor1cut front)
              (help assist assistance know list explain options )
              (rotate rotated  rotating rotation spin spun turn turned turning)
              (next advance forward increment)
              (last previous backward decrement)
              (use used with select selected  using)
              (fleshtone wine)
              (ct-axial ct-slice)
              (ct-incremental ct-slices)
              (sagittal-slice sag-slice)
              (axial-slice)
              (coronal-slice cor-slice)
```

```
(sagittal-slices sag-slices)
(axial-slices)
(coronal-slices cor-slices)
(incremental-sagittal incremental-sag inc-sag incremental-sag
 sagittal-slices)
(incremental-axial inc-axial axial-slices)
(incremental-coronal inc-coronal inc-cor incremental-cor
 coronal-slices)
(tri-coordinate-slice-view tri-view tri-slice
 tri-coordinate-view all)
(pixel-imaging pix pix-imaging contours)
(face )
(head cranium)
(help-screen screen-help helper)
(exit kill-graphics graphics-kill)
(rotating-head demo)
(welcome-screen)
(21 gray grey)
(22 spectrum red reddish color)
(23 flesh natural wine)
(24 yellow)
(32 black dark)
(31 standard )
(0 zero)
(1 one first)
(2 two second)
(3 three third)
(4 four four fourth)
(5 five fifth)
(6 six sixth)
(7 seven seventh)
(8 eight eighth)
(9 nine ninth)
(10 ten tenth)
(11 eleven eleventh)
(12 twelve twelveth)
(13 thirteen thirteenth)
(14 fourteen  fourteenth)
(15 fifteen fifteenth)
```

```
(16 sixteen sixteenth)
(17 seventeen seventeenth)
(18 eighteen eighteenth)
(19 nineteen nineteenth)
(20 twenty twentieth)
(30 thirty thirtieth)
(40 forty fortieth)
(50 fifty fiftieth)
(60 sixty sixtieth)
(70 seventy seventieth)
(80 eighty eightieth)
(90 ninety ninetieth)))
```

```
;*****************************************************************************
;*                                                                           *
;*                          Associator Submodule                             *
;*                                                                           *
;*Purpose: This submodule applies the hueristic convergence, proximity, and  *
;*         ghost rules to group the words in the array structure in order to *
;*         develop the plan.                                                 *
;*                                                                           *
;*         The convergence rule states that if two adjacent array locations  *
;*         contain numbers, then the numbers must be combined to form an     *
;*         equivalent composite number; "blank" is put in the first of the two *
;*         array locations, and the generated composite number is placed in the*
;*         second location                                                   *
;                                                                           *
;*         The proximity rule states that a parameter in the array structure *
;*         should be assigned the number in the array to which the parameter *
;*         is related. This relationship is based on the  proximty of the    *
;*         parameter and value to each other, and can be of three cases:     *
;*                                                                           *
;*         Case1: The parameter has a number occuring immediately before it  *
;*                in the array.                                              *
;*         Case2: The parameter is followed in the array (not necessarily    *
;*                immediately) by a number.                                  *
;*         Case3: The parameter has a number occuring two locations before   *
;*                it in the array.                                           *
;*                                                                           *
;*         If none of the proximity cases apply, then the ghost rule is applied*
;*         which states that if a value cannot be found for a parameter, then *
;*         the parameter retains its last assigned value.                    *
;*                                                                           *
;*Functions used: switcher - sets the input mode                             *
;*                unifier - forms the program name                           *
;*                de_blanker - removes the blanks from the array             *
;*                non_blank - counts number of blanks before next non-blank  *
;*                                                                           *
;*Major variables: sentence - array containing words converted by Translator *
;*                 plan - parameter value list containing data for graphics  *
;*                        module                                             *
```

```
;*****************************************************************************
(in-package 'user)
;-----------------------------------------------------------------
;
;    Associator Function
;    This function groups the words in the array  to update the plan
;
;-----------------------------------------------------------------
(defun Associator ()
  (switcher)          ;set input mode for module
  (unifier)           ;form program name
;
;----- This code applies the convergence rule to group numbers ---------
;
   (setq count 0)
  (loop while (not(eq (setf word (aref sentence count)) nil))
    do
     (cond((member word ddigits)
        (if (member (aref sentence (+ count 1)) digits)
        (setf word (cadr (assoc word digiter)))
        (cond ((and (> count 0) (numberp (aref sentence (- count 1))))
               (setf (aref sentence count)
               (+ (* 100 (aref sentence (- count 1)))
               (cadr (member word ddigits))))
               (setf (aref sentence (- count 1)) 'blank))
             ((setf (aref sentence count) (cadr (member word ddigits)))))))))


    (if
        (setf num (cadr (member word digits)))
        (and (if (numberp num)(setf (aref sentence count) num))
             (cond((and (> count 0) (numberp (aref sentence (- count 1)))
                   (numberp (aref sentence count)))
                   (setf (aref sentence count)
                   (+ (* 10 (aref sentence (- count 1)))
                   (aref sentence count)))
                   (setf (aref sentence (- count 1)) 'blank)))))
    (setq count (+ count 1)))

  (if (setq spot (position 'hundred sentence))
```

```
      (cond((and (numberp (aref sentence (- spot 1)))
                 (not (numberp (aref sentence (+ spot (non_blank spot))))))
            (setf (aref sentence spot) (* 100 (aref sentence (- spot 1))))
            (setf (aref sentence (- spot 1)) 'blank))
           ((numberp (aref sentence (+ spot (non_blank spot))))
            (setf (aref sentence (+ spot (non_blank spot))) (+ (* 100 (aref sentence (- spot
1))) (aref sentence (+ spot (non_blank spot)))))
            (setf (aref sentence spot) 'blank)
            (setf (aref sentence (- spot 1)) 'blank))))

  (if (setq spot (position 'thousand sentence))
      (cond((and (numberp (aref sentence (- spot 1)))
                 (not (numberp (aref sentence (+ spot (non_blank spot))))))
            (setf (aref sentence spot) (* 1000 (aref sentence (- spot 1))))
            (setf (aref sentence (- spot 1)) 'blank))
           ((numberp (aref sentence (+ spot (non_blank spot))))
            (setf (aref sentence (+ spot (non_blank spot))) (+ (* 1000 (aref sentence (- spot
1))) (aref sentence (+ spot (non_blank spot)))))
            (setf (aref sentence spot) 'blank)
            (setf (aref sentence (- spot 1)) 'blank))))

 (de_blanker)                ; remove blanks in the array


;------------------- Develop and update the plan -------------------------
;
 (setq count 0)
  (loop while (not(eq (setq parameter (aref sentence count)) nil))
    do
    (cond((not(or (eq parameter 'plan) (numberp parameter)
                  (and (member parameter cut_list)
                  (position 'rotate Sentence))))
    (if (setq datum (cadr (member  parameter scale)))
        (setq plan (substitute (list 'cscale datum) (assoc 'cscale plan) plan)))

    (cond( (not(eq(setq value (next-num count)) nil))  ;if nil, ghost rule applies

        (if (and (eq parameter 'background)  (eq value 21))
            (setq value (+ value 10)))
```

```
        (setq d (substitute (list parameter value)
        (assoc parameter d) d))))))
    (setq count (+ count 1)))

   (if (or (and (position 'head sentence)
            (not (eq (car d3) 15)))
           (position 'restore sentence))
       (and (setq d (substitute (list 'sag1cut 1) (assoc 'sag1cut d) d))
            (setq d (substitute (list 'sag2cut 61) (assoc 'sag2cut d) d))
            (setq d (substitute (list 'ax1cut 1) (assoc 'ax1cut d) d))
            (setq d (substitute (list 'ax2cut 255) (assoc 'ax2cut d) d))
            (setq d (substitute (list 'cor1cut 1) (assoc 'cor1cut d) d))
            (setq d (substitute (list 'cor2cut 255) (assoc 'cor2cut d) d)))))


;-------------------------------------------------------
;
;   Non_blank Function
;   This function returns how many array locations
;   away the first array location not containig a blank
;   is from the current spot1 argument
;
;-------------------------------------------------------
(defun non_blank (spot1)
  (setq counter (+ spot1 1))
  (setq blanks 1)
  (loop while (eq (aref sentence counter) 'blank)
   do
   (setq blanks (+ blanks 1))
   (setq counter (+ counter 1))
   finally
   (return blanks)))


;-----------------------------------------------------------
;
;   De_blanker Function
;   This function compresses the array by removing the blanks
;   generated by the convergence function
```

```
;
;-----------------------------------------------------
(defun de_blanker ()
  (setq count 0)
  (loop while (not (eq (aref sentence count) nil))
   do
   (cond((eq (aref sentence count) 'blank)
         (setq counter count)
         (loop while (not (eq (aref sentence counter) nil))
           do
           (setf (aref sentence counter) (aref sentence (+ counter 1)))
           (setq counter (+ counter 1)))))
  (setq count (+ count 1)))
  (if (position 'blank sentence)(de_blanker)))


;-----------------------------------------------------------
;
;   Switcher function
;   This function switches the system between the voice and
;   speech modes
;
;-----------------------------------------------------------
(defun switcher ()
   (setq count 0)
   (setq test t)
   (loop while (and test (not (eq (aref sentence count) nil)))
   do
   (cond((eq (aref sentence count) 'voice)
         (setq keyboard nil)
         (setq test nil))
        ((eq (aref sentence count) 'keyboard)
         (setq keyboard t)
         (setq test nil)))
   (setq count (+ count 1))))
```

```
;-------------------------------------------------------------
;
;   Unifier Function
;   This function groups the program name words in the array to
;   form a complete program name
;
;-------------------------------------------------------------
(defun unifier ()
   (cond( (position 'plan Sentence)
         (cond((and (setf place (position 'slice Sentence))
                    (< 0 (- place 1))
              (member (translate (aref sentence (- place 1))) slice-types))
                (setf (aref Sentence place) (read-from-string
      (format nil "~S-slice" (setf slice (aref Sentence (- place 1))))))
                (setf (aref Sentence (- place 1)) (aref sentence place))
                (setf (aref Sentence place) slice))
               ((and (setf place (position 'slices Sentence))
                    (< 0 (- place 1))
      (member (translate (aref sentence (- place 1))) slice-types))
                (setf (aref Sentence place) (read-from-string
      (format nil "~S-slices" (setf slice (aref Sentence (- place 1))))))
                (setf (aref Sentence (- place 1)) (aref sentence place))
                (setf (aref Sentence place) slice))
        ((and (not (position 'coronal Sentence))
              (not (position 'sagittal Sentence))
              (not (position 'axial Sentence)))
         (setf point (position 'slice Sentence))
         (if (eq slice 'ct) (setf slice 'axial))
         (if point (setf (aref Sentence point) slice)))))))

;-------------------------------------------------------------
;
;    Justify Function
;    This function converts the program name into a numeric format
```

```
;
;-------------------------------------------------------------
(defun Justify ()
 (setq count 0)
   (loop while (not(eq (setq datum (aref sentence count)) nil))
   do
   (if (member (translate datum) plans)
       (setq d (substitute (list 'plan (cadr (assoc (translate datum) plan_list)))
       (assoc 'plan d) d)))
   (setq count (+ count 1))
   finally
   (update)))


;-------------------------------------------------------------
;
;    Next_num Function
;    This function groups a parameter with its appropriate value
;    by applying the proximity.
;
;-------------------------------------------------------------
(defun next_num (count1)
  (setq count3 (- count1 1))
  (setq count4 (- count1 2))
  (loop while (not(or(numberp (aref sentence count1)) (eq (aref sentence count1) nil)))
    do
    (setq count1 (+ count1 1))
    finally
   (cond
;----------- Proximity case when a number immediately precedes a parameter -----

          ((and (< 0 count3) (numberp (aref sentence count3))
               (not (member (aref sentence count4) params)))
          (return (aref sentence count3)))

;----------- Proximity case when a number follows a parameter ------------------

         ((numberp (aref sentence count1))
          (return (aref sentence count1)))
```

```
;---------- Proximity case when a number precedes a parameter by 2 locations --

          ((and (< 0 count4) (numberp (aref sentence count4))
                (not (member (aref sentence (- count4 1) params))))
           (return (aref sentence count4))))))
```

```
;*****************************************************************************
;*                                                                           *
;*                        AI INFERENCE SUBMODULE                             *
;*                                                                           *
;*Purpose: This submodule determines the user's intent and validates the plan *
;*         formed by the Associator Submodule                                *
;*                                                                           *
;*Major sections: Pragmatic rule base - contains rules for determining intent *
;*                Pragmatic knowledge base - contains definitions to support *
;*                                           the pragmatic rule base         *
;*                Critic rule base - contains rules for validating plan data *
;*                Critic knowledge base - contains definitions to support the *
;*                                        critic rule base                   *
;*                Inference engine - contains functions to process the rules *
;*                                                                           *
;*****************************************************************************
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;    Pragmatic Rule Base
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun Pragmatic_rb ()
  (list


      '(rule1
          (if
              (The_ Sentence contains_no parameters)
              (The_ Sentence has_no value))
          (then
              (Execute Reword_request)))

      '(rule2
          (if
             (The_ Sentence contains help_word))
          (then
             (Execute Helper)))
```

```
      '(rule3
          (if
              (The_ Sentence contains plan_parameters)
              (The_ Sentence has_no value))
          (then
              (Execute Reword_request)))


      '(rule4
          (if
              (The_ Sentence is old))
          (then
              (Execute Repeat_message)))

      '(rule5
          (if
              (The_ Sentence is_not old))
          (then
              (Process 'Sentence_handler)))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;     SENTENCE HANDLER RULE BASE
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun Sentence_handler_rb ()
  (list

      '(rule1
          (if
              (The_ Sentence contains query))
          (then
              (Execute Answer_sentence)))
```

```
    '(rule2
         (if
             (The_ Sentence contains 'rotate))
        (then
             (Process 'Rotate_handler)))

     '(rule3
         (if
             (The_ Sentence contains direction))
        (then
             (Process 'Rotate_handler)))

    '(rule4
         (if
             (The_ Sentence contains_no parameters))
         (then
             (Execute Insert)))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;    ROTATE HANDLER RULE BASE
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun Rotate_handler_rb ()
  (list

     '(rule1
         (if
             (The_ plan is_not 3-D))
         (then
             (Execute Dimension_error2)))
```

```
'(rule2
    (if
        (The_ direction is 'home))
    (then
        (Execute Home_adjustment)))

'(rule3
    (if
        (The_ direction is back))
    (then
        (Execute Back_adjustment)))

'(rule4
    (if
        (The_ direction is 'left_side))
    (then
        (Execute Left_side_adjustment)))

'(rule5
    (if
        (The_ direction is 'right_side))
    (then
        (Execute Right_side_adjustment)))

'(rule6
    (if
        (The_ direction is 'up))
    (then
        (Execute Up_adjustment)))

'(rule7
    (if
        (The_ direction is 'down))
    (then
        (Execute Down_adjustment)))
```

```
    '(rule8
        (if
            (The_ direction is left))
        (then
            (Execute Left_adjustment)))

    '(rule9
        (if
            (The_ direction is right))
        (then
            (Execute Right_adjustment)))

    '(rule9
        (if
            (The_ direction is not_given))
        (then
            (Execute Direction_request)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   Pragmatic Knowledge Base
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun Sentence_dictionary ()
   (setq empty nil)
   (setq not_given nil)
   (setq intent nil)
   (setq help_word 'help)
   (setq contains 'contains)
   (setq contains_no 'contains_no)
   (setq has_no 'has_no)
   (setq numbers 'numbers)
   (setq value 'value)
   (setq cut 'cut)
   (setq back 'cor2cut)
   (setq right 'sag1cut)
   (setq left 'sag2cut)
```

```
   (setq 3-D '(p11 p12 p13 p14 p15))
   (setq query '(who what when where why how  are is ))
   (setq parameters '(plan rotate keyboard voice next stop last sag axial cor max min dist
yaw pitch cscale
    background saglcut sag2cut axlcut ax2cut corlcut cor2cut greyscale colorscale fleshtone
    restore ))
   (setq plan_parameters '(max min dist yaw pitch cscale
    background  saglcut sag2cut axlcut ax2cut corlcut cor2cut)))


;-------------------------------------------------------------
;
;    Keyword Function
;    This function identifies the first recognized help sensitive word
;
;-------------------------------------------------------------
(defun keyword ()
   (setq key nil)
   (setq count2 0)
   (loop while (not(eq (aref sentence count2) nil))
    do
      (if (not(eq (assoc (aref sentence count2) class) nil))
        (setq key (aref sentence count2)))
      (setq count2 (+ count2 1))
      finally
      (return key)))


;-------------------------------------------------------------
;
;    Insert Function
;    This function places a parameter in a sentence by default
;
;-------------------------------------------------------------
(defun Insert()
  (cond((or (numberp (aref Sentence 0)) (eq (aref sentence 0) 'blank))
   (Find_parameter)
   (setq count4 0)
```

```
   (loop while (not (eq (aref sentence count4) nil))
   do
    (setq count4 (+ count4 1))
   finally (setf (aref sentence count4 ) value))
   (swap)
   (update))))

(defun Find_parameter ()
  (setq pno (car d2))
     (cond ((or (eq pno 4) (eq pno 7))
            (setq value 'sag))
           ((or (eq pno 2) (eq pno 5) (eq pno 8))
            (setq value 'axial))
           ((or (eq pno 6) (eq pno 9))
            (setq value 'cor))
           ((eq pno 10)
            (slice_prompt))))

(defun slice_prompt ()
   (setq msg "what type of slice are you referring: axial coronal or sagittal")
   (print msg)
   (efree '(writer msg))
   (efree '(reader))
   (if (member (setq slice-type (read-from-string n)) '(axial coronal sagittal))
       (setq value slice-type))
   (cond((eq value 'coronal)
         (setq value 'cor))
        ((eq value 'sagittal)
         (setq value 'sag))))
```

```
;-------------------------------------------------------------
;
;    Contains Function
;    This function detrmines if the two args have any common elements
;
;-------------------------------------------------------------
(defun contains (first second)
   (setq marker 0 )
   (if (atom second) (setq second (list second)))
   (loop while (not (eq (aref first marker) nil))
   do
     (if (member (aref first marker) second) (return t))
     (setq marker (+ marker 1))
   finally (return nil)))

;-------------------------------------------------------------
;
;    Contains_no Function
;    This function checks that no element is common to the two args
;
;-------------------------------------------------------------
(defun contains_no (first second)
   (setq marker 0 )
   (if (atom second) (setq second (list second)))

   (loop while (not (eq (aref first marker) nil))
   do
     (if (member (aref first marker) second) (return nil))
     (setq marker (+ marker 1))
   finally (return t)))
```

```
;-------------------------------------------------------------
;
;    Dimension_error Function
;    This function prints a dimension error message
;
;-------------------------------------------------------------
(defun Dimension_error1 ()
   (setf flag1 t)
   (setf flag3 nil)
   (setq msg "Only the 3-Dimensional head images can use cuts ")
   (efree '(writer msg)))
;-------------------------------------------------------------
;
;    Dimension_error2 Function
;    This function prints a dimension error message
;
;-------------------------------------------------------------
(defun Dimension_error2 ()
   (setf flag1 t)
   (setf flag3 nil)
   (setq msg "Only a 3-Dimensional images can be rotated ")
   (efree '(writer msg)))
;-------------------------------------------------------------
;
;    Has_no_value Function
;
;-------------------------------------------------------------
(defun has_no ( first second)
  (setq count 0)
  (loop while (not (eq (aref sentence count) nil))
   do
   (if (numberp (aref sentence count)) (return nil))
   (setq count (+ count 1))
   finally (return t)))
```

```
;-------------------------------------------------------------
;
;    Left_adjustment/Draw_left_pos Functions
;    These functions adjust for any left rotation
;
;-------------------------------------------------------------
(defun Left_adjustment ()
   (cond ((numberp (setq dirnum (next-num (position direction sentence))))
          (setq unit (mod dirnum 360))
          (Draw_left_pos))
         (t(Query_user 'degrees)
           (Draw_left_pos))))

(defun Draw_left_pos ()

   (setq yold (cadr (assoc 'yaw d)))
          (if (minusp (- yold (+ unit 1)))
            (setq ynew (- (+ 360 yold) unit))
           (setq ynew (- yold unit)))
          (if (eq ynew 0)
              (setq ynew 1))
          (setq d (substitute (list 'yaw ynew) (assoc 'yaw d) d))
          (update))


;-------------------------------------------------------------------
;
;    Right_adjustment/Draw_right_pos Functions
;    These functions adjust for any right rotation
;
;-------------------------------------------------------------------
(defun Right_adjustment ()

   (cond ((numberp(setq dirnum (next-num (position direction sentence))))
          (setq unit (mod dirnum 360))
          (Draw_right_pos))
         (t(Query_user 'degrees)
           (Draw_right_pos))))
```

```
(defun Draw_right_pos ()

    (setq yold (cadr (assoc 'yaw d)))
          (setq ynew (mod (+ yold unit) 360))
          (if (eq ynew 0)
              (setq ynew 1))
          (setq d (substitute (list 'yaw ynew) (assoc 'yaw d) d))
          (update))


;---------------------------------------------------------------------
;
;    Down_adjustment/Draw_down_pos Functions
;    These functions adjust for any down rotation
;
;---------------------------------------------------------------------

(defun Down_adjustment ()

     (cond ((numberp(setq dirnum (next-num (position direction sentence))))
            (setq unit (mod dirnum 180))
            (Draw_down_pos))
           (t(Query_user 'degrees)
             (Draw_down_pos))))

(defun Draw_down_pos ()

     (setq zold (cadr (assoc 'pitch d)))
          (if (minusp (- zold (+ unit 1)))
            (setq znew (- (+ 180 zold) unit))
            (setq znew (- zold unit)))
          (if (eq znew 0)
              (setq znew 1))
          (setq d (substitute (list 'pitch znew) (assoc 'pitch d) d))
          (update))
```

```
;-----------------------------------------------------------------------
;
;    Up_adjustment/Draw_up_pos Functions
;    These functions adjust for any up rotation
;
;-----------------------------------------------------------------------

(defun Up_adjustment ()

     (cond ((numberp(setq dirnum (next-num (position direction sentence))))
            (setq unit (mod dirnum 180))
            (Draw_up_pos))
           (t(Query_user 'degrees)
             (Draw_up_pos))))

(defun Draw_up_pos ()

     (setq zold (cadr (assoc 'pitch d)))
          (setq znew (mod (+ zold unit) 180))
          (if (eq znew 0)
              (setq znew 1))
          (setq d (substitute (list 'pitch znew) (assoc 'pitch d) d))
          (update))

;-----------------------------------------------------------------------
;
;    Home_adjustment Function
;    This function displays the front of the object
;
;-----------------------------------------------------------------------
(defun Home_adjustment ()
    (setq d (substitute (list 'yaw 180) (assoc 'yaw d) d))
    (setq d (substitute (list 'pitch 90) (assoc 'pitch d) d))
    (update))
```

```
;---------------------------------------------------------------------
;
;    Back_adjustment Function
;    This function displays the back of the object
;
;---------------------------------------------------------------------
(defun Back_adjustment ()
    (setq d (substitute (list 'yaw 360) (assoc 'yaw d) d))
    (setq d (substitute (list 'pitch 90) (assoc 'pitch d) d))
    (update))

;---------------------------------------------------------------------
;
;    Left_side_adjustment Function
;    This function displays the left side of the object
;
;---------------------------------------------------------------------
(defun Left_side_adjustment ()
    (setq d (substitute (list 'yaw 90) (assoc 'yaw d) d))
    (setq d (substitute (list 'pitch 90) (assoc 'pitch d) d))
    (update))

;---------------------------------------------------------------------
;
;    Right_side_adjustment Function
;    This function displays the right side of the object
;
;---------------------------------------------------------------------
(defun Right_side_adjustment ()
    (setq d (substitute (list 'yaw 270) (assoc 'yaw d) d))
    (setq d (substitute (list 'pitch 90) (assoc 'pitch d) d))
    (update))
```

```
;-------------------------------------------------------------------
;
;    Top_adjustment Function
;    This function displays the top of the object
;
;-------------------------------------------------------------------
(defun Top_adjustment ()
    (setq d (substitute (list 'yaw 180) (assoc 'yaw d) d))
    (setq d (substitute (list 'pitch 180) (assoc 'pitch d) d))
    (update))

;-------------------------------------------------------------------
;
;    Bottom_adjustment Function
;    This function displays the bottom of the object
;
;-------------------------------------------------------------------
(defun Bottom_adjustment ()
    (setq d (substitute (list 'yaw 180) (assoc 'yaw d) d))
    (setq d (substitute (list 'pitch 1) (assoc 'pitch d) d))
    (update))
;-------------------------------------------------------------------
;
;    Query_user Function
;    This function obtainsa missing degree measurement
;
;-------------------------------------------------------------------
(defun Query_user (item)
  (prog ()
    (setq msg (format nil "Tell me how many degress to rotate it ~A" direction))
    (efree '(writer msg))
    (Utterance-loader)
    (translator)
    (setq count 0)
    (loop while (not (eq (setq word (aref sentence count)) nil))
    do
    (if (numberp word) ( setq unit word))
    (setq count (+ count 1)))))
```

```
(defun Reword_request ()
   (setq intent nil)
   (setq type2 'reword)
   (error_write))

(defun Direction_request ()
    (setq intent nil)
    (setq type2 'direction)
    (error_write))

(defun Repeat_message ()
  (setq intent nil)
  (setq type2 'repeat)
  (error_write))


(defun Answer_sentence ()
   (setq intent nil)
   (setq type2 'answer)
   (error_write))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   Critic Rule Base
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun Critic_rb ()
  (list

      '(rule1
          (if
              (The_ intent is not_known))
          (then
              (Execute Nothing)))
```

```
'(rule2
    (if
        (The_ plan is unknown))
    (then
        (Execute Handle_error 'plan)))

'(rule3
    (if
        (The_ 'sag is relevant)
        (The_ sag is_not_within constraint_sag))
    (then
        (Execute Handle_error 'sag)))

'(rule4
    (if
        (The_ 'axial is relevant)
        (The_ axial is_not_within constraint_axial))
    (then
        (Execute Handle_error 'axial)))

'(rule5
    (if
        (The_ 'cor is relevant)
        (The_ cor is_not_within constraint_cor))
    (then
        (Execute Handle_error 'cor)))

'(rule6
    (if
        (The_ 'max is relevant)
        (The_ max is_not_within constraint_max))
    (then
        (Execute Handle_error 'max)))
```

```
'(rule7
    (if
        (The_ 'min is relevant)
        (The_ min is_not_within constraint_min))
    (then
        (Execute Handle_error 'min)))

'(rule8
    (if
        (The_ 'dist is relevant)
        (The_ dist is_not_within constraint_distance))
    (then
        (Execute Handle_error 'dist)))

'(rule9
    (if
        (The_ 'yaw is relevant)
        (The_ yaw is_not_within constraint_yaw))
    (then
        (Execute Handle_error 'yaw)))

'(rule10
    (if
        (The_ 'pitch is relevant)
        (The_ pitch is_not_within constraint_yaw))
    (then
        (Execute Handle_error 'pitch)))

'(rule11
    (if
        (The_ 'cscale is relevant)
        (The_ cscale is_not_within constraint_cscale))
    (then
        (Execute Handle_error 'cscale)))
```

```
'(rule12
    (if
        (The_ 'background is relevant)
        (The_ background is_not_within constraint_background))
    (then
        (Execute Handle_error 'background)))

'(rule13
    (if
        (The_ 'sag1cut is relevant)
        (The_ sag1cut is_not_within constraint_sag1cut))
    (then
        (Execute Handle_error 'sag1cut)))

'(rule14
    (if
        (The_ 'sag2cut is relevant)
        (The_ sag2cut is_not_within constraint_sag2cut))
    (then
        (Execute Handle_error 'sag2cut)))

'(rule15
    (if
        (The_ 'ax1cut is relevant)
        (The_ ax1cut is_not_within constraint_ax1cut))
    (then
        (Execute Handle_error 'ax1cut)))

'(rule16
    (if
        (The_ 'ax2cut is relevant)
        (The_ ax2cut is_not_within constraint_ax2cut))
    (then
        (Execute Handle_error 'ax2cut)))
```

```
      '(rule17
          (if
              (The_ 'corlcut is relevant)
              (The_ corlcut is_not_within constraint_corlcut))
          (then
              (Execute Handle_error 'corlcut)))


      '(rule18
          (if
              (The_ 'cor2cut is relevant)
              (The_ cor2cut is_not_within constraint_cor2cut))
          (then
              (Execute Handle_error 'cor2cut)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;              KNOWLEDGE BASES
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;*****************************************************
;     Function List Knowledge Base
;*****************************************************
(defun Constraint_Functions ()
  (list
       'The_
       'is
       'is_not
       'is_not_within
       'graphics
))

;*****************************************************
;     Critic Knowledge Base
;*****************************************************
(defun Critic_Definitions ()
  (setq unknown 'p99)
```

```
  (setq not_known nil)
  (setq is 'is)
  (setq is_not_within 'is_not_within)
  (setq is_not 'is_not))


;-------------------- The Function ------------------
(defun The_ (first second third)
    (prog ()
    (setq a first)
    (setq b third)
    (setq x (list second  'a 'b))
    (if (eval x)
        (return t))))

;-------------------- Is Function -------------------
(defun is (first second)
    (prog ()
    (if (or (eq first second) (if (listp second) (member first second)) )
        ( return t)
        (return nil))))

;------------------- Is_not Function ----------------
(defun is_not (first second)
    (prog ()
    (if (not (or (eq first second) (if (listp second)(member first second))))
        (return t)
        (return nil))))

;-------------------- Is_not_within Function --------
(defun is_not_within (first second)
    (prog ()
    (if (and (>= first (car second))
             (<= first (cadr second)))
        (return nil)
        ( return t))))
;------------------- Nothing Function ---------------
(defun Nothing ())
```

```
;-------------------Handle_error Function -----------
(defun Handle_error (type)
  (setq type2 type)
  (efree '(error_write)))

(defun error_write ()
 (setq flag1 t)
 (cond(keyboard
       (cond ((eq type2 'plan)
        (Vary pool1)
        (format t response)
        (terpri))
             ((eq type2 'reword)
        (Vary pool2)
        (format t response)
        (terpri))
             ((eq type2 'direction)
        (Vary pool3)
        (format t response)
        (terpri))
             ((eq type2 'repeat)
        (Vary pool4)
        (format t response)
        (terpri))
             ((eq type2 'answer)
        (Vary pool5)
        (format t response)
         (terpri))
       (t (format t "The value of ~A , ~A,is outside the acceptable range, ~A, for the  ~A
plan try using a valid value this time" type2 (eval type2) b p_name)
          (terpri))))
      (t
 (cond ((eq type2 'plan)
        (format voice "Sorry, but I do not know how to generate a plan to do that")
        (terpri voice))
             ((eq type2 'reword)
        (setq msg "I dont understand what you want done")
        (efree '(writer msg))
       )
```

```
            ((eq type2 'direction)
          (setq msg "You need to specify which direction")
          (efree '(writer msg)))
            ((eq type2 'repeat)
          (setq msg " I dont perceive anything new from that statement")
          (efree '(writer msg)))
            ((eq type2 'answer)
          (setq msg "How do I know, I am only a baby")
          (efree '(writer msg)))
         (t (format t "~A is outside the acceptable range for the ~A " (eval type2) p_name)
           (setq msg "look at the systems screen ")
           (efree '(writer msg)))))))


(defun Vary (group)
  (setq state (random 100))
  (cond((< state 25)
        (setq response (caar group)))
       ((< state 50)
        (setq response (caadr group)))
       ((< state 75)
        (setq response (caaddr group)))
       ((setq response (caar (cdddr group))))))


(defun Pooler ()
   (setq pool1 '(("I don't know how to plan that task")
                 ("The graphics system can't do that")
                 ("I don't know how")
                 ("That is not a plannable task")))
   (setq pool2 '(("I can not make an inference from that")
                 ("I do not understand what you want")
                 ("Can you rephrase that to help me understand")
                 ("I can not interpret that phrase")))
   (setq pool3 '(("You need to specify which direction")
                 ("Give me a direction of rotation")
                 ("You forgot to indicate what direction")
                 ("I need a direction before I can do that")))
   (setq pool4 '(("You already have that displayed")
                 ("You are repeating yourself")
```

```
                ("It is already on the display screen")
                ("I do not see anything new in that phrase")))
  (setq pool5 '(("My question answering module is disabled")
                ("I am still unable to answer that question")
                ("I am not sure in my present state")
                ("I can not answer that yet"))))


;-------------------------------------------------------------------
;
;                          Inference Engine
;
;Functions: Inference - parses and processes the rules
;           Prove - parses a rule's if part into individual if clauses
;           Process_if - evaluates the if clauses
;           Process - executes a rule's then part
;
;-------------------------------------------------------------------

;----------------- Inference Function -------------------------

(defun Inference ()
    (loop for i in rule_group
     do
     (setq if_part (cdadr i))
     (setq then_part (car (cdaddr i)))
     (cond
           ((Prove if_part )
            (Process then_part)
            (return nil)))
     finally
    (return t)))

;----------------- Prove Function -----------------------------

(defun Prove (if_part)
   (prog ()
    loop
```

```
     (cond
          ((not (eq (car if_part) nil))
           (setq new (car if_part))
     (cond ((setq ok (Process_if ))
            (setq if_part (cdr if_part))
            (go loop))
           ((not ok) (return nil))))
           (t(return t)))))

;---------------- Process_if Function -------------------------

(defun Process_if ()
   (prog ()
   (setq funct_name (car new))
   (cond ((not(eq (member funct_name function_list) nil))
          (if (eval new) (return t)))
         (t(return nil)))))

;---------------- Process Function ----------------------------
(defun Process (action)
   (cond
        ((eq (car action) 'Process)
         (setq rule_group (eval (eval (cadr action))))
         (Inference))
        ((eq (car action) 'Execute)
         (eval (cdr action)))))
```

```
;*****************************************************************************
;*                                                                           *
;*                        GRAPHICS DRIVER SUBMODULES                         *
;*                                                                           *
;*Purpose: This submodule transfers the plan to the Graphics module          *
;*        and issues an interrupt to the Graphics module to cause the        *
;*        module to read and begin processing of the plan                    *
;*                                                                           *
;*                                                                           *
;*****************************************************************************

;---------------------------------------------------------------------
;
;    Drive_Graphics Function
;    This function  sets unused data in plan to -1 (for compatibility
;    with the Graphics module), and executes the functions that sends the
;    the plan data (send_data function) and the interrupt (send_sig)
;
;---------------------------------------------------------------------
(defun Drive_Graphics ()
  (setq d3 d2)
  (loop for i in plan
    do
    (if (not(member (car i) relevant))
        (setf (elt d2 (cadr (assoc (car i) pos_list))) -1)))
  (send_data d2)
  (update)
  (shell "cp /tmp/tmp.list /tmp/param.list " )
  (shell "chmod 0777 /tmp/param.list /tmp/tmp.list")
  (shell "cat /tmp/param.list >> log1.list")
  (send_sig))
```

```
;-----------------------------------------------------------------------
;
;    Send_data Function
;    This function transfers the plan data to a file
;
;-----------------------------------------------------------------------
(defun send-data (dl )
  (prog ()
   (setq tst t)
   (loop while tst
    do
   (cond((not(efree '(checkfile)))
          (with-open-file
          (Dfile "/tmp/tmp.list" :Direction :Output)
          (loop for item in dl do (princ item Dfile)  (princ " " Dfile))
          (setq tst nil)))
        (t(setq tst t))))))

;-----------------------------------------------------------------
;
;    Checkfile Function
;    This function checks to see if the graphics process has read the datafile
;
;-----------------------------------------------------------------
(defun checkfile ()
   (with-open-file
      (Dfile "/tmp/param.list" :Direction :Input)
        (read Dfile)))
(in-package 'user)
;-------------------------------------------------------------
;
;    Reader Function
;    This function inputs a sentence from the voice system
;
;-------------------------------------------------------------
(defun reader ()
   (cond (keyboard
          (setq n (read-line)))
         (t
```

```
          (if (probe-file "/tmp/ti.dat") (shell " rm /tmp/ti.dat"))
          (shell "/micro/john/tispeech/read")
          (WITH-OPEN-FILE
             (voice "/tmp/ti.dat" :DIRECTION :INPUT)
             (setq p nil)
             (setq n " ")
             (setq o " ")
             (loop while (not (eq p 'endtrans))
             do
               (setq p (read voice))
               (setq n (concatenate 'string n o (format nil "~S" p))))
))))
;*****************************************************************************
;*                                                                           *
;*                            RESPONDER SUBMODULE                            *
;*Purpose: This submodule sends error and image completion messages to the   *
;*         TI Speech module and the Sun-4 computer screen                    *
;*                                                                           *
;*Major variables: msg - storage for the message to be transmitted           *
;*                 keyboard - a switch indicating the current communication *
;*                            mode                                           *
;*                                                                           *
;*****************************************************************************
;-------------------------------------------------------------
;
;    Writer Function
;    This function sends messages to the Speech module and screen
;
;-------------------------------------------------------------
(defun writer (msg)
;--------------- Format the message ------------------------
 (setq index (length msg))
  (setq counter 0)
  (loop while (> (- index counter) 0)
   do
    (if (eq #\_ (char msg counter))
        (setf (char msg counter) #\Space))
    (setq counter (+ counter 1)))
 (setq msg (string-downcase msg))
```

```
  (terpri )
;---------------- Print message to screen ------------------
  (format t "SYSTEM RESPONSE:  ~S" msg)
  (terpri )
  (terpri )
  (format t "YOUR NEXT STATEMENT:  ")
;---------------- Send message to Speech module ------------
(cond ((not keyboard)
       (WITH-OPEN-FILE
             (voice "/tmp/tmp.dat" :DIRECTION :OUTPUT)
             (princ msg voice)
             ( terpri voice))
          (shell "cp /tmp/tmp.dat /tmp/ti.dat")
          (shell "chmod 0777 /tmp/ti.dat /tmp/tmp.dat")
          (shell "/micro/john/tispeech/write"))))
```

```
;*****************************************************************************
;*                                                                           *
;*                      HELP, UTILITY, AND FOREIGN FILES                     *
;*                                                                           *
;*                                                                           *
;*****************************************************************************
(in-package 'user)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;    PARAMETER AND HELP SCREENS GENERATORS
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;-----------------------------------------------------------------------
;
;    Message Function
;    This function draws the parameter list screen with the current values posted
;
;-----------------------------------------------------------------------
(defun message()
  (shell "clear")
  (With-Open-File
  (AA "/dev/tty" :Direction :Output)
  (princ "                    ******************************" AA)
  (terpri AA)
  (princ "                    *                            *" AA)
  (terpri AA)
  (princ "                    *  SYSTEM  COMMAND  SCREEN   *" AA)
  (terpri AA)
  (princ "                    *                            *" AA)
  (terpri AA)
  (princ "                    ******************************" AA)
  (terpri AA)
  (terpri AA)
  (terpri AA)
  (terpri AA)
  (princ "PARAMETER      VALUE-RANGE         LAST-TRANSMITTED
  CURRENT-VALUE" AA)
  (terpri AA)
```

```
(setq temp1 d3)
(setq temp3 d2)
(setq temp2 (cdr temp1))
(setq temp4 (cdr temp3))
(terpri AA)
(princ " sagittal       MR: 1-61                    " AA)
(setq temp1 (cdr temp2))
(space (car temp2) AA)
(princ (car temp4) AA)
(setq temp3 (cdr temp4))
(terpri AA)
(princ " axial          MR: 1-256  CT: 1-18         " AA)
(setq temp2 (cdr temp1))
(space (car temp1) AA)
(princ (car temp3) AA)
(setq temp4 (cdr temp3) )
(terpri AA)
(princ " coronal        MR: 1-256                   " AA)
(setq temp1 (cdr temp2))
(space (car temp2) AA)
(princ (car temp4) AA)
(setq temp3 (cdr temp4))
(terpri AA)
(princ " maximum        1-255                       " AA)
(setq temp2 (cdr temp1))
(space (car temp1) AA)
(princ (car temp3) AA)
(setq temp4 (cdr temp3))
(terpri AA)
(princ " minimum        1-255                       " AA)
(setq temp1 (cdr temp2))
(space (car temp2) AA)
(princ (car temp4) AA)
(setq temp3 (cdr temp4))
(terpri AA)
(princ " distance       1-18000                     " AA)
(setq temp2 (cdr temp1))
(space (car temp1) AA)
(princ (car temp3) AA)
```

```
(setq temp4 (cdr temp3))
(terpri AA)
(princ " theta          1-360                    " AA)
(setq temp1 (cdr temp2))
(space (car temp2) AA)
(princ (car temp4) AA)
(setq temp3 (cdr temp4))
(terpri AA)
(princ " phi            1-180                    " AA)
(setq temp2 (cdr temp1))
(space (car temp1) AA)
(princ (car temp3) AA)
(setq temp4 (cdr temp3))
(terpri AA)
(princ " cscale         21-23                    " AA)
(setq temp1 (cdr temp2))
(space (car temp2) AA)
(princ (car temp4) AA)
(setq temp3 (cdr temp4))
(terpri AA)
(princ " background     31-33                    " AA)
(setq temp2 (cdr temp1))
(space (car temp1) AA)
(princ (car temp3) AA)
(setq temp4 (cdr temp3))
(terpri AA)
(princ " shade          51-53                    " AA)
(setq temp1 (cdr temp2))
(space (car temp2) AA)
(princ (car temp4) AA)
(setq temp3 (cdr temp4))
(terpri AA)
(terpri AA)
(princ "-------------------------------------------------------------------
 --------" AA)
(terpri AA)
(princ " PLAN :  " AA)
(princ p_name AA)
(terpri AA)
```

```
 (princ "--------------------------------------------------------------------
 --------" AA)
 (terpri AA)
 (princ " INPUT MODE :  " AA)
 (if keyboard
     (princ "KEYBOARD" AA)
     (princ "VOICE" AA))
 (terpri AA)
 (princ "--------------------------------------------------------------------
 --------" AA)
 (terpri AA)
 (terpri AA)
 (terpri AA)
 (princ "YOUR LAST STATEMENT:  " AA)
 (princ n AA)
 (terpri AA)))

(defun space (tvalue file)
  (princ tvalue file)
  (magnitude tvalue)
  (spacer file))

(defun magnitude (number)
  (if (> 0 number) (setq mag 2) (setq mag 1))
  (loop while (> number 9)
   do
     (setq number (/ number 10))
     (setq mag (+ mag 1))))

(defun spacer ( file)
  (setq spaces (- 20 mag))
  (loop while (> spaces 0)
   do
    (princ " " file)
    (setq spaces (- spaces 1))))

;-------------------------------------------------------------------------
;
;    Helper Function
```

```
;    This function determines which help screen to activate
;
;-------------------------------------------------------------------------
(defun Helper ()
   (shell "clear")
   (setq field (keyword))
   (cond
         ((eq field 'plan) (plan-help))
         ((eq field 'sag)  (sag-help))
         ((eq field 'axial)(axial-help))
         ((eq field 'cor)  (cor-help))
         ((eq field 'max)  (max-help))
         ((eq field 'min)  (min-help))
         ((eq field 'shade)(shade-help))
         ((eq field 'dist)    (x-help))
         ((eq field 'yaw)    (y-help))
         ((eq field 'pitch)    (z-help))
         ((eq field 'cscale)(cscale-help))
         ((eq field 'background) (background-help))
         ((eq field 'help) (general-help))
         ((eq field 'command) (command-help))
         (t (default-help))))

;-------------------------------------------------------------------------
;
;    Plan-Help Function
;    This function generates the plan help screen
;
;-------------------------------------------------------------------------
(defun plan-help ()
   (terpri)
   (terpri)
   (princ "                PLAN   PARAMETER ")
   (terpri)
   (princ "               ------------------ ")
   (terpri)
   (terpri)
   (princ "The plan parameter is used to indicate which plan is to be executed.
   ")
```

```
(terpri)
(princ " The following is a list of the valid plans: ")
(terpri)
(terpri)
(princ "        Catscan Tomography - 2D")
(terpri)
(princ "      2  - CT_Axial_Plan")
(terpri)
(princ "      3  - CT_Incremental_Axial_Slice_Plan ")
(terpri)
(terpri)
(princ "        Magnetic Resonance -2D ")
(terpri)
(princ "      4  - Sagittal_Slice_Plan ")
(terpri)
(princ "      5  - Axial_Slice_Plan ")
(terpri)
(princ "      6  - Coronal_Slice_Plan")
(terpri)
(princ "      7  - Incremental_Sagittal_Slice_Plan  ")
(terpri)
(princ "      8  - Incremental_Axial_Slice_Plan ")
(terpri)
(princ "      9  - Incremental_Coronal_Slice_Plan")
(terpri)
(princ "      10 - Tri_Coordinate_Slice_View_Plan")
(terpri)
(terpri)
(princ "        Pixel - 3D")
(terpri)
(princ "      11 - Pixel_imaging_Plan")
(terpri)
(terpri)
(princ "        Magnetic Resonance - 3D ")
(terpri)
(princ "      12 - Face_Generating_Plan ")
(terpri)
(princ "      13 - Face_Normal_Lighting_Plan ")
(terpri)
```

```
(princ "      14 - Head_Generating_Plan")
(terpri)
(princ "      15 - Head_Normal_Lighting_Plan ")
(terpri)
(terpri)
(princ "        SPECIAL NON-IMAGING PLANS")
(terpri)
(princ "      1  - Graphic Help Screen Generator")
(terpri)
(princ "      16 - Kill Graphics Process ")
(terpri)
(princ "      17 - Rotation_Demo_Plan ")
(terpri)
(terpri)
(princ " say CONTINUE to proceed or a plan number for more info about that
plan")
(writer " Say continue to proceed or a plan number for more info about that
plan")
(terpri)
(reader)
(cond ((numberp(setq num (read-from-string n)))
       (cond((eq num 1)
             (princ "Plan 1 generates a graphic help screen to aid the user")
             (terpri)
             (princ " Say CONTINUE to proceed")
             (writer " Say continue to proceed")
             (terpri))
            ((eq num 2)
             (princ "Plan 2 displays the axial slice indicated by the axial
              parameter")
             (terpri)
             (princ "Relevant significant parameters :")
             (terpri)
             (princ "     axial        values 1-18")
             (terpri)
             (princ "     max          values 1-255")
             (terpri)
             (princ "     min          values 1-255")
             (terpri)
```

```
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 3)
(princ "Plan 3 displays the 18 CT axial slices")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 4)
(princ "Plan 4 displays a specified sagittal (side) MR slice")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     sag          values 1-61")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
```

```
(writer " Say continue to proceed")
(terpri))
((eq num 5)
(princ "Plan 5 display a specified axial (horizontal) MR slice")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     axial          values 1-256")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 6)
(princ "Plan 6 display a specified coronal (vertical) MR slice")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     cor          values 1-256")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 7)
```

```
(princ "Plan 7 displays 18 sequential sagittal (side) MR slices
 beginning with the current slice")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     sag          values 1-61")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 8)
(princ "Plan 8 displays 18 sequential axial (horizontal) MR
 slices beginning with the current slice")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     axial          values 1-256")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 9)
(princ "Plan 9 displays 18 sequential coronal (vertical) MR
```

```
slices beginning with the current slice")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     cor          values 1-256")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 10)
(princ "Plan 10 displays current axial, sagittal, and coronal
 MR slices")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     sag          values 1-61")
(terpri)
(princ "     axial        values 1-255")
(terpri)
(princ "     cor          values 1-255")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
```

```
(terpri))
((eq num 11)
(princ "Plan 11 displays a 3 dimensional pixel image of the
 head")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     dist         values 1-18000")
(terpri)
(princ "     yaw          values 1-360")
(terpri)
(princ "     pitch        values 1-180")
(terpri)
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 12)
(princ "Plan 12 displays a 3 dimensional MR face image")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     dist         values 1-18000")
(terpri)
(princ "     yaw          values 1-360")
(terpri)
(princ "     pitch        values 1-180")
(terpri)
```

```
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 13)
(princ "Plan 13 displays a 3 dimensional normalized MR face
 image")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
(princ "     dist         values 1-18000")
(terpri)
(princ "     yaw          values 1-360")
(terpri)
(princ "     pitch        values 1-180")
(terpri)
(princ "     cscale       values 21-23")
(terpri)
(princ "     background   values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 14)
(princ "Plan 14 displays a 3 dimensional MR head image")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     max          values 1-255")
(terpri)
(princ "     min          values 1-255")
(terpri)
```

```
(princ "     dist          values 1-18000")
(terpri)
(princ "     yaw           values 1-360")
(terpri)
(princ "     pitch         values 1-180")
(terpri)
(princ "     cscale        values 21-23")
(terpri)
(princ "     background    values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 15)
(princ "Plan 15 displays a 3 dimensional normalize MR head
 image")
(terpri)
(princ "Relevant significant parameters :")
(terpri)
(princ "     max           values 1-255")
(terpri)
(princ "     min           values 1-255")
(terpri)
(princ "     dist          values 1-18000")
(terpri)
(princ "     yaw           values 1-360")
(terpri)
(princ "     pitch         values 1-180")
(terpri)
(princ "     cscale        values 21-23")
(terpri)
(princ "     background    values 31-33")
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(terpri))
((eq num 16)
(princ "Plan 16 commands the graphics process to terminate")
(terpri)
```

```
                (princ "There are no relevant significant parameters ")
                (terpri)
                (princ " Say CONTINUE to Proceed")
                (writer " Say continue to proceed")
                (terpri))
                ((eq num 17)
                (princ "Plan 17 executes a rotating head demo; you must say
                 HALT_DEMO to end the demo")
                (terpri)
                (princ "There are no relevant significant parameters ")
                (terpri)
                (princ " Say CONTINUE to Proceed")
                (writer " Say continue to proceed")
                (terpri)))))
    (if (numberp num)
        (reader)))

;---------------------------------------------------------------------
;
;   Sag-help Function
;   This function generate the sagittal help screen
;
;---------------------------------------------------------------------
(defun sag-help ()
   (terpri)
   (terpri)
   (princ "                SAG   PARAMETER ")
   (terpri)
   (princ "               ------------------ ")
   (terpri)
   (terpri)
   (princ " The sag parameter indicates which of 61 sagittal (side) slices  ")
   (terpri)
   (princ " is to be used to generate the image. It is a significant parameter")
   (terpri)
   (princ " when running 2D plans #4 ,#7, and #10. ")
   (terpri)
   (terpri)
   (terpri)
```

```
   (princ " Say CONTINUE to Proceed")
   (writer " Say continue to proceed")
   (reader))

;-------------------------------------------------------------------
;
;    Axial-help Function
;    This function generates the axial help screen
;
;-------------------------------------------------------------------
(defun axial-help ()
   (terpri)
   (terpri)
   (princ "                AXIAL   PARAMETER ")
   (terpri)
   (princ "               ------------------- ")
   (terpri)
   (terpri)
   (princ " The  parameter indicates which of 256 axial (horizontal) slices  ")
   (terpri)
   (princ " is to be used to generate the image. It is a significant parameter")
   (terpri)
   (princ " when running 2D plans #4,#7, and #9. ")
   (terpri)
   (terpri)
   (terpri)
   (princ " Enter any letter and RETURN to Continue ....")
   (terpri)
   (read))

;-------------------------------------------------------------------
;
;    Cor-help Function
;    This function generates the coronal help screen
;
;-------------------------------------------------------------------
(defun cor-help ()
   (terpri)
   (terpri)
```

```
   (princ "                CORONAL   PARAMETER ")
   (terpri)
   (princ "               ---------------------- ")
   (terpri)
   (terpri)
   (princ " The  parameter indicates which of  256 axial (horizontal) slices  ")
   (terpri)
   (princ " is to be used to generate the image. It is a significant parameter")
   (terpri)
   (princ " when running 2D plans #6,#9, and #10. ")
   (terpri)
   (terpri)
   (princ " Say CONTINUE to Proceed")
   (writer " Say continue to proceed")
   (reader))

;-------------------------------------------------------------------------
;
;    Max-help Function
;    This function generates the maximun parameter help screen
;
;-------------------------------------------------------------------------
(defun max-help ()
   (terpri)
   (terpri)
   (princ "                MAX   PARAMETER ")
   (terpri)
   (princ "               ------------------ ")
   (terpri)
   (terpri)
   (princ " The max parameter is used to set the upper threshold for the data
    that ")
   (terpri)
   (princ " is used to generate the image. Its maximum value is 256 and its  ")
   (terpri)
   (princ " minimum value is 1. It is a relevant parameter in all image
   generation ")
   (terpri)
   (princ "plans. ")
```

```
   (terpri)
   (terpri)
   (princ " Say CONTINUE to Proceed")
   (writer " Say continue to proceed")
   (reader))

;---------------------------------------------------------------------
;
;    Min-help Function
;    This function generates the minimum parameter help screen
;
;---------------------------------------------------------------------
(defun min-help ()
   (terpri)
   (terpri)
   (princ "                MIN   PARAMETER ")
   (terpri)
   (princ "               ------------------ ")
   (terpri)
   (terpri)
   (princ " The min parameter is used to set the lower threshold for the data
   that ")
   (terpri)
   (princ " is used to generate the image. Its maximum value is 256 and its  ")
   (terpri)
   (princ " minimum value is 1. It is a relevant parameter in all image
    generation ")
   (terpri)
   (princ "plans. ")
   (terpri)
   (terpri)
   (princ " Say CONTINUE to Proceed")
   (writer " Say continue to proceed")
   (reader))

;---------------------------------------------------------------------
;
;    Shade-help Function
;    This function generate the shade help screen
```

```
;
;-------------------------------------------------------------------
(defun shade-help ()
   (terpri)
   (terpri)
   (princ "                SHADE   PARAMETER ")
   (terpri)
   (princ "               ------------------ ")
   (terpri)
   (terpri)
   (princ " The shade parameter is used to determine which type of shading to
    use")
   (terpri)
   (princ " in generating a 3-D image. the following are valid choices: ")
   (terpri)
   (terpri)
   (princ "   51 - Wire shading ")
   (terpri)
   (princ "   52 - No shading")
   (terpri)
   (princ "   53 - Gouraud shading")
   (terpri)
   (princ "   54 - Phong diffuse shading")
   (terpri)
   (terpri)
   (princ " Shading is only a relevant parameter when running plans #11-#15.")
   (terpri)
   (terpri)
   (princ " Say CONTINUE to Proceed")
   (writer " Say continue to proceed")
   (reader))

;-------------------------------------------------------------------
;
;    Dist-help Function
;    This function generates the distance parameter help screen
;
;-------------------------------------------------------------------
(defun dist-help ()
```

```
   (terpri)
   (terpri)
   (princ "                DISTANCE   PARAMETER ")
   (terpri)
   (princ "               ---------------------- ")
   (terpri)
   (terpri)
   (princ " The DISTANCE parameter is used to select the perspective distance
   (radius) from a 3-D image on")
   (terpri)
   (princ " the display. Its valid values ranges from 1 to 18000 and is a
    relevant parameter for")
   (terpri)
   (princ " plans #10 thru #15. ")
   (terpri)
   (terpri)
   (princ " Say CONTINUE to Proceed")
   (writer " Say continue to proceed")
   (reader))

;-------------------------------------------------------------------------
;
;    Yaw-help Function
;    This function generates the yaw parameter help screen
;
;-------------------------------------------------------------------------
(defun yaw-help ()
   (terpri)
   (terpri)
   (princ "                YAW   PARAMETER ")
   (terpri)
   (princ "               ------------------ ")
   (terpri)
   (terpri)
   (princ " The YAW parameter is used to select the angle of tilt or "slant" of a 3-D image
on")
   (terpri)
   (princ " the Y- plane. Its valid values ranges from 1 to 360 degrees and is a relevant
parameter for ")
```

```
   (terpri)
   (princ " plans #10 and #11 - #15. ")
   (terpri)
   (terpri)
   (terpri)
   (princ " Enter any letter and RETURN to Continue ....")
   (terpri)
   (read))

;---------------------------------------------------------------------
;
;    Pitch-help Function
;    This function generates the pitch parameter help routine
;
;---------------------------------------------------------------------
(defun pitch-help ()
   (terpri)
   (terpri)
   (princ "                PITCH   PARAMETER ")
   (terpri)
   (princ "                ------------------ ")
   (terpri)
   (terpri)
   (princ " The PITCH parameter is used to select the rotation of a 3-D image
    on the Z- plane.")
   (terpri)
   (princ " Its valid values ranges from 1 to 180 degrees and is  a relevant
    parameter for ")
   (terpri)
   (princ " plans #10 thru #15. ")
   (terpri)
   (terpri)
   (princ " Say CONTINUE to Proceed")
   (writer " Say continue to proceed")
   (reader))
```

```
;---------------------------------------------------------------------
;
;    Cscale-help Function
;    This function generates the color scale help screen
;
;---------------------------------------------------------------------
(defun cscale-help ()
   (terpri)
   (terpri)
   (princ "                CSCALE   PARAMETER ")
   (terpri)
   (princ "               -------------------- ")
   (terpri)
   (terpri)
   (princ " The cscale parameter is used to select which color scale is to be
    used")
   (terpri)
   (princ " in generating an image. It is a relevant parameter in running all")
   (terpri)
   (princ " the imaging plans and may have the following valid values: ")
   (terpri)
   (terpri)
   (princ "   21 - Grey scale ")
   (terpri)
   (princ "   22 - Spectrum scale ")
   (terpri)
   (princ "   23 - Flesh scale ")
   (terpri)
   (terpri)
   (princ " Say CONTINUE to Proceed")
   (writer " Say continue to proceed")
   (reader))

;---------------------------------------------------------------------
;
;    Background-help Function
;    This function generates the background parameter help screen
;
;---------------------------------------------------------------------
```

```
(defun background-help ()
   (terpri)
   (terpri)
   (princ "                BACKGROUND   PARAMETER ")
   (terpri)
   (princ "               ------------------ ")
   (terpri)
   (terpri)
   (princ " The back parameter is used to set the background color and is a
    relevant")
   (terpri)
   (princ " parameter for running all the imaging plans. The following
    are the ")
   (terpri)
   (princ " valid values for this parameter: ")
   (terpri)
   (terpri)
   (princ "   31 - Gray background ")
   (terpri)
   (princ "   32 - Black background ")
   (terpri)
   (princ "   33 - Olive background ")
   (terpri)
   (terpri)
   (princ " Say CONTINUE to Proceed")
   (writer " Say continue to proceed")
   (reader))

;---------------------------------------------------------------------
;
;    General-help Function
;    This function generates a general help screen
;
;---------------------------------------------------------------------
(defun general-help ()
   (terpri)
   (terpri)
   (princ "                    GENERAL INFORMATION ")
   (terpri)
```

```
(princ "                    ---------- ")
(terpri)
(terpri)
(princ " The input to the graphics interface has the following
 general form :")
(terpri)
(terpri)
(princ "  ")
(terpri)
(terpri)
(princ "   [parameter-name parameter-value] modifies the value of
 a parameter.")
(terpri)
(terpri)
(princ "   [parameter-name help] generates a help file about that
 parameter.")
(terpri)
(terpri)
(princ " There may be 0 or multiple [...] before a command-key.
 The following")
(terpri)
(princ " command-keys are currently supported:")
(terpri)
(terpri)
(princ "   enter - displays current parameter/value pairs and last values
 transmitted")
(terpri)
(princ "   go - transmits the current parameter values list")
(terpri)
(princ "   stop - exits graphics interface mode. ")
(terpri)
(princ "   running plan 16 will disable the graphics output generator .")
(terpri)
(terpri)
(princ " Say CONTINUE to Proceed")
(writer " Say continue to proceed")
(reader))

;---------------------------------------------------------------
```

```
;
;    Default-help Screen
;    This function generate a warning screen for unknown help parameters
;
;---------------------------------------------------------------
(defun default-help ()
   (terpri)
   (terpri)
   (princ " ------------------------------------------------------------")
   (terpri)
   (terpri)
   (princ " WARNING : ")
   (princ field)
   (princ " is not a valid help option.")
   (terpri)
   (terpri)
   (princ " ------------------------------------------------------------")
   (terpri)
   (terpri)
   (princ " Say CONTINUE to Proceed")
   (writer " Say continue to proceed")
   (reader))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;    Utility Files
;
;Purpose: These functions build and access the plan objects for the AI module.
;         These plan objects contain the plans' relevant parameter value ranges.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;-----------------------------------------------------
;
;    Plan Flavors
;    These flavors are the frameworks for the plan objects
;
;-----------------------------------------------------
(defflavor plan
```

```
     (name  (signif plan))
     ()
     :settable-instance-variables)

(defflavor 2D-plan
     ((sag 61) (axial 255) (cor 255) (max 255) (min 255)
     (cscale-l 21) (cscale-u 24) (background-l 31) (background-u 32))
     (plan)
     :settable-instance-variables)

(defflavor CT-plan
     ((axial 18) (signif '(plan axial max min cscale background)))
     (2D-plan)
     :settable-instance-variables)

(defflavor 3D-plan
     ((dist 18000) (yaw 360) (pitch 180) (sag1cut 61) (sag2cut 61) (ax1cut 255)
      (ax2cut 255) (cor1cut 255) (cor2cut 255))
     (plan 2D-plan)
     :settable-instance-variables)

;-------------------------------------------------------------
;
;    Plan Instances
;    These instances build the plan objects
;
;-------------------------------------------------------------
(setq p11
    (make-instance 'plan
     :name 'Graphics_Help_Screen_Generator
     :signif '(plan)))

(setq p2
    (make-instance 'CT-plan
    :name 'CT_Axial
    :signif '(plan axial max min cscale background)))

(setq p3
    (make-instance 'CT-plan
```

```
    :name 'CT_Incremental_Axial_Slice
    :signif '(plan max min cscale background)))

(setq p4
    (make-instance '2D-plan
    :name 'Sagittal_Slice
    :signif '(plan sag max min cscale background)))

(setq p5
    (make-instance '2D-plan
    :name 'Axial_Slice
    :signif '(plan axial max min cscale background)))

(setq p6
    (make-instance '2D-plan
    :name 'Coronal_Slice
    :signif '(plan cor max min cscale background)))

(setq p7
    (make-instance '2D-plan
    :name 'Incremental_Sagittal_Slice
    :signif '(plan sag max min cscale background)))

(setq p8
    (make-instanc 7 154
Ce '2D-plan
    :name 'Incremental_Axial_Slice
    :signif '(plan axial max min cscale background)))

(setq p9
    (make-instance '2D-plan
    :name 'Incremental_Coronal_Slice
    :signif '(plan cor max min cscale)))

(setq p10
    (make-instance '2D-plan
    :name 'Tri_Coordinate_Slice_View
    :signif '(plan sag axial cor max min cscale background)))
```

```
(setq p12
    (make-instance '3D-plan
    :name 'Pixel_Imaging
    :signif '(plan max min dist yaw pitch cscale background)))

(setq p14
    (make-instance '3D-plan
    :name 'Face_Generator
    :signif '(plan max min dist yaw pitch cscale background )))

(setq p15
    (make-instance '3D-plan
    :name 'Head_Generator
    :signif '(plan max min dist yaw pitch cscale background
     sag1cut sag2cut ax1cut ax2cut cor1cut cor2cut)))

(setq p1
    (make-instance 'plan
    :name 'Kill_Graphics
    :signif '(plan)))

(setq p13
    (make-instance '3D-plan
    :name 'Volume
    :signif '(plan max min dist yaw pitch cscale background
     sag1cut sag2cut ax1cut ax2cut cor1cut cor2cut)))

(setq p99
    (make-instance 'plan
    :name 'Unknown_Plan
    :signif '(plan)))

;-------------------------------------------------------------
;
;    Binder Function
;    This function binds the parameters to their current associated values
;
;-------------------------------------------------------------
(defun binder ()
```

```
  (loop for i in d
  do
  (set (car i) (cadr i))))

;-------------------------------------------------------------
;
;    Name_Plan_Obj Function
;    This function binds the plan variable to the suitable plan object
;
;-------------------------------------------------------------
(defun Name_Plan_Obj ()
  (cond ((eq plan 1)
         (setq plan 'p1))
        ((eq plan 2)
         (setq plan 'p2))
        ((eq plan 3)
         (setq plan 'p3))
        ((eq plan 4)
         (setq plan 'p4))
        ((eq plan 5)
         (setq plan 'p5))
        ((eq plan 6)
         (setq plan 'p6))
        ((eq plan 7)
         (setq plan 'p7))
        ((eq plan 8)
         (setq plan 'p8))
        ((eq plan 9)
         (setq plan 'p9))
        ((eq plan 10)
         (setq plan 'p10))
        ((eq plan 11)
         (setq plan 'p11))
        ((eq plan 12)
         (setq plan 'p12))
        ((eq plan 13)
         (setq plan 'p13))
        ((eq plan 14)
         (setq plan 'p14))
```

```
        ((eq plan 15)
         (setq plan 'p15))
        (t
         (setq plan 'p99))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;                 Foreign Function
;
;Purpose: The foreign function written in C is called by the AI module
;         when it needs to send an interrupt to the Graphics module
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;------------------ Signaler interrupt generator -----------------------

#include <signal.h>
#include <rasterfile.h>

signaler()
{
  int pid,sig;

  pid = -9999;

  sig = SIGUSR1;

  kill(pid,sig);
}
```

VITA

Patrick Leroy Sherley was born in Memphis, Tennessee on July 27, 1961. He attended elementary schools in that city and Airways Junior High School. The author then attended Wooddale High School where he was voted "most likely to succeed" and graduated third in his class in May 1979. The following September he accepted a Cecil C. Humphrey's Presidential Scholarship and a National Achievement Scholarship and entered Memphis State University. He was a member of Alpha Lambda Delta, Gamma Beta Phi, and Tau Alpha Pi Honor Societies. He served as vice president of the student body for two terms and graduated cum laude in May 1983 with a Bachelor of Science degree in Engineering Technology. In the summer of 1988, he accepted a fellowship and began studies toward a Master's degree in Computer Science at The University of Tennessee Space Institute, Tullahoma. This degree was awarded in May 1990. The author currently holds membership in the ACM and IEEE professional societies.