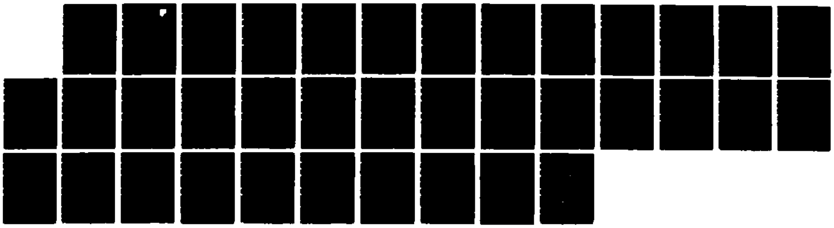
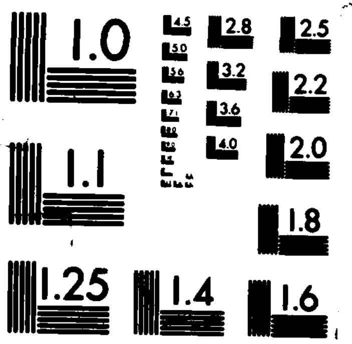
AD-A191 695 A PORTABLE NATURAL LANGUAGE INTERFACE(U) MITRE CORP 1/1
BEDFORD MA C E KALISH SEP 87 RADC-TR-87-155
F19628-86-C-0001

UNCLASSIFIED F/G 12/5 NL




1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

4

RADC-TR-87-155
Final Technical Report
September 1987

A PORTABLE NATURAL LANGUAGE INTERFACE

The MITRE Corporation

Candace E. Kalish

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

APPROVED: [illegible]

SHARON M. WALTER
Project Engineer

APPROVED: [illegible]

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Richard W. Pouliot

RICHARD W. POULIOT
Directorate of Plans & Programs

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188
1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS: N/A	
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A		
4. PERFORMING ORGANIZATION REPORT NUMBER(S): N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S): RADC-TR-87-155	
6a. NAME OF PERFORMING ORGANIZATION: The MITRE Corporation; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: Rome Air Development Center (COES)	
6c. ADDRESS (City, State, and ZIP Code): Burlington Road, Bedford MA 01730	7b. ADDRESS (City, State, and ZIP Code): Griffiss AFB NY 13441-5700	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Rome Air Development Center; 8b. OFFICE SYMBOL (If applicable): COES	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F19628-86-C-0001	
8c. ADDRESS (City, State, and ZIP Code): Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO 63728F; PROJECT NO. 2532; TASK NO 01; WORK UNIT ACCESSION NO. 09	
11. TITLE (Include Security Classification): A PORTABLE NATURAL LANGUAGE INTERFACE		
12. PERSONAL AUTHOR(S): Candace E. Kalish		
13a. TYPE OF REPORT: Final; 13b. TIME COVERED: FROM Oct 85 TO Oct 86	14. DATE OF REPORT (Year, Month, Day): September 1987	15. PAGE COUNT: 36
16. SUPPLEMENTARY NOTATION: N/A		
17. COSATI CODES: FIELD 12; GROUP 07; SUB-GROUP	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Artificial Intelligence; Automatic Programming; Expert Systems; Natural Language	

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

Project 5470, A Portable Natural Language Interface, was an outgrowth of the experience gained with the linguistic interface to the Knowledge-Based System (KNOBS) and the KNOBS Replanning System (KRS). The KNOBS/KRS interface provided multiple media including English and graphics, enabling users to easily communicate with the underlying mission planning system. Notable and very limiting deficiencies were however present in the interface. The most significant of these deficiencies were the lack of extensibility and portability. Application of KNOBS/KRS system to new domains and even simple enhancements to the vocabulary were inhibited by the complexity of the task. Integration of the interface to any other system was out of the question. The purpose of project 5470 was to create a natural language interface for expert systems that would be portable across different applications and would be capable of reasoning about user goals. This interface would integrate the use of graphics, menus and natural language within a new model and architecture that combined syntactic, semantic and discourse analysis in such a way as

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Sharon M. Walter	22b. TELEPHONE (Include Area Code): (315) 330-3564; 22c. OFFICE SYMBOL: RADC (COES)

DD Form 1473, JUN 86 — Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

UNCLASSIFIED

to promote generality and application independence. This report discusses the design and implementation of this portable interface and the degree of success attained in accomplishing the original goals.

UNCLASSIFIED

TABLE OF CONTENTS

FINAL REPORT FOR 5470, A PORTABLE NATURAL LANGUAGE INTERFACE

1. INTRODUCTION

The purpose of the 5470 project was to create a natural language interface for expert systems that would be portable across different backends, that would be capable of reasoning about user goals, and that would integrate graphics, mouse deixis, and natural language. Although the project was originally intended to last several years, it has been brought to a close after one year. Despite this, the 5470 staff has accomplished, at least in part, all three of the above goals.

1.1 PROGRAM RATIONALE

In order to use an expert system a user must communicate with it through some kind of interface. Designers of such interfaces have a wide variety of media they can choose to serve as the vehicle for such communication; they can provide graphical interfaces, menu driven interfaces, query languages and specialized command languages, natural language, or a combination of any or all of these media. Graphics can provide a rich, information dense vehicle of communication that is particularly useful in providing geographical information. Unfortunately, graphical means of communication have restricted expressive power; it is difficult to ask questions using pictures, for example, Menus are convenient, simple to use, and "safe" since they can keep a user out of trouble by restricting him to a given path, but they are also rigid, and they can be very slow since the user must often go through a long series of clicks and submenus before composing the command or question that he had in mind. They also are hard to use if the user wishes to do more than one thing in a single interaction. Query languages have great expressive power, but they require training and sharply limit the pool of users of an expert system.

Natural language, especially when combined with graphics and menus, is the ideal form of communication with most expert systems for several reasons. In the first place, the ability to use English or some other natural language enables all perspective users to issue commands and questions to a backend system regardless of training or experience. Natural language is also fast; there is no risk of wandering down the wrong path of a set of menus only to find that one has made a wrong choice somewhere and has to backtrack. A single sentence can also order a backend to do many things; one can say "Hit Dresden at 7:30 hours with 4 f-4cs from Hahn." Note that this command actually instructs the backend (KRS) to do five things: fill in the target slot, the homebase slot, the aircraft slot, the aircraft number slot, and the departure time slot. When speech understanding becomes a reality, linguistic communication will become so fast and so much moie natural than other forms of communication that users will insist that it be provided in most expert system interfaces.

Having said all this, however, one must acknowledge that as of 1986 many of the advantages of using natural language are undercut by the weaknesses and shortcomings of implementations. Not all of the problems involved in producing natural language interfaces have been solved; there are still gaps in both theory and in implementation. No natural language interface yet implemented has full linguistic coverage; all of them have various embarrassing (or worse) bugs, and each such program requires a user to get acquainted with its foibles. The promise of no learning period, therefore, exists in theory but has not been realized in practice.

What all this means is that research in natural language interfaces must continue. The problems must be solved soon because the need will become ever more pressing as more and more users wish to communicate with expert systems and as speech understanding comes closer to becoming a reality. If the problems are not solved, the difficulty of communicating with expert systems will become an obstacle to their increased use, and AI in general will suffer.

1.2 HISTORY

5470 was a successor to project 6070, which was itself a spinoff of the KNOBS project funded by RADC and developed at MITRE. 6070 produced a natural language interface for the KNOBS/KRS mission planning program, an expert system used to plan air attack missions for the Air Force. This interface combined English with graphics and mouse deixis and enabled KNOBS/KRS users to communicate easily with the backend. However, the interface had a number of deficiencies, some of which were so deeply embedded in the system architecture that correcting them was impossible.

Chief among these was a complete lack of portability. The 6070 interface was an implementation of conceptual dependency theory. All parsing was driven by a bottom up semantic analysis of the meaning of each word in the sentence, and the parser's job was to look up conceptual dependency definitions in its dictionary and attempt to match semantic expectations associated with each definition to what was actually present in the sentence. In order for the parser to be able to understand a sentence's meaning, the dictionary entries had to contain very large quantities of domain specific "clues" about the sort of structures expected to precede and follow the defined word. As a result of this, it was very difficult indeed to define new words, and there was virtually no generality in the parser that would carry over from one domain to another.

As the preceding statements suggest, extending the parser was no easy task either because of the difficulty in defining new words. In fact, the dictionary for the parser eventually reached a given size and then froze; even the most dedicated researchers were not able to figure out the necessary tricks needed to be able to add new definitions.

Another serious deficiency, which was not related to the choice of theoretical model but rather to a wrong turn in the development of the system, was an inability to understand user intentions and goals. The interface designers originally hoped that the interface would be capable of producing intelligent responses of the form illustrated by the following dialogue:

User: What is the range of an f–4c?

KRS: 600 miles.

User: How far is Hahn from the target?

KRS: 200 miles. This is within the range of an f–4c at Hahn.

In answering the question, the interface assumes that the user's intention in asking about the distance between the friendly airbase Hahn and the target was to find out if the target were in range of the attack plane, f–4c.

The 6070 developers hoped to represent user intentions by instantiating a script representing the actions involved in carrying out an air attack. Unfortunately, they were never able to match the user inputs with scenes in the script; this process, a form of plan recognition, is difficult, slow, and prone to error. As a result, the use of scripts to guide goal recognition never worked in 6070.

Because of these various problems, the developers of 5470 decided to turn to a new theoretical model, a new architecture, and a new approach to scripts. They developed a three part system that combined syntactic analysis, a case frame based semantic analysis, and a discourse analysis. This system was designed to have as much backend independence and thus generality as possible. It was also designed to use discourse principles to capture information about context an to enable the production of intelligent responses to user questions.

2. SYSTEM ARCHITECTURE

The system can be divided into three parts: the scenes and scene controller, the syntactic parser, and the semantic and relational dictionary. Each of these components is in charge of a specific function. The scenes are used to handle discourse phenomena and to keep track of context. The parser performs a syntactic analysis (and applies some semantic processing as well), and the vocabulary component takes care of figuring out the meaning of the input question or command. The three components were designed to work together rather than strictly sequentially because no part of linguistic analysis can be completely separated from any other part. In other words, one cannot understand the meaning of a word without considering the context in which it was uttered as well as the structure of the phrase in which it appears.

2.1 THE THEORETICAL BACKGROUND

Our work in this project was influenced by several linguistic theories, most notably, the theory of relational grammar developed by Perlmutter and his associates, and the theory of discourse developed by Barbara Grosz and Candace Sidner. The discussions of the various components of the interface below will frequently refer to these researchers and their work.

2.2 THE SCENES

One of the goals of this interface is to provide the user with intelligent responses to user input. To respond intelligently to a query, the system should provide relevant data that was not specifically requested. Similarly, an intelligent response to a command should execute all appropriate expert system operations without requiring the user to explicitly mention each. An interface that demonstrates this behavior must be able to capture the underlying intention of an utterance. Grosz and Sidner (1985) claim that any discourse has three main constituents: 1) the structure of the actual sequence of discourse utterances; 2) a structure of intentions; 3) an attentional state. We developed the scene mechanism in order to capture the latter two constituents of a discourse and use them to provide intelligent responses. Specifically, we claim that for restricted, well-defined domains (e.g. specialized expert systems), a library of structures can be compiled and utilized to provide intelligent responses. This is achieved by tracking the intentional and attentional states of a user's interaction with an expert system.

2.2.1 Intention and Attention

Grosz and Sidner distinguish between the intentional state of discourse and the attentional state. Intentional structure captures the purposes behind any utterance. These purposes shape coherent discourse. Utterances are causally related by associating them with intentional states. Attentional state captures the focus of attention in the discourse at any one moment. It is a dynamic structure that records the salient objects and relationships.

2.2.2 Intentional Structure

Discourses can be partitioned into segments, each representing some purpose or intention. These discourse segment purposes (DSP's) can be related in special ways. Grosz and Sidner (1985) concentrate on two kinds of DSP relationships: dominance and satisfaction-precedence. The satisfaction-precedence relation represents one DSP being a prerequisite of another. The dominance relation states that satisfying one DSP contributes to the satisfaction of another. This relation establishes a hierarchical structure of DSP's representing their dependencies. According to the type of discourse (eg. general conversation vs. task-oriented dialogue), the dominance relation can be more precisely defined. An expert system interface is primarily concerned with the computer accomplishing tasks. Therefore, the DSP dominance relation, in an expert system interface, adopts the rule for a task-oriented dialogue. This rule is presented here in the guise of an expert system-human *interaction:*

$$\forall x\ i = 1, \ldots, n\ [\ \text{Intend}(\text{ human}, \text{Intend}(\text{ computer}, \text{Do}(A)))\ \wedge$$
$$\text{Intend}(\text{ human}, \text{Intend}(\text{ computer}, \text{Do}(a_i)))\ \wedge$$
$$\text{Believe}(\text{ human}, \text{Generates}(\ A, a_1, a_2, \ldots, a_n))]$$
$$\Leftrightarrow$$
$$\text{Dominates}(\ \text{Intend}(\text{ human}, \text{Intend}(\text{ computer}, \text{Do}(A)))$$
$$\text{Intend}(\text{ human}, \text{Intend}(\text{ computer}, \text{Do}(a_i))))$$

A general interpretation of the above is: If the user intends that the expert system needs to accomplish tasks A and a_i, and the human believes that the performance of task a_i contributes to task A, then the intention concerning task A dominates the intention of task a_i.

2.2.3 Attentional Structure

The attentional structure captures the focus of attention in a discourse. It represents the prominent objects and relationships that are dynamically encountered in conversation. The attentional state is modeled by a set of focus spaces and rules for transitioning among them. Focus spaces also contain the DSP they are associated with. Abstractly, a focus space represents *what* the discourse participant is talking about, in addition to *why* he is talking about those things.

2.3 DATA STRUCTURE - SCENES

Scenes are data structures that represent the intentional and attentional states of discourse. They are schema representations (Minsky75, Bobrow75, Schank77) of stereotypical interactions with the expert system. An interaction is represented as a set of intended actions for the expert system to accomplish. The dependencies of these actions are represented by the hierarchical structure of scenes. This hierarchy is determined by the dominance relation defined for intentional structure. For example, consider scenes A and B. Assume scene A is superior to scene B. Then the set of actions represented by scene B contribute to the satisfaction of the actions represented by scene A. The root of a scene hierarchy represents the overall discourse purpose (DP) of the dialogue.

In addition to intentional structure, scenes also set up the attentional structure. This is accomplished by defining the kinds of objects that would be prominent if a scene were active in some discourse. These object classes are called the *roles* of a scene. They are used to determine whether a proposed scene is suitable. This filtering process will be fully described later.

When a scene is appropriate (i.e. recognized as the current intentional state), it is instantiated. At this point the scene represents attentional state. Its roles are filled with the referents of the objects in the current clause and other objects already present in the discourse. The preceding current scene is linked to the new one, maintaining a network of scenes modeling the discourse. An instantiated scene holds the salient objects (i.e. role-fillers) of the discourse, the current intentional context and a model of the conversational flow.

The following figure presents the salient fields that an instantiated scene possesses:

Scenes

The Intentional Structure

Field	**Description**
Name	*The type of scene.*
Roles	*The prominent object classes.*
Inferiors	*The scenes that this one dominates.*
Superior	*The scene dominating this one.*
Precedes	*The post-requisite scenes.*

Preceded-by	*The pre-requisite scenes.*
Triggers	*The lexical items that are recognized for this scene and their filtering maps.*

The Attentional Structure

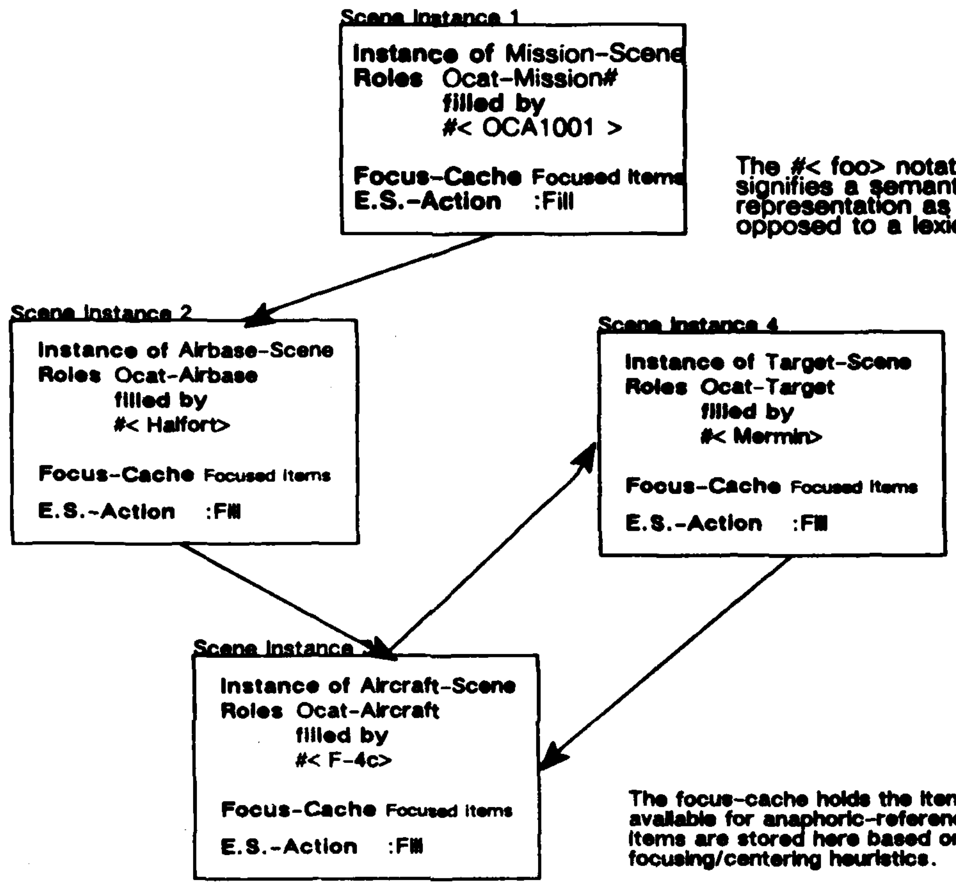
Field	**Description**
Role Fillers	*The semantic representations of those objects filling the roles.*
Predecessors	*The scenes preceding this one in the actual discourse.*
Successors	*The scenes following this one.*
Focus Cache	*The objects available for anaphoric reference*
Expert System Goal	*The abstract expert system actions to apply*

An important distinction must be made between a plan and an intentional scene hierarchy. The scenes represent **stereotypical interactions** with an expert system. However, they **do not** represent the sequence of actions an expert system will take. This information is captured in the expert system's plans goals and problem-solving strategies. Only data relevant to the user-machine interface is captured by the scene hierarchy.

To clarify the exposition of a scene hierarchy, the following figures present examples of scenes from the KRS mission planning application. The primary goal of this application is to plan an OCA mission task In order to complete the plan, among other things, a target, an airbase, and a type of aircraft must be chosen. The following scenes represent the interactions of a user planning a mission (as opposed to the system). The first figure presents the intentional structure of the scenes while the second illustrates the attentional state of a specific interaction. Here is the discourse that transpired:

1. Build a mission
2. Leave from Halfort.
3. Send F-4cs.
4. Make Mermin the target.
5. What is the range of an F-4c.

Attentional Structure - The Network of Predecessor/Successor Links



The #< foo> notation signifies a semantic representation as opposed to a lexical

The focus-cache holds the items available for anaphoric-reference. Items are stored here based on focusing/centering heuristics.

By capturing both intentional and attentional state, our expert system interface demonstrates the ability to derive intelligent responses. Intentional structure (i.e. scene hierarchies) allow the response handler to perform limited plan recognition (Allen 1980). Attentional structure provides the dynamic information encountered in conversation. Scenes also facilitate the handling of interruptions as well as focusing for referring expressions. The next section describes how the attentional state is maintained (i.e. how to choose the current scene). Then the recognition and handling of interruptions is described. Subsequently, a demonstration of the system is shown, followed by an in-depth look at the response mechanism which draws inferences based on intentional and attentional structure.

2.4 SCENE RECOGNITION

Scene recognition is the process of finding the currently active scene. At any point in the discourse, there is only one current scene. Recognition of attentional state is failure driven. Hence, a new scene is found if the response mechanism is unable to derive a response in the current scene. The response handler can either tell the scene controller what scene to transfer to, or it can instruct the scene controller to use its discourse heuristics to find the new attentional state.

2.4.1 General Control Flow

The scene heuristics utilize both intentional and attentional structure to resolve a response failure. Using these heuristics a scene is proposed. Then the current input clause is tested against the lexical triggers of the scene. These triggers define a mapping between the head verb of the sentence and its arguments, to the roles of the scene. If the role filter is successful, the proposed scene becomes the new current scene and is inserted into the predecessor/successor network. If the heuristics fail to provide a current scene, the user is asked what context his question pertained to (ie. which scene is appropriate) and then requested to re-phrase his input. More research is necessary to provide intelligent failure mechanisms with the ability to learn new scene triggers. In summary, the scene recognition process is a generate and test process where the scene heuristics guide the generation of possibilities and the role filter tests those possibilities.

2.4.2 Scene Heuristics

The following scene heuristics are currently driving the scene controller:

1. Check intentional structure.
2. Go back in the predecessor/successor network.
3. Lexical Triggering.

4. Exhaustively search the scene directory.
5. Ask the user.

1. Check intentional structure - The scene controller actually has a few heuristics that are based on intentional structure:

superior - Check the parent scene (i.e. the more abstract task).
inferiors - Check the child scenes (i.e. the more specific tasks).
siblings - Check the scenes on the same intentional level.
all-relatives - Check all the scenes that share the same overall DSP.
precedes - Check all scenes that this satisfaction-precedes.
preceded-by - Check all scenes that satisfaction-precedes this.

An interesting analogy can be made with the first two heuristics above and some of Allen's planning rules (Allen 1980). Informally, two of his heuristics are:
1. If one wants to perform a task A, one might want to perform the tasks satisfying the preconditions of A. (Action-Precondition Rule).

2. If an agent wants to perform some task that satisfys a precondition of task A, then one could infer that the agent really wants to perform task A. (Precondition-Action Rule).

If one draws an analogy between the intentional relationships of scenes (i.e. dominance and satisfaction-precedence) with Allen's notion of a plan's preconditions, the superior and inferior heuristics encompass Allen's planning heuristics.

2. Go back in the predecessor/successor network - There are two heuristics for traversing back in the predecessor chain.

predecessor - Try the last n scenes accessed.
predecessors-tree - Try the last n scene *trees* accessed.

Check the scene lexicon

lexical-triggers - Try all the scenes that are triggered by the main verb of the sentence.

Exhaustive Search

Other-Contexts - Try all the scenes in defined.

Ask the user
Ask-user – Query the user for advice.

In the interest of flexibility, the order and collection of heuristics are highly declarative so that new heuristics can be added and the order of their application can be updated without major revisions.

2.4.3 Scene Filter

The scene filter tests the possibilities generated by the heuristics above. The scene filter is a role consistency check. For each verb defined in a scene, there is a mapping from the case frame representation of the parse to the roles of the scene. This mapping stipulates where to extract a role filler from the sentence. The semantic arguments of the verb are found in a case frame, while modifiers of the case fillers take a relational representation. Scene roles can be filled by both semantic arguments as well as their modifiers. When mapping from a modifier to a scene role, the following information is necessary:

1. Semantic Argument – The case role that is being modified.
2. Relation – The type of modifier. (e.g., time, space-destination, possession).
3. Arguments – The arguments of the relation where the fillers can be found.
4. Scene Role – The scene role to fill.

Simple mappings do not involve modifiers. They need the following information:

1. Semantic Argument.
2. Scene Role.

These expectation mappings are stored along side the verbs on the triggers list of a scene.

To summarize, the role checking process first finds the head verb of the clause. Then the expected role fillers are extracted from the semantic arguments (and their modifiers) of the case frame. Let #<f-4c> be the the referent of the noun phrase that is found in the position specified by the mapping. If #<f-4c> can play the scene role specified in the mapping (e.g. ocat-aircraft), then the role is consistently filled. If #<f-4c> were a referent that was inconsistent with the specified role, the current scene would be rejected. If the semantic argument were empty, the role would still be considered consistent (i.e. not a cause a rejection).

Taking a step back, role checking is a pattern match on the expected semantic interpretation of a verb and its arguments. When the expectations of a scene are correct, the role fillers are stored in the scene. Inconsistent role fillers lead to scene failure.

To recapitulate, in the event of a response failure, a new scene must be found. The scene recognition process, which achieves this, is a generate and test algorithm. Possibilities are generated through the application of heuristics which utilize the intentional and attentional states of the discourse. These possibilities are tested by applying the role filtering procedure. This procedure compares the verb and its arguments in the current clause with the trigger maps defined for a scene. The possibility is only accepted if a consistent mapping is found.

In the following examples, the first four columns of the chart represent the mappings found in the triggers field of a scene indexed by the verb they are associated with (i.e. send and leave). The fifth column is the referent of the object specified in the map. The first two examples are successful because the referent can be classified as the scene role specified. However, the third clause would result in a scene failure because a Kc-135 is not a valid ocat-aircraft (its refueling plane) and Hartfort is a friendly airbase.

1. Send an F-4c to Mermin

Semantic Role	Relation	Arguments	Scene Role	Referent
Object			Ocat-Aircraft	#<F-4c>
Event	Space-Destination	(position)	Ocat-Target	#<Mermin>

2. Leave Halfort at noon.

Semantic Role	Relation	Arguments	Scene Role	Referent
Object			Ocat-Airbase	#<Halfort>
Event	Time-Destination	(position)	Ocat-Departure	#<noon>

3. Send a KC-135 to Halfort.

Semantic Role	Relation	Arguments	Scene Role	Referent
Object			Ocat-Aircraft	#<Kc-135>
Event	Space-Destination	(position)	Ocat-Target	#<Halfort>

The event semantic role means the modifier is attatched to the entire case frame.
For an explanation of the relational modifiers see Bayer (1986).

2.4.4 Implementation

The entire interface is implemented in Zetalisp on Symbolics lisp machines. Scene instances are represented as flavor instances. Each scene has instance variables for the fields presented and methods for executing the heuristics. To apply one of the heuristics, the name of the heuristic is sent as a message to the current scene. This will return a list of proposed scenes. Scene recognition is handled by the scene controller which is also implemented as a flavor instance in order to retain dynamic information and also to facilitate integration with the expert system's i/o loop. The overall control structure takes the following form. The scene controller is sent a :process-input message which takes the following steps:

1. Accept a clause.
2. Parse the clause.
3. Perform semantic analysis.
4. Try to fill roles from the current clause.
5. Call the response handler.
6. Upon success go to step 1.
7. Otherwise, send the scene controller a :next-possibility message

which performs the generate and test algorithm presented previously.
8. Go to step 5.

2.5 INTERRUPTIONS

Grosz and Sidner have categorized the different type of interruptions occurring in discourse and define the following kinds :

1. True Interruptions.
2. Flashbacks.
3. Digressions.
4. Semantic Returns.

True interruptions are sudden jumps to utterances which share no intentions with the interrupted segment. For example, "what do you want for dinner ?", "Wait a minute son, take this trash out also !", ... "manicotti or moo shu chicken." The command to take out the garbage is obviously completely unrelated to the discussion about dinner.

Flashbacks are jumps to relevant parts of the conversation that need to be filled in and probably signify a satisfaction-precedence violation. For example, "Install the experiment in the casing." "Oops, first unpack the experiment." The "Oops" and "first" designate the flashback. Unpacking an experiment is a pre-requisite to installing it. The speaker realizes this and performs the flashback.

Digressions are interruptions in which the interrupted segment shares a salient object with the interruption. For example "Send an f-4c to Halfort," "By the way, has the F-4c maintenance crew arrived yet." In this case the maintenance schedule and a mission planning choice are disjoint DSP's but they do share the F-4c as a prominent object.

The last type of interruptions, called semantic returns, occurs when discourses from the past are explicitly re-introduced. For example, "Remember, the discussion of the airbase," "Why was Halfort inappropriate?" The conversation is recalled and the discourse is continued.

2.5.1 Stack-based Model

Grosz and Sidner utilize a stack-based model to manage the focus spaces. This facilitates interruption handling and the correct resolution of referring expressions. The scene mechanism achieves similar behavior by marking scenes in the predecessor/successor network. We suspect this method is more appropriate for expert system interactions because it allows the system to retrieve a specific attentional state for both interruptions and normal focus space movements. This will be discussed further when the processing of semantic returns is discussed. In the following paragraphs, the management of each interruption type will be presented. In some cases, this process will be compared to the stack-based model.

2.5.2 Clue Words

Another difference between the scene mechanism and the Grosz and Sidner model is the emphasis on clue words. We have deemphasized the use of clue words because they are less prevalent in user-expert system interactions. This is simply due to the fact that users tend to type in the actions they want accomplished. They are not concerned with the machine understanding why they are requesting some information or action which is characteristic of human-human spoken discourse. Therefore, the scene mechanism attempts to recognize the type of interruption using the scene heuristic that lead to the interruption instead of recognizing clue words. Ideally, both methods should be used, however, we have not yet implemented any clue word handler.

2.5.3 True Interruptions

Handling a true interruption requires focusing the prominent objects of the interruption but then forgetting those objects when the interrupted segment is returned to. In the stack model this is accomplished by pushing the interruption on to the stack and then popping the stack which leaves the interrupted segment at the top of the stack. In the scene mechanism the predecessor/successor links are annotated with flags representing the interruption barriers. The crucial point is that the interruption can be recognized when traversing back through the chain and therefore skipped if necessary. This would exhibit the same behavior as the stack model without enforcing the strict control structure exhibited by stacks.

In the implementation, true interruptions are recognized when the "other contexts" or "lexical-triggers" heuristic is successful. More specifically , when the attentional state encountered differs in the overall discourse purpose (DP) from the interrupted segment (i.e. they are part of different scene trees), the movement is flagged as an interruption. Later, when the "predecessor" or "predecessor-tree" heuristics return the attentional state to the interrupted segment, the interruption barrier is flagged. Hence, the beginning and end of the interruptions are flagged in the predecessor/successor chain. See the interruption example in the sample interaction.

2.5.4 Flashbacks

To date, flashbacks have not been handled by our implementation. However, the following presents a proposed strategy which is very similar to the handling of interruptions. They differ in that the flashback is realized by the "enabled-by" scene heuristic and, the beginning and end of the interruption is marked as a flashback. Grosz and Sidner present three plausible models of flashbacks which will not be deeply explained here. They are handling the flashback as normal focus space stacking, or handling the flashback like an interruption, or using an auxiliary stack. However, it should be noted that by marking the flashback boundaries, one can process it using any of their strategies.

2.5.5 Digressions

Digression handling is trivial in the implementation. The new attentional state is found using the scene heuristics and the role checking filter. After a new scene is found its roles are filled accordingly. The role-filler in the interrupted segment could be the same referent as the interruption's role filler. This is how the digression shares the prominent object from the old scene. Further the name of the role is probably different in both scenes since the different focus spaces represent different intentions. Grosz and Sidner handle digressions as interruptions which have prominent objects in common which is precisely the scene mechanism's strategy.

2.5.6 Semantic Returns

Especially in expert systems that solve planning problems, users often return to an attentional state that was part of an interruption. In other words, users describe objects in the retrieved attentional state with referring expressions that need the focussed objects of the old attentional state.

2.5.7 Focus Space History

The use of a planning system as a backend forced us to handle discourses about hypothetical situations. A hypothetical situation is one in which the attentional state changes but the intentional state does not. More specifically, the relevant objects change but they serve the same purpose. For example, in the aircraft scene, an f-111e is introduced. Then later in the discourse, the user inquires about the speed of an f-4c. There are basically two strategies one can take in this situation:

1. Retain the old focus-space but replace the f-111e with the f-4c.
2. Build a new focus space for the f-4c and save the old focus space away.

Only the second strategy retains enough information to allow a reference to the old state of affairs:

1. Use the old choice for the aircraft.
2. Remember the f-111e.

3. Remember when the aircraft was an f-111e.
4. What aircraft have been discussed.

In the first example a new focus space with the old choice would be inserted into the current intentional hierarchy and perform the :change expert system action. The second example is identical less the expert system action. The third example creates a completely new focus hierarchy representing the state of the discourse the last time an f-111e was relevant. The last example would traverse the focus space history and retrieve the appropriate role fillers. All of this behavior utilizes the focus history which is maintained during scene changes. When a role is about to be overwritten, a snapshot of the current state is taken (i.e. the complete list of active role fillers) and pushed onto the current history. Then a new focus space is created (for the new filler) and the history is stored in the new scene.

3. THE PARSER

Two strategies for parsing natural language dominate the research scene; one strategy relies most heavily on syntax, the other on semantics. The syntactically based schemes usually contain a phrase structure grammar for the target language along with a series of syntactic rules to handle structural transformations. The semantic parsers often make use both of local, word-based meaning representations like CD's and more global representations such as scripts, plans, or tau's. We have combined these approaches through the use of relational grammar, a theory of language which ties syntax and semantics together through the medium of relational categories such as subject and direct object. Our approach uses both a formal grammar for the syntax of English and also CD-like structures (case frames) to represent word meanings, but the case frame slots are filled through the application of mapping rules based on grammatical relations, which associate semantic categories like "actor" with relational categories like "subject." This hybrid approach enables us to overcome many of the weaknesses associated with other parsing strategies. A brief sketch of some recent work in natural language parsing and text understanding will provide some context.

During the 1970's, Woods, and Bresnan and Kaplan among others developed syntax driven parsers based on various kinds of augmented transition networks. The theoretical linguistic base for these parsers was provided by theories of syntax which to a greater or lesser extent grew out of transformational grammar. This is perhaps more true of Woods's system than of Bresnan and Kaplan's parser since the latter relied on an unusually rich lexical component which provided information about subcategorization and semantic slot filling in addition to a syntactic component. A later approach to syntax driven parsing is represented by Marcus's work, which, unlike ATN's, does no backtracking and builds permanent structure through manipulation of a (basically) 3-position buffer. These three parsers, which are only a subset of the many syntax-based attempts to deal with natural language, share some of the same flaws as well as the same strengths. One of the principal difficulties with the type of parser represented by Woods's work and Marcus's work, and also to a lesser extent of Bresnan's and Kaplan's, is illustrated by the problems associated with garden path sentences. In such sentences, an example of which is "The horse raced past the barn fell," the early part of the sentence lures the reader/listener down a false path of understanding. Somewhere in the middle of the sentence information appears which indicates that the parse up to now is incorrect and must be replaced: in other words, one must backtrack. Backtracking is expensive in ATN directed parsers; as we noted above, in Marcus's parser it is impossible. One excuse that developers of this type of parser offer is that since human beings have so much trouble with garden paths it is not unreasonable that automatic parsers confronted by such sentences should come to grief as well.

Prepositional phrase attachment represents another problem for syntactic parsers since this phenomenon is primarily semantic in nature and as such does not lend itself to syntactic solutions. More abstractly, one can say that syntactic parsers generally have trouble dealing with sentences with multiple interpretations, where the ambiguity involves the placement of constituents. Marcus uses a related problem, that of locating the source of a moved WH-phrase, to argue that semantic as well as syntactic information is necessary for an accurate syntactic parse and appeals to Woods' (1973) procedure of Selective Modifier Placement, although he does not formally incorporate it.

One syntactic parser, the CHART parser, developed by Kay for the MIND system, dealt extensively with this type of problem. The solution chosen was to keep several possibilities open at once by building constituents and, in effect, treating them as building blocks which could be put together in a variety of ways. The final parse would be the result not only of the identification and analysis of the constituents but also of the choice of how to put them together. The CHART parser relied on some semantic knowledge to make this choice and was in a way a hybrid of syntactic and semantic approaches to parsing.

Another approach to ambiguity is offered by members of the semantic school of parsing, among them Schank, Abelson, and their students. The developers of such semantic parsers as MARGIE, SAM, PAM, and BORIS concentrated on the problem of disambiguation by recognizing and keeping multiple senses from the start. Typically, these parsers reduced the array of multiple word senses and their consequent inferences to a "path" of consistent senses that represented the meaning of the input sentence. Attachment problems were resolved either by reference to larger meaning structures such as a scripts or by lexical expectations. SAM, PAM, and BORIS used other structures to represent information about plans and goals as well as the kind of stereotyped behavior captured by a script.

The ability to handle ambiguity is the great advantage of semantically oriented parsers, but this ability comes at a high price. Not only are syntactic generalizations lost, a serious theoretical liability, but, even more important, the need to keep track of many word senses and inferences leads to a serious implementation problem involving heavy time and space costs. These problems proved to be particularly intractable for MARGIE, SAM, and PAM, in effect, crippling them as true text understanding programs.

The 5470 parser differs from most parsers in that it recognizes not two but three levels of representation: the level of structure, the level of meaning, and the level of grammatical relations (subject, object, indirect object), which mediates between the first two for the purpose of identification of semantic roles. This notion is based mainly on work in Bayer (1984), although a similar approach in a different framework has been suggested in Wasow (1978). The premise of this approach is that there is no straightforward mapping between structural information (which, in typical CD approaches, seems to amount to no more than appeals to position) and semantic roles. This is clear even in English, a language whose structure is quite rigid and yields more clues than most languages about the mapping between structure and semantic roles.

An approach in terms of grammatical relations may be justified by i) simplification of syntactic generalizations, ii) simplification of identification of semantic roles, and iii) generality with respect to complex sentences.

In English, as in other languages, various syntactic facts are best expressed in terms of grammatical relations. The facts of verb agreement, for example, are most elegantly captured through appeals to the categories subject, object, and indirect object (in English as well as languages with richer case marking). Semantic subcategorization facts, such as the requirement that the verb "walk" be predicated of an animate being, are also most easily expressed in terms of grammatical relations. Since final grammatical relations are deduced from surface structure almost solely by positional and morphological information, one might claim that a statement of the facts of verb agreement and subcategorization in terms of this sort of information would be adequate. While it is true that this sort of description can be made, its awkwardness calls its intuitiveness and usefulness into question. The term "subject" in English is isomorphic with the phrase "the np directly preceding the verb in an uninverted clause, or the np directly following the initial verb in an inverted clause," but the fact that this disjunction must be employed in all those references where the word "subject" would be naturally used suggests that an important generalization is being missed. And this is in one of the best possible situations, where structure is strict and fairly unambiguous. In a language where case-marking and word-order combine to identify the subject, description of the above phenomena in structural and morphological terms becomes much harder.

Analogously, the explicit identification of subject aids in the identification of semantic roles. Generalizations or defaults, such as the mapping into the ACTOR slot, can be greatly simplified by referring to the notion of subject rather than to the positions the np in question may occupy. The mapping properties of classes of exceptions can also be described easily, when the group of np's conveniently labeled "subject" map into the OBJECT slot instead, for example.

We referred earlier to the notion of final grammatical relations. This phrase hints at the idea, central to a relational approach to syntax, that grammatical relations may CHANGE. This is how, in the passive sentence "John was struck by Mary," it is the surface subject which gets mapped into the OBJECT slot instead of the surface direct object in the corresponding declarative sentence "Mary struck John" (violent girl, that Mary). The operation of Passive, on this view, is that of a direct object becoming a subject, with concomitant displacement of the original subject (into the "by"-phrase in English, for example). In order to identify the original relations, we apply the operation backward in order to "undo" the application of Passive. Although English has relatively few rules which change grammatical relations, these few rules interact to derive complicated multi-clause sentences which positional approaches are hard-pressed to analyze elegantly or easily.

Consider two more of these relation-changing rules: Subject-to-Subject Raising, which makes the subject of a clause in subject position the subject of the dominating clause, relating "That John will go is likely" and "John is likely to go," and Subject-to-Object Raising, which makes the subject of a clause in direct object position the direct object of the dominating clause, relating "I believe that John left" and "I believe John to have left." These two rules, combined with the rule of Passive above, may cooperate in their application to yield quite complex sentences. For example,

i) John is believed by Mary to be likely to have left.

is derived by Subject–to–Subject Raising in the most embedded clause, followed by Subject–to–Object Raising in the next clause up, and finally Passive in the matrix clause. While a relational approach can, having identified and extracted those np's which bear the relevant grammatical relations, simply change the grammatical relations of the np's involved when they undo these operations, a positional/structural approach must physically move the np's or try to develop a set of complicated conditions which alter the slot–mappings for a verb. Both these approaches are, in our view, unwieldy, an uninsightful alternative to the relational approach to these operations; the intuitive appeal of grammatical relations is demonstrated even in the names of the raising operations just described, names which were coined not by relational grammarians but by the classical transformational grammarians of the '50's and '60's, notably Noam Chomsky, for whom grammatical relations were (and still are) derivative.

3.1 PARSER IMPLEMENTATION

The current implementation of the 5470 parser contains a Marcus–type syntactic parser coupled with mechanisms for manipulating grammatical relations and semantic roles. It also relies on a semantic representation scheme in which case frame structures are embedded in a semantic net consisting of similarity and function links to other case frames. As the syntactic parse progresses, the grammatical relations of the np's in the sentence are identified. Once the syntactic parser has completed its task, it calls the function DO–RELATIONAL–RULES, which is handed a disembodied p–list which represents the relational network. This function "undoes" the relational operations as we outlined in our discussion of grammatical relations through a series of PUTPROPs and REMPROPs. Once the initial relations have been reached, the function FILL–CD takes the relational network, along with a set of slot–mappings (which are produced by modifying a set of global defaults, such as SUBJ -> ACTOR, DOBJ -> OBJECT, with whatever verb–specific mappings are appropriate) and maps in the values for the case frames.

The dictionary entries contain both syntactic and relational slots. The "part–of–speech" slot for verbs contains subcategorization information, and the "features" slot specifies syntactic/morphological properties peculiar to this particular word. Two fields "gr<->slots" and "slot–completions" contain crucial information which ties the semantic properties of "fly" to its syntactic properties. They rely on the relational categories "subject" and "object" and also on the functional/semantic notions of "command," "control," and "instrumentality." The latter concepts are used to guide the choice of constituents to fill the case frame slots "actor," "object," and "instrument." An actor, for example, must be able to command an action in the sense that he must be able to cause the action to occur. A true actor must also be able to control the action in the sense of being able to implement it himself without benefit of some instrument. Any candidate for the actor slot that does not meet these conditions should rightfully be relegated to the object or instrument slot. The set of word-specific slot-mappings, together with the default specifications, specify the possible sources for these roles; the slot-completions apply to provide information about the relationship between semantic roles. An example may clarify this.

The verb fly has a number of senses as is illustrated by the following sentences:

1. The businessman flew the plane to Cairo. - The businessman was a passenger.
2. The pilot flew the plane to Cairo. - The pilot actually manipulated the plane's controls.
3. The mummy flew to Cairo. - The mummy was cargo.
4. The plane flew to Cairo. - The plane was the instrument of its own motion.
5. The pigeon flew to Cairo. - The bird not only was the instrument of its own motion but also willed its motion.

The filled case frames for the verb fly within sentences 1-5 are as follows:

1. fly: actor
object businessman
instrument plane
2. fly: actor pilot
object plane
instrument plane
3. fly: actor
object mummy
instrument
4. fly: actor
object plane
instrument plane
5. fly: actor pigeon

object pigeon
instrument pigeon

As one can see from this example, one needs information about command, control, and instrumentality to make these choices. There is some evidence to suggest that these three concepts have cross-linguistic validity and are thus not an ad hoc solution to the problem.

4. PORTING THE INTERFACE

4.1 THE TOOLS

Porting the intelligent interface from one expert system backend to another will always be a demanding task requiring several months of high level effort and considerable thought. However, the job has been made easier by the development of several software tools specifically provided by the 5470 developers to help in porting. Broadly speaking, porting the interface to a new domain requires three things: writing a new vocabulary to define domain specific concepts, defining a new set of scenes for the target backend, and "gluing" the interface onto the new backend by writing the code to connect the interface's output to the backend routines that users wish to invoke. Writing the "glue" is a tedious but not very challenging task that must be done by hand whenever the system is ported. The process cannot be automated because it involves no generality; the whole process is specific to each backend. Vocabulary definition and scene definition, however, exhibit a great deal of generality from domain to domain, and the intelligent interface developers were able to build tools to simplify and speed both tasks.

4.2 THE VOCABULARY DEFINITION TOOL

The purpose of this tool is help system maintainers to define new words. Two considerations guided the design of this tool: the developers wanted to capture as many generalizations about word definition as possible, and they wanted to guide the word definer so that he would neither have to write code nor be forced to make *ad hoc* decisions about meaning.

There is one *caveat* that must be offered to users of the system, and that is that no amount of clever toolsmithing can alter the fact that defining new words requires thought and some training in whatever formalism is chosen to represent dictionary entries. Neither machines nor people learn new words by telepathy; if one considers the amount of knowledge a human being requires to understand an entry in a typical dictionary one realizes that it is considerable. The entry, below, for example, is taken from the definition of the word crumble in *Webster's Ninth New Collegiate Dictionary:*

crumb ble vb crum bled crum bling [alter. of ME kremelen, freq. of OE gecrymian to crumble, fr. cruma] vt (15c): to break into small pieces - vi: 1. to fall into small pieces :DISINTEGRATE 2. to fall into ruin : (COLLAPSE) <marriages -> – crumble n.

Even after disregarding the etymological obscurities, one can see that the reader needs to know what nouns, verbs, transitive and intransitive verbs are and also how to interpret the morphological information provided by the past and present participles. This not a job for the uneducated; neither is defining words for the intelligent interface.

Defining words using the King Kong Vocabulary Definition Package is a two-step process. First, the definer defines the morphological and syntactic properties of a word, then he defines the meaning of the word in the context of a scene.

The definer invokes the tool by hitting Select-V on the Lisp Machine keyboard. The tool then brings a many-paned window that covers the entire screen display. Each pane contains information about word definitions. One pane contains documentation about the entire definition process; if the user gets confused at any time he can scroll through this window in order to obtain documentation about how the vocabulary definition tool works.

The first thing that happens is that a small write in window appears on the display to ask the user what word he wishes to define. After the user has typed the word in, the system asks him if the word is a synonym of any word already in the dictionary. If it is a synonym, the system simply asks to user to define any irregular forms the new word has and then enters the word into the dictionary. Defining synonyms is trivially easy and takes seconds.

In most cases, the user needs to define a non-synonym. In this case he must specify the part of speech of the word and its syntactic behavior such as subcategorization. Menus pop up to guide each step in the specification of syntactic behavior, and there is a heavy use of defaults. The process is still complex and requires a college level understanding of English grammar, but there is always on-line documentation, and the use of menus means that even if the user makes a mistake, he can gracefully and quickly recover. No programming whatsoever is required; the user can make only choices permitted by the menus, and there is no place for him to go really wrong. With training users quickly become comfortable with the system, and the 5470 research team has shown that, with the tool, it is possible to define words in a matter of minutes (see the description of porting the interface to ISFI below.)

When the user has finished specifying the morphological/syntactic behavior of the word he must tell the system what it means. In the world of the King Kong Vocabulary Tool this means that the user must define word meanings in the context of the scenes. The tool guides the user by popping up a menu of all the scenes that have been defined for the backend. The user chooses a scene, and menus appear asking him to specify the roles of that scene and what relation they have to the direct object, subject, indirect object, and other arguments of the word. This process is necessary for verbs, but not for other parts of speech. For example, if the user were defining the verb "leave" in the context of the "choose homebase" scene, a menu would pop up directing him to relate the direct object of the verb with a scene role. He would relate the direct object to the role "homebase" because typical uses of the verb would be in sentences like "leave Hahn."

This is all there is to defining a word. By giving a command to the tool to install the new definition, the user can enter the word permanently in the dictionary and turn to defining another. Using this tool, the 5470 developers were able to create a 400 word dictionary in ten days. This is a much faster and much less error prone process than that of creating word definitions in CD oriented systems and requires no programming at all.

4.3 SCENE DEFINITION

The scene definition tool has the same window/menu organization that the vocabulary definition tool does, but has an additional graphic capability which is used to display the tree structure representing the scene hierarchy. Usrs can enter a new scene into the hierarchy graphically by pointing the mouse at a node in the tree and clicking to open up a new node. They can enter scene information such as roles and role fillers via windows; any updates to the hierarchy are automatically made to the tree and graphically displayed as they are made. All the petty details of scene definition are handled without the need for writing flavor definitions or any other type of code; the user is freed to do the hard part: choosing appropriate scenes for a backend and deciding what their roles and relationships should be.

This is probably the hardest part of porting, requiring good sense, creativity, and a sound understanding of the backend and of the needs of its users. Our own experience in porting the system to a new backend, the automatic programming system, showed that this was indeed the most demanding phase of porting.

5. PORTING THE INTERFACE TO ISFI

The job of porting the interface to a new backend was entrusted to one staffer who was given four months in which to get the interface onto a new backend and to bring it to the point at which it could handle user questions about the new expert system's domain. We chose the MITRE automatic programming expert system, ISFI, as our target backend because it differed dramatically both in domain and in architecture from KRS. It remains our belief that if we can port the interface to ISFI, we can certainly port it to expert systems more like KRS.

Within four months the chosen developer had defined a large dictionary for ISFI, a preliminary set of scenes, and the bulk of the interface to backend glue. Because ISFI already has a sophisticated graphical interface which enables its users to issue commands to the automatic programming system, we concentrated on giving the linguistic interface the ability to understand questions. ISFI is now capable of responding to a variety of English questions about the objects in its domain, furthermore, ISFI users have found that they are able to access information about ISFI using English that would otherwise be very hard to obtain. In this way, the addition of a linguistic interface to the preexisting graphical interface has greatly enriched the user's ability to communicate with the expert system.

However, we discovered in the course of talking to the ISFI developers and in playing with the system ourselves, that the sorts of questions that one would be most interested in asking ISFI were quite different from those that one would be likely to ask KRS. Users would prefer to ask ISFI "why" questions such as "Why did propagation fail?" rather than "what" or "how" questions. This poses a problem - not so much to the interface staff as to the developers of ISFI. The problem is that ISFI frequently does not record the reasons it does things. This means that there is no way for the interface to answer "why" questions even though the interface is fully capable of understanding them. In order for such questions to be answerable, the ISFI staff must extend ISFI, and, in fact, they are in the process of doing so. Here is an example of an interface driving the development, in part, of a backend system.

Some work remains to be done in scene definition and inference specification for ISFI. The porting is not quite complete. However, the relative speed and ease with which so much has been accomplished indicates that we have met our goal in creating a portable interface. No porting will ever be painless, and all will take time, but the combination of the system's architecture and its tools greatly simplify this demanding task.

6. WEAKNESSES OF THE INTERFACE

Although the 5470 project can justifiably be viewed as successful, it has several weaknesses. The most serious failing is that the parser lacks full syntactic coverage of English. In particular, the parser cannot handle conjunction or lexical ambiguity. In order to deal with these phenomena it will be necessary to select a faster parsing algorithm than is used by the current parser because conjunction and lexical ambiguity require the parser to search amongs a very large space of possible parse trees. Fortunately, one can still communicate in a fairly natural manner without resorting to conjunction, and one can restrict word definitions to a single part of speech without losing essential expressive power. Furthermore, one can at least predict the types of input likely to cause the parser to fail; this means that a user will not be mystified such failure and can quickly be trained to avoid certain constructions. This is much less painful than a situation in which the user has no idea whatsoever which constructions will work and which will not.

Nonetheless, the staff of 5470 regards the lack of full syntactic coverage to be a serious failing. Extending the parser would have been a top priority for a third year of effort had the project continued.

The interface also has at least one serious semantic gap: it cannot handle the meaning of quantifiers. This weakness, unfortunately, seriously undermines the useability of the system as an interface to databases since quantified questions are extremely common in database interactions. Again, this problem would have been a top priority for a third year of project effort.

Finally, we would have liked to have had the time to polish up the error messages. It is very important to provide the user with helpful messages when the system for whatever reason is unable to understand an input. We have provided some helpful messages, but there are still many cases in which messages are obscure or inadequate. That we were unable to improve this feature of the system remains one of our regrets.

BIBLIOGRAPHY

Bayer, Samuel. "A Theory of Linearization in Relational Grammar," Senior essay, Yale University, 1984.

Dyer, Michael. *In-Depth Understanding*. Cambridge: MIT Press, 1983.

Kaplan, R. M. "Augmented Transition Networks as Psychological Models of Sentence Comprehension," Artificial Intelligence, 1972.

Kay, Martin. "The MIND System." In Rustin, 1973.

Marcus, Mitchell. *A Theory of Syntactic Recognition for Natural Language*. Cambridge: MIT Press, 1980.

Pazzani, Mike. "APE - A Parsing Example" (unpublished research) MITRE, 1980-83.

Perlmutter, David, ed. *Studies in Relational Grammar 1*. Chicago: University Chicago Press, 1983.

Rustin, R., ed., *Natural Language Processing*. New York: Algorithmics Press, 1973.

Wasow, Tom. "Remarks on Processing, Constraints, and the Lexicon." In D. Waltz, ed., Theoretical Issues in Natural Language Processing - N.Y. ACM, 1978.

Woods, Bill. *The Lunar Sciences Natural Language Information System*. Cambridge: BBN Report No. 2378, Bolt Beranek and Newman, 1972.

Woods, Bill. "An Experimental Parsing System for Transition Network Grammars." In Rustin 1973.