An Architecture for Voice Dialog Systems Based on Prolog-Style Theorem Proving

Ronnie W. Smith*
East Carolina University

D. Richard Hipp†
Hipp, Wyrick & Company, Inc.

Alan W. Biermann‡
Duke University

A pragmatic architecture for voice dialog machines aimed at the equipment repair problem has been implemented. This architecture exhibits a number of behaviors required for efficient human-machine dialog. These behaviors include:

(1) *problem solving to achieve a target goal*

(2) *the ability to carry out subdialogs to achieve appropriate subgoals and to pass control arbitrarily from one subdialog to another*

(3) *the use of a user model to enable useful verbal exchanges and to inhibit unnecessary ones*

(4) *the ability to change initiative from strongly computer controlled to strongly user controlled or to some level in between*

(5) *the ability to use context dependent expectations to correct speech recognition and track user movement to new subdialogs.*

The paper gives a description of the dialog theory, presents examples of its capabilities, and includes a detailed trace of one of those examples showing all significant mechanisms. The paper gives performance data for a series of 141 problem-solving dialogs carried out with human subjects.

1. A Theory of Voice Dialog Systems

This paper presents a theory of voice dialog systems that integrates into a single, self-consistent architecture a variety of capabilities necessary for successful dialog. These capabilities include problem solving, natural language input and output, a user model, variable initiative, and the use of expectation for speech error correction and plan recognition. The theory revolves around a Prolog-style theorem-proving model with a variety of special features to accommodate the needs of a dialog system. The paper describes the target behaviors of the processor, the theory that achieves them, an implementation of the theory in a system to aid in electric circuit repair, and performance statistics gathered as human subjects used it in a series of tests.

* Department of Mathematics, East Carolina University, Greenville, N.C. 27858.
† Hipp, Wyrick & Company, Inc., 6200 Maple Cove Lane, Charlotte, N.C. 28269.
‡ Department of Computer Science, Duke University, Durham, N.C. 27706.

2. Target Behaviors

The purpose of the architecture is to deliver to users in real time the behaviors needed for efficient human–machine dialog. Specifically, it is aimed at achieving the following:

Convergence to a goal. Efficient dialog requires that each participant understand the purpose of the interaction and have the necessary prerequisites to cooperate in its achievement. This is the *intentional structure* of Grosz and Sidner (1986), the goal-oriented mechanism that gives direction to the interaction. The primary required facilities are a problem solver that can deduce the necessary action sequences and a set of subsystems capable of carrying out those sequences.

Subdialogs and effective movement between them. Efficient human dialog is usually segmented into utterance sequences, *subdialogs,* that are individually aimed at achieving relevant subgoals (Grosz 1978; Linde and Goguen 1978; Polanyi and Scha 1983; Reichman 1985). These are called "segments" by Grosz and Sidner (1986) and constitute the *linguistic structure* defined in their paper. The global goal is approached by a series of attempts at subgoals each of which involves a set of interactions, the subdialogs.

An aggressive strategy for global success is to choose the subgoals judged most likely to lead to success and carry out their associated subdialogs. As the system proceeds on a given subdialog, it should always be ready to drop it abruptly if some other subdialog suddenly seems more appropriate. This leads to the fragmented style that so commonly appears in efficient human communication. A subdialog is opened, leading to another, then another, then a jump to a previously opened subdialog, and so forth, in an unpredictable order until the necessary subgoals have been solved for an overall success.

An accounting for user knowledge and abilities. Cooperative problem solving involves maintaining a dynamic profile of user knowledge, termed a *user model.* This concept is described, for example, in Kobsa and Wahlster (1988, 1989), Chin (1989), Cohen and Jones (1989), Finin (1989), Lehman and Carbonell (1989), Morik (1989), and Paris (1988). The user model specifies information needed for efficient interaction with the conversational partner. Its purpose is to indicate what needs to be said to the user to enable the user to function effectively. It also indicates what should be omitted because of existing user knowledge.

Because considerable information is exchanged during the dialog, the user model changes continuously. Mentioned facts are stored in the model as known to the user and are not repeated. Previously unmentioned information may be assumed to be unknown and may be explained as needed. Questions from the user may indicate lack of knowledge and result in the removal of items from the user model.

Change of initiative. A real possibility in a cooperative interaction is that the user's problem-solving ability, either on a given subgoal or on the global task, may exceed that of the machine. When this occurs, an efficient interaction requires that the machine yield control so that the more competent partner can lead the way to the fastest possible solution. Thus, the machine must be able to carry out its own problem-solving process and direct the actions to task completion or yield to the user's control and respond cooperatively to his or her requests. This is a variable initiative dialog, as studied by Kitano and Van Ess-Dykema (1991), Novick (1988), Whittaker and Stenton (1988), and Walker and Whittaker (1990). As a pragmatic issue, we have found that at least four initiative *modes* are useful:

(1) *directive.* The computer has complete dialog control. It recommends a subgoal for completion and will use whatever dialog is necessary to obtain the needed item of knowledge related to the subgoal.

(2) *suggestive.* The computer still has dialog control, but not as strongly. The computer will make suggestions about the subgoal to perform next, but it is also willing to change the direction of the dialog according to stated user preferences.

(3) *declarative.* The user has dialog control, but the computer is free to mention relevant, though not required, facts as a response to the user's statements.

(4) *passive.* The user has complete dialog control. The computer responds directly to user questions and passively acknowledges user statements without recommending a subgoal as the next course of action.

Expectation of user input. Since all interactions occur in the context of a current subdialog, the user's input is far more predictable than would be indicated by a general grammar for English. In fact, the current subdialog specifies the *focus* of the interaction, the set of all objects and actions that are locally appropriate. This is the *attentional structure* described by Grosz and Sidner (1986), and its most important function in our system is to predict the meaning structures the user is likely to communicate in an input. For illustration, the opening of a chassis cover plate will often evoke comments about the objects behind the cover; the measurement of a voltage is likely to include references to a voltmeter, leads, voltage range, and the locations of measurement points.

Thus the subdialog structure provides a set of expected utterances at each point in the conversation, and these have two important roles:

(1) The expected utterances provide strong guidance for the speech recognition system so that error correction can be enhanced. Where ambiguity arises, recognition can be biased in the direction of meaningful statements in the current context. Earlier researchers who have investigated this insight are Erman et al. (1980), Walker (1978), Fink and Biermann (1986), Mudler and Paulus (1988), Carbonell and Pierrel (1988), Young (1990), and Young et al. (1989).

(2) The expected utterances from subdialogs other than the current one can indicate that a shift from the current subdialog is occurring. Thus, expectations are one of the primary mechanisms needed for tracking the conversation as it jumps from subdialog to subdialog. This is known elsewhere as the *plan recognition problem,* and it has received much attention in recent years. See, for example, Allen and Perrault (1980), Allen (1983), Kautz (1991), Litman and Allen (1987), Pollack (1986), and Carberry (1988, 1990).

Systems capable of all of the above behaviors are rare, as has been observed by Allen et al. (1989): "no one knows how to fit all of the pieces together." An impressive early example along these lines is the MINDS system of Young et al. (1989). This system maintains an AND–OR goal tree to represent the problem-solving space, and it engages in dialog in the process of trying to achieve subgoals in the tree. A series of interactions related to a given subgoal constitute a subdialog, and expectations associated with currently active goals are used to predict incoming user utterances. These predictions are further sharpened by a user model and then are passed down to the signal-processing level to improve speech recognition. The resulting system demonstrated dramatic improvements over performance levels that had been observed without such predictive capabilities. For example, the effective perplexity in one test was reduced from 242.4 to 18.3 using dialog level constraints, while word accuracy recognition was increased from 82.1 percent to 97.0 percent.

In another dialog project, Allen et al. (1989) describe an architecture that concentrates on representations for subdialog mechanisms and their interactions with sentence-level processing. Their mechanism uses the blackboard organization, which displays at a global level all pertinent information and has subroutines with specialty functions to update the blackboard. There are subroutines, for example, to do processing at the lexical, syntactic, and semantic levels, to handle referencing problems, to manage discourse structure and speech act issues, tense, and much more. A typical task for their system is to properly parse and analyze a given dialog.

A third interesting project has produced the TINA system (Seneff 1992), which uses probabilistic networks to parse token sequences provided by a speech recognition system, SUMMIT, created by Zue et al. (1989). The networks and their probabilities are created automatically from grammatical rules and text samples input by the designer. Their main utility is to provide expectation for error correction as we do in our system. However, their expectation is primarily syntax-based while ours uses structure from all levels, subdialog (or focus-based), semantic, and syntactic. Their semantics is built directly into the parse trees and translated into SQL for access to a database.

Our approach resembles that of Young et al. (1989), except that we employ Prolog-style rules for the knowledge base and the associated proofs for directing the goal-oriented behavior. This leads to the "missing axiom theory" we describe for processing discourse, and some rather simple mechanisms for employing the user model, for managing multiple subdialogs, and for creating and using expectation. Our theory also shows how variable initiative is built into the same simple architecture. Our implementation shows that the mechanisms are efficient enough to run in real time and sufficiently well designed to yield successful dialogs with humans.

3. Sample Dialogs

Many of the points discussed above are illustrated by the following excerpts from repair dialogs using our system. In the following example, the system is in directive mode. It has determined that a particular knob should be set to 10 and then a voltage measurement should be made. The movement to subdialogs is indicated by indentation.

1.	Computer:	Put the knob to one zero.
2.	User:	Okay.
3.	Computer:	What is the voltage between connector one two one and connector three four?
4.	User:	I do not know.
5.	Computer:	Locate the voltmeter.
6.	User:	Done.
7.	Computer:	Add a wire between the minus com hole on the voltmeter and the connector one two one.
8.	User:	How do I do that?
9.	Computer:	Connect the end of the black wire with the small plug to the minus com hole on the voltmeter.
10.	User:	Done.

11. Computer: Connect the end of the black wire with the large plug to connector one two one.
12. User: Which knob?
13. Computer: Near the top boundary and in the center is a green region. Now, in the middle right of this green region is the control knob.
14. User: Okay.
15. Computer: Put the knob to zero one.
16. User: Done.
17. Computer: Connect the end of the black wire with the large plug to connector one two one.
18. User: Where is connector three four?
19. Computer: Near the left boundary and toward the middle is a green region. Now, in the bottom center of this green region is connector three four.
20. User: Okay.
21. Computer: What is the voltage between connector one two one and connector three four?
22. User: Zero.

This dialog fragment is interesting in that it illustrates goal-oriented behavior while simultaneously jumping between subdialogs. The dialog also illustrates the effects of the user model. For example, the user model at the beginning of the dialog indicates that the user can find the knob (find(knob)). This assertion prevents the system from giving any explanation about the location of the knob. However, utterance 12 indicates an error in the current model and the system removes this assertion from the user model. This leads to a clarification subdialog explaining the position of the knob and a response from the user, "Okay." The user's affirmation then enables the system to add the assertion find(knob) back into the user model and proceed.

The use of expectation can also be illustrated by the example. When the user is preparing to respond with utterance 2, the system expects the response to pertain to the command "Put the knob to one zero." The primary expectations are for an assertion that this request has been done or for a question about how to do this task. Consequently, speech recognition can be biased to accept this limited class of utterances. A different use of expectation appears in sentence 12, where local expectations were related to the connection of a wire. Here, local expectations failed but a previous expectation from sentence 1 is fulfilled. So control is returned to the previously active subdialog where the goal was to "Put the knob to one zero."

The following dialog segment illustrates the behavior of the system in declarative mode.

1. Computer: This is the circuit-fixit-shop. How may I help you?
2. User: RS111 is not working.
3. Computer: I am familiar with that circuit. The LED is supposed to be displaying alternately flashing one and seven.
4. User: There is no wire on one zero four.
5. Computer: There is supposed to be a wire between connector one zero four and connector one zero two.
6. User: The circuit is working.
7. Computer: Glad to have been of assistance. Good-bye.

Here the machine diagnoses as well as it can the current subdialog from user comments. Then it presents information it surmises may be helpful, specifically facts from the currently active subdialog that are not in the user model. The user may ignore suggestions by following his or her own preferred dialog paths, as occurs in statements 4 and 6. In each case, the system tracks the selected topic and responds in an appropriate manner.

4. Mechanisms for Achieving the Behaviors

This paper presents a single self-consistent mechanism capable of achieving simultaneously the above-described behavior. We will examine sequentially (1) a theory of task-oriented language, (2) an implementation of the subdialog feature, (3) a method for accounting for user knowledge, (4) mechanisms needed to obtain variable initiative, and (5) the implementation and uses of expectation. The section following this one will give a detailed example showing all of these mechanisms working together.

4.1 A Theory of Task-Oriented Language

The central mechanism of our architecture is a Prolog-style theorem-proving system. The goal of the dialog is stated in a Prolog-style goal and rules are invoked to "prove the theorem" or "achieve the goal" in a normal top-down fashion. If the proof succeeds using internally available knowledge, the dialog terminates without any interaction with the user. Thus it completes a dialog of length zero. More typically, however, the proof fails, and the system finds itself in need of more information before it can proceed. In this case, it looks for so-called "missing axioms," which would help complete the proof, and it engages in dialog to try to acquire them.

As an example, the system might have the goal of determining the position, up or down, of a certain switch sw1. This might appear in the proof tree as

$$\text{observeposition(sw1,X)} \leftarrow \text{find(sw1), reportposition(sw1,X)}$$

That is, it is necessary to find sw1 and then to report its position X. It is possible that in the process of the interaction the user has both found the switch and reported its position and that both find(sw1) and report position(sw1,up) appear in the database. Then the system will achieve observeposition(sw1,up) and answer its question without interaction with the user. It is also possible that the user has previously found the switch but not recently reported its position. Here theorem proving would succeed with the first goal, find(sw1), but fail to find reportposition(sw1,-). Then reportposition(sw1,X) would become a missing axiom and could be returned to the dialog controller for possible vocalization. The third possibility is that neither find(sw1) or reportposition(sw1,-) would exist in the database, in which case both could be sent to the controller as missing axioms for possible vocalization. (The decision as to whether to send a missing axiom to the controller depends on whether the axiom represents an answerable question by the user. The system maintains a mechanism for indicating what is reasonable to ask and what is not, as described below.)

Thus our system is built around a theorem prover at its core, and the role of language is to supply missing axioms. Our system engages in dialog only for the purpose of enabling theorem proving, and voice interactions do not otherwise occur. (A later publication will propose other uses of voice interactions, but our current system uses them for only this purpose.)

The user may not always respond with the desired information. He or she may respond with a request for a clarification such as "Where is the switch?" or with an unanticipated comment such as "There is no wire connected to terminal 102." Thus the theorem prover needs to be immensely more flexible than ordinary Prolog, and this is the topic of the next section.

4.2 Implementing the Subdialog Feature

Instead of turning control over to a depth first policy, theorem proving in this system must allow for abrupt freezing of any proof and transfer of control to any other partially completed subproof or function of the system. The implementation of this was the IPSIM (Interruptible Prolog SIMulator) theorem prover, which can maintain a set of partially completed proofs and jump to the appropriate one as dialog proceeds.

The processing of IPSIM proceeds with normal theorem proving, but is interruptable in two ways. First, IPSIM may discover that there could be outside information that could be used to supply missing axioms, as described above. When this occurs, IPSIM can halt and pass control to the dialog controller, indicating an opportunity to engage in dialog. The dialog controller may choose to invoke the proposed interaction or it may select another action. Second, IPSIM may be interrupted by the dialog controller to inquire about proof status. In a real-time system, a theorem prover can never be released arbitrarily. Timing considerations by the controller may dictate the halting of a given proof and resorting to other action.

Since voice dialog is always tied to proving a given subgoal, the set of all interactions related to that goal comprise a *subdialog*. The set of logical rules leading to the subgoal are, by definition, related to that subgoal, and the voice interactions will necessarily have the coherence that theorists (Hobbs 1979) have often discussed.

The partial or completed proof of a subgoal is not erased or popped from any stack when processing moves to another part of the proof tree. This makes it possible to reopen any subdialog at a later time to clarify, revise, or continue that interaction. The reopening may be initiated either by the system because of a change in priorities in its agenda or by the user.

Subdialogs are entered in several ways. First, normal theorem proving may create a new subgoal to be proved, and its related voice interactions will yield a subdialog. Second, the controller may halt an interaction that it deems unfruitful and send the system in pursuit of a new subgoal. Third, the user may initiate dialog on a new subgoal in ways that will be discussed below.

4.3 Accounting for User Knowledge

Theorem proving will often reach goals that can only be satisfied by interactions with the user. For example, if the machine has no means to manipulate some variable and the user does, the only alternative is to make a request of the user. It is necessary that the knowledge base store information related to what the user can be expected to do, and the system only should make requests that will be within this repertoire. Of course, the abilities of the user will depend on his or her level of expertise and experience in the current environment. Thus most users will know how to adjust a knob to a specified level even if they are novices; but they may be able to measure a voltage only after they have been told all of the steps at least once in the current situation.

It is necessary to style outputs to the user to account for these variations, and the Prolog theorem-proving tree easily adapts to this requirement. The example above related to finding the position of a switch shows how this works. If the user knows, for example (according to the user model), how to find the switch, the model prevents the useless interaction related to finding the switch from occurring. If the user does not know (according to the user model) where the switch is, the model releases the system to so inform the user. The theory of user modeling thus is simply to specify user capabilities in the Prolog-style rules and let the natural execution of IPSIM select what to say or not say.

The user model must change continuously during a dialog as the interactions occur. Almost every statement will change the user's knowledge base, and future interactions will be ill-conceived if the appropriate updates are not made. Acquisition of user model axioms is made from inferences based on user inputs. A description of the inferences is given below:

(1) If the input indicates that the user has a goal to learn some information, then conclude that the user does not know about the information.

(2) If the input indicates that an action to achieve or observe a physical state was completed, then conclude that the user knows how to perform the action.

(3) If the input describes some physical state, then conclude that the user knows how to observe this physical state. In addition, if the physical state is a property, then infer that the user knows how to locate the object that has the property.

(4) If the input indicates that the user has not performed some primitive action, make the appropriate inference about the user's knowledge about how to perform this action.

(5) If the user has completed an action by completing each substep, then conclude that the user knows how to do the action.

(6) Infer that the user has intensional knowledge about a physical state if the user has knowledge on how to observe or achieve the physical state.

(7) Infer that the user has knowledge on how to observe a physical state if he or she has knowledge on how to achieve the physical state.

The basic implementation of these rules is a "compute_inferences" predicate in Prolog that takes the meaning of the user's current utterance and causes inferences to be asserted into the axiom base. Here is an example from the Prolog code. It is the implementation of statement (2) given above:

```
/* Inference 2:
If we have learned that an action to achieve or observe a physical state was completed, then conclude that the physical state has the appropriate status and that the user knows how to perform the action. */


compute_inferences(ProofNum,Meaning,_) :-
 Meaning = phys_state(prop(GoalAction,action_attribute,done),true),
 ( GoalAction = ach(phys_state(StateDes,TS)) ;
   GoalAction = obs(phys_state(StateDes,TS))),
 make_inference(ProofNum,phys_state(StateDes,TS),infer(Meaning)),
 make_inference(ProofNum,mental_state(user,int_know_action(how_to_do(
    GoalAction)),true),infer(Meaning)).
```

In typical dialogs, the user modeling system added a net of about 1.2 Prolog-style assertions to the user model per user utterance. There are both additions and deletions occurring after each utterance, and this figure gives the average increase in assertions per user utterance.

Formalizing a theory of what constitutes appropriate inferences could be a separate research project. What this research has contributed is a theory of usage for these inferences, which is usage by the theorem prover as it attempts to complete task goals.

4.4 Mechanisms for Obtaining Variable Initiative

Variable initiative dialog allows either participant to have control, and it allows the initiative to change between participants during the exchange. It also allows intermediate levels of control where one participant may gently rather than strongly lead the interactions.

A system can participate in variable initiative dialog if it properly manages (1) the selection of the current subdialog, (2) the level of assertiveness in its outputs, and (3) the interpretation of its inputs. We discuss each in the following paragraphs.

Selection of Subdialog. The most important aspect of dialog control is the ability to select the next subdialog to be entered. Very strong control means that the participant will select the subdialog and will ignore attempts by the partner to vary from it. Weaker control allows the partner to introduce minor but not major variations from the selected path. Loss of control means that the partner will select unconditionally the next subgoal.

In our system, the four implemented levels of initiative follow these guidelines:

(1) Directive Mode. Unless the user explicitly needs some type of clarification, the computer will select its response solely according to its next goal for the task. If the user expresses need for clarification about the previous goal, this must be addressed first. No interruptions to other subdialogs are allowed.

(2) Suggestive Mode. The computer will again select its response according to its next goal for the task, but it will allow minor interruptions to subdialogs about closely related goals. As before, user requests for clarification of the previous goal have priority.

(3) Declarative Mode. The user has dialog control. Consequently, the user can interrupt to any desired subdialog at any time, but the computer is free to mention relevant, though not required, facts as a response to the user's statements.

(4) Passive Mode. The user has complete dialog control. Consequently, the computer will passively acknowledge user statements. It will provide information only as a direct response to a user question.

Level of Assertiveness of Outputs. The system must output statements that are compatible with its level of initiative. This means that the output generator must have a parameter that enables the system to specify assertiveness. Examples of the request to "turn the switch up" at various levels of assertiveness are as follows:

Turn the switch up.	(command)
Would you [please] turn the switch up?	(request)
Can you turn the switch up?	(indirect request)
The switch can be turned up.	(statement of fact)
Turning the switch up is necessary.	(statement of fact)

Two examples of querying for the switch position are as follows:

What is the switch position?	(request)
I need to know the switch position.	(indirect request)

Interpretation of Inputs. Inputs from a passive participant can be expected to be more predictable and well behaved than those from a directive one. Our system does not account for this effect at this time. The only implemented variation in behavior concerns the treatment of silence. The system may allow rather longer silences when it is in passive mode than when it is in directive mode.

4.5 The Implementation and Uses of Expectation

The response received after a given input is likely to be related to the currently active subdialog. If it is not, then it may be related to a nearby active subdialog or, with less probability, a more remote one. The expectation facility provides a list of expected meanings organized in a hierarchy, and it is used for two purposes: (1) If the incoming utterance is syntactically near the syntax for an expected meaning in the active subdialog, the expectation provides a powerful error-correction mechanism. (2) If the incoming utterance is not in the locally active subdialog, the expectations of other active subdialogs provide a means for tracing the movement to those subdialogs. (This is known as "plan recognition" in the literature. See, for example, Allen and Perrault (1980), Allen (1983) and Carberry (1990). Expectations are specified in GADL (Goal and Action Description Language) form, an internal language for representing predicates. For example, the expectation that the user is going to report the setting of a switch would be represented as

obs(phys_state(prop(switch1,state, PropValue), TruthStatus)).

Expectation of user responses provides a model of the attentional state described by Grosz and Sidner (1986). It contains the list of semantic structures that have meaning for the current subdialog and for other active subdialogs. For example, after the computer produces an utterance that is an attempt to have a specific task step *S* performed, there are expectations for any of the following types of responses:

(1) A statement about missing or uncertain background knowledge necessary for the accomplishment of *S*.

(2) A statement about a subgoal of *S*.

(3) A statement about the underlying purpose for *S*.

(4) A statement about ancestor task steps of which accomplishment of *S* is a part.

(5) A statement about another task step which, along with *S*, is needed to accomplish some ancestor task step.

(6) A statement indicating accomplishment of *S*.

The central pragmatic issues for the management of expectation are, what are the sources of expectation (or how is the expectation list created) and how is it used. The following paragraphs describe both.

Sources of expectation. The first source of expectation is the *domain processor* described below. One of the main tasks of this system is to supply the dialog machine with debugging queries such as "What is the LED showing?" A secondary task is to associate with each such query a list of expected answers. Thus the query about the LED would yield as expectations some possible descriptions for the LED. These are called *situation_specific_expectations*. The domain processor also supplies *situation_related_expectations*, which are not directly connected to the observation but which could naturally occur. For example, the LED query might also result in observations about the presence or absence of wires, the position of the power switch, and the presence or absence of a battery.

The other source of expectations is the *dialog_controller*, also described below, which provides coordination for the complete system. The dialog controller manages a number of generic rules related to dialog and can provide the associated expectations. For example, "What is the LED showing?" can represent the action "*observe* the *value* for the display *property* of the *object* LED." The associated *task_specific_expectations* would represent expectations based on this general notion with values for *property* and *object* instantiated to the situation values. Thus the task-specific expectations for the sample topic would include questions on the location of the object, on the definition of the property, and on how to perform the action. In addition, these expectations would include responses that can be interpreted as state descriptions of the relevant property. In general, they include potential questions and statements about subtasks of the current task. There are rules for 12 different generic actions (Smith 1991). The rules are based on a characterization of response types obtained during a Wizard-of-Oz study on the effects of restricted vocabulary (Moody 1988).

The dialog controller also provides a broader class of expectations, called *task-related expectations*, which are based on general principles about the performance of actions. Example task-related expectations would include general requests for help or questions about the purpose of an action. Another important member of the task-related expectations are the expectations for topics that are ancestors of the current topic in the discourse structure. For example, the current topic could be the location of a connector, which could be a subtopic of connecting a voltmeter wire, which could be a subtopic of performing a voltage measurement. The task-related expectations for the location of this connector would include all the expectations related to the topics of connecting a voltmeter wire and performing a voltage measurement.

Utilizing expectation. After the semantic expectations are computed, they are translated into linguistic expectations according to grammatical rules. Once the linguistic expectations are produced, they are labeled with an *expectation cost*, which is a measure of how strongly each is anticipated at the current point in the dialog. The situation-specific expectations are the most strongly anticipated, followed by the other three types. Meanings output by the minimum distance parsing algorithm (described below) have a corresponding utterance cost, which is the distance between the user's input and an equivalent well-formed phrase. Each meaning is matched with its corresponding dialog expectation, and its expectation and utterance costs are combined into a total cost by an expectation function. The meaning with the smallest total cost is selected to be the final output of the parser. An important side effect of matching meanings with expectations is the ability to interpret an utterance whose content does not fully specify its meaning. These semantic and linguistic expectations can provide the necessary context. Some examples are as follows:

(a) *The referent of pronouns* (In the implemented system, the only pronoun is "it.") The parser leaves the slot for the referent of "it" unspecified in its interpretation. If this interpretation of the utterance can be matched to the linguistic expectation, the value for "it" is filled with the value provided by the expectation. Consider the following example:

(1) Computer: Turn the switch up.

(2) User: Where is it?

In computing the task-specific expectations for the user utterance, one expectation is for a statement asking about the location of the object of interest in the current topic—in this case the switch. The parser interprets the user statement as a statement asking about the location of an unspecified object. The linguistic expectation provides the value of the unspecified object.

(b) *The meaning of short answers* (In the implemented system, these include such responses as "yes," "no," and "okay.") The idea for each of these is similar. These utterances may have any of several meanings. The proper choice is determined by the expectations produced based on the situation. In the following example, it is likely that "okay" denotes affirmation of completion of the goal to turn up the switch:

(1) Computer: Turn the switch up.

(2) User: Okay.

It is less likely that it denotes comprehension of the request. In any case, after statement (1), the situation-specific expectations include an expectation for an affirming utterance indicating completion, and this becomes the interpretation given to "okay." Contrast this with the following:

(1) Computer: Turn up the switch.

(2) User: Where is it?

(3) Computer: In the lower left corner.

(4) User: Okay.

In this case, the interpretation of "okay" could be either that the location description (3) has been understood or that the original goal (1) has been accomplished. The expectation system scores the likelihood of each meaning and selects the most likely one using its scoring method.

(c) *Maintain dialog coherence.* Consider the following subdialog taken from usage of the implemented system:

(1) Computer: What is the voltage between connector 121 and connector 120?

(2) User: I need help.

(3) Computer: Locate the voltmeter.

(4) User: Done.

(5) Computer: Add a wire between the "- com" hole on the voltmeter and connector 121.

(6) User: Done.

(7) Computer: Add a wire between the "+ v omega a" hole on the voltmeter and connector 120.

(8) User: Nine.

When utterance (8) is spoken, there are two active task steps: (1) performing the voltage measurement and (2) connecting a wire between the "+ v omega a" hole and connector 120. The user response does not satisfy the missing axiom for completing the substep (7). The expectations for the response to (7) are checked, but this utterance is not one of them. However, (8) does satisfy the missing axiom for completing the main task step (1). It has meaning in that context and it is so interpreted.

5. The Zero-Level Model

The system built in our laboratory (described at length in Smith [1991] and Hipp [1992] and Smith and Hipp [1995]) implements the theory given above. Figure 1 presents a *zero-level model* of the main processor, which illustrates the system principles of operation without burdening the reader with too much detail. This model is the recursive subroutine ZmodSubdialog, and it is entered with a single argument, a goal to be proven. Its actions are to carry out a Prolog-style proof of the goal. A side effect of the proof may be some voice interactions with the user to supply missing axioms as described above. In fact, the only voice interactions the system undertakes are those called for by the theorem-proving machinery.

The ZmodSubdialog routine is a Prolog-style interpreter with a number of special features designed for the dialog processing application. It is typical of such interpreters in that it lifts goals from a priority queue and applies rules from the knowledge base to try to satisfy them. See, for example, the first and third branches beginning with "If R..." where respectively, the trivial case and the general case for applying a rule are handled. The deviations from a standard such interpreter are (1) in the second branch "If R...", which handles the case of a missing axiom where voice interaction is to be invoked, (2) in three steps (marked "mode") where processing will vary according to the level of the initiative the system is in, and (3) in the controlling module for the interpreter, which may freeze execution of this computation at any time to initiate or continue some other such computation.

Typical execution of ZmodSubdialog involves opening a proof tree and proceeding with a computation until an interrupt or clarification subdialog occurs. This may come, for example, from a new goal suggested by the domain processor or from a statement by the user causing movement to a different subdialog. The interrupt will cause control to pass to another existing proof tree (that was previously frozen) or to a new one aimed at the newly presented goal. Thus a set of partially completed trees will exist at all times, and control will jump back and forth between them. Of course, many of these trees will invoke associated voice interactions—and these constitute the subdialogs of the conversation.

The process of dialog described here is a kind of interactive theorem proving where the guidance down paths can come from either the user's knowledge or system knowledge. However, the emphasis in the traditional interactive theorem-proving literature is on giving the user substantial opportunity to propose the notations and individual steps of the proof in a way that is not possible or desirable in our environment.

```
Recursive subdialog routine (enter with a goal to prove)

ZmodSubdialog(Goal)

Create subdialog data structures
While there are rules available which may achieve Goal
   Grab next available rule R from knowledge; unify with Goal
   If R trivially satisfies Goal, return with success
   If R is vocalize(X) then
      Execute verbal output X                    (mode)
      Record expectation
      Receive response                           (mode)
      Record implicit and explicit meanings for response
      Transfer control depending on which expected response was received
         Success response: Return with success
         Negative response: No action
         Confused response: Modify rule for clarification; prioritize for execution
         Interrupt: Match response to expected response of another subdialog;
            Go to that subdialog      (mode)
   If R is a general rule then
      Store its antecedents
      While there are more antecedents to process
         Grab the next one and enter ZmodSubdialog with it
         If the ZmodSubdialog exits with failure then terminate processing of R
      If all antecedents of R succeed, return with success
Halt with failure

NOTE: SUCCESSFUL COMPLETION OF THIS ROUTINE DOES NOT NECESSARILY
MEAN TRANSFER OF CONTROL TO THE CALLING ROUTINE. CONTROL PASSES
TO THE SUBDIALOG SELECTED BY THE DIALOG CONTROLLER.
```

Figure 1
The zero-level model of the main subdialog processing algorithm.

6. Executing an Example Subdialog

The operation of ZmodSubdialog (and similarly our implemented system) becomes clear if a complete example subdialog is carried out. Here we will trace the execution of the example 22 utterance subdialog given in Section 3 and thereby illustrate the theory of operation in detail. An overview of the computation is given here and a detailed trace of all significant details appears in Appendix A. The following database of Prolog-like rules is needed for proper system operation.

Specific Device Debugging Rule

T1_circuit_Test2(V) $\leftarrow$ set(knob,10), measurevoltage(121, 34, V).

General Debugging Rules

set(knob,Y) $\leftarrow$ find(knob), adjust(knob,Y).
measurevoltage(X,Y,V) $\leftarrow$ find(voltmeter),

```
                                  set(voltmeterscale,20),
                                  connect(bw,com,X),
                                  connect(rw,+,Y),
                                  vocalize(read(voltmeter,V))
      set(voltmeterscale,20)

      connect(W,X,Y) ← connectend(W,X),
                       connectend(W,Y)
```

General Dialog Rules

```
      Y ← usercan(Y), vocalize(Y)
      vocalize(Y)
      find(Y) ← vocalize(find(Y))
```

User Modeling Rules

```
      find(knob)
      usercan(adjust(knob,X))
      usercan(measurevoltage(X,Y,V))
      usercan(find(voltmeter))
      usercan(connect(X,Y,Z))
      usercan(connectend(X,Y))
```

We assume that the machine has selected a new goal that comes from the domain processor: T1_circuit_Test2(V), where V is a voltage to be returned by the test. Thus, the domain processor is asking that test 2 on circuit T1 be performed returning a voltage V. ZmodSubdialog begins in Prolog fashion looking for a rule in the database to prove the goal T1_circuit_Test2(V), and it finds the debugging rule T1_circuit_Test2(V) ← set(knob,10), measurevoltage(121,34,V). Referring to Figure 1, this rule *R* is a general rule resulting in the third choice branch. Here the algorithm selects the first subgoal set(knob,10) of *R* and creates another subdialog by entering ZmodSubdialog with this subgoal.

The new subdialog finds the rule

set(knob,Y) ← find(knob), adjust(knob,Y),

which says the way to set the knob is to first find it and then do the adjustment. Execution of this rule demonstrates the mechanisms related to the use of the user model and the initiation of voice interaction. For example, the first new subgoal find(knob) causes a new entry into ZmodSubdialog, where it is immediately satisfied by find(knob) in the user model. That is, the user has achieved find(knob) (knows how to find the knob), and no further consideration of this subgoal is needed. If the user did not know (according to the user model) how to find the knob, the system might have invoked a voice interaction to try to achieve this subgoal. Moving to the second goal, adjust(knob,10), again it might have occurred that the user has just achieved this also. (Since the algorithm does unification as a rule is invoked, the variable Y has been set to 10.) Entry of ZmodSubdialog with this subgoal, however, finds no trivial resolution for this subgoal. But it can invoke

Y ← usercan(Y), vocalize(Y),

which says that a goal Y can be achieved if the user is capable of doing Y (which is represented as usercan(Y)) and if we vocalize Y. (This type of rule has not been explicitly implemented in our system, but we include it here as a model of what does happen.) Further recurrences on ZmodSubdialog discover usercan(adjust(knob,X)) and undertake vocalize(adjust(knob,10)).

Entry of ZmodSubdialog with vocalize(adjust(knob,10)) sends control down its second path. Sentence generation and voice output produces the statement

"Put the knob to one zero."

Next a set of expected responses is compiled. Some of these include:

question(location,knob).
question(ACTION,how-to-do).
assertion(knob,status,10).
assertion(ACTION,done).

When a vocalized response comes back, parsing and error correction will be biased to recognize one of these meanings.

After a response meaning has been resolved, it is entered into the database with all its presuppositions. For example, the mention of an object is assumed to indicate that the user can recognize and find that object if needed. These assertions are entered into the user model. The user model thus changes on almost every interaction to note new facts that are probably known to the user or to remove facts that the user apparently does not know. Finally, control changes because of the response either to (1) return from this subdialog with success, (2) continue in this subdialog searching for a success (having received an unsuccessful response), (3) enter special processing to deal with a need for clarification, or (4) interrupt processing to jump to some other subdialog.

In the example subdialog, the user responds "OK," which corresponds to one of the expected meanings: assertion(ACTION,done). Expectations equate ACTION to the current action. Consequently, there is a successful exit of the current subdialog and achievement of the set(knob,10) goal.

The first invoked rule,

T1_circuit_Test2(V) $\leftarrow$ set(knob,10), measurevoltage(121,34,V),

is thus half satisfied, and the goal measurevoltage(121,34,V) is undertaken.

The new goal leads to vocalization in the same manner described above. But the response in this case is negative: "I do not know." So attempts to achieve measurevoltage(121,34,V) continue and involve the rule

```
measurevoltage(X,Y,V) ← find(voltmeter),
                        set(voltmeter,20),
                        connect(bw,com,X),
                        connect(rw,+,Y),
                        vocalize(read(voltmeter,V)).
```

This leads to a number of interactions, as traced in detail in Appendix A. The reader should note in this continued interaction the manner in which the theorem proving drives the dialog and the user model inhibits or enables voice interactions appropriate to the situation.

The next interesting action occurs in utterance 12, when the user answers a request to connect a wire with the question "Which knob?" Here the response is parsed against expected meanings without success. So the system looks for expectations of other subdialogs that either have been invoked or might be invoked. In this example, "which knob" is an expected response for the first utterance, so control returns to that subdialog. In fact, in that subdialog, this response corresponds to a request for clarification. In our studies, we have found such requests for clarification to be routine and have designed a special mechanism for handling them. Our system dynamically modifies the active rule

```
adjust(knob,10) ← usercan(adjust(knob,10)),
                  vocalize(adjust(knob,10)).
```

to include a clarification subdialog:

```
adjust(knob,10) ← find(knob),
                  usercan(adjust(knob,10)),
                  vocalize(adjust(knob,10)).
```

Thus, the original user model was incorrect where it included find(knob), and this assertion is deleted. Next, as specified in Figure 1, the system reattempts the computation with this revised rule. The newly inserted subgoal causes the voice output of utterance 13.

This discussion shows the operation of all parts of the ZmodSubdialog model and illustrates the mechanisms used in our dialog machine. Appendix A gives the detailed steps required for completing the first part of the 22-utterance sample dialog.

7. An Overview of the System Architecture

The architecture of the system is given in Figure 2, where five major subsystems are shown: the dialog controller, the domain processor, the knowledge base, the general reasoning system, and the linguistic interface. These modules will be described next.

Dialog controller. This is the overall "boss" of the dialog processing system. It formulates goals at the top level to be passed on to the theorem-proving stage. It determines the role that the computer plays in the dialog by determining how user inputs relate to already established dialog as well as determining the type of response given. It also maintains all dialog information shared by the other modules and controls their activation. Its control algorithm is the highest-level dialog processing algorithm.

The basic cycle followed by the dialog controller is shown below.

(1) Obtain suggested goal from the domain processor.

(2) Based on the suggested goal and the current state of the dialog, select the next goal to be pursued by the computer and determine the expectations associated with that goal. (The goal may thus be selected from one of the active subdialogs. The choice is partially dependent on the current level of initiative.)

(3) Attempt to complete the goal using the IPSIM system, possibly invoking voice interactions.

(4) Update system knowledge based on efforts at goal completion.

(5) Determine next operations to be performed by the domain processor in providing a suggested goal.

(6) Go to step 1.

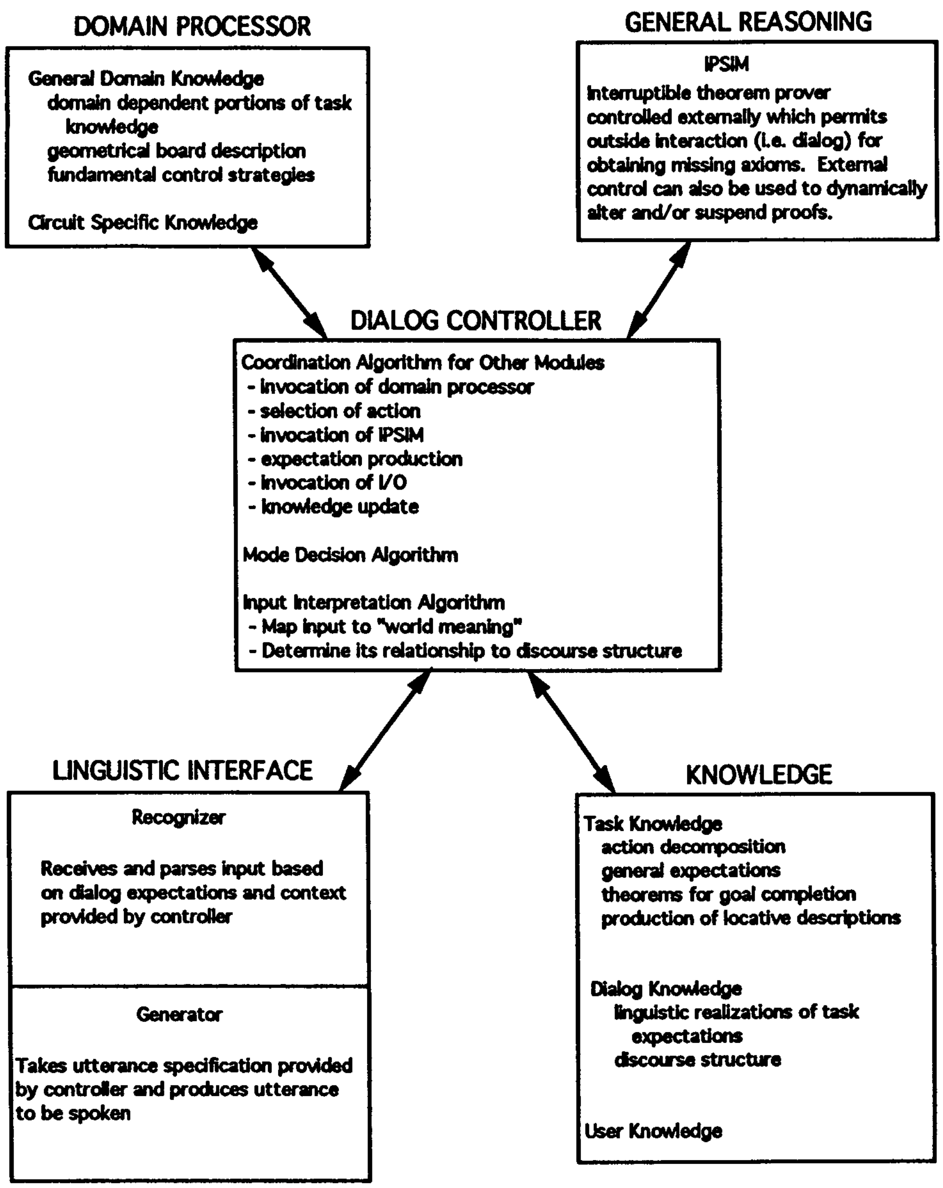

DOMAIN PROCESSOR
General Domain Knowledge
domain dependent portions of task knowledge
geometrical board description
fundamental control strategies
Circuit Specific Knowledge
GENERAL REASONING
IPSIM
Interruptible theorem prover controlled externally which permits outside interaction (i.e. dialog) for obtaining missing axioms. External control can also be used to dynamically alter and/or suspend proofs.
DIALOG CONTROLLER
Coordination Algorithm for Other Modules
- invocation of domain processor
- selection of action
- invocation of IPSIM
- expectation production
- invocation of I/O
- knowledge update
Mode Decision Algorithm
Input Interpretation Algorithm
- Map input to "world meaning"
- Determine its relationship to discourse structure
LINGUISTIC INTERFACE
Recognizer
Receives and parses input based on dialog expectations and context provided by controller
Generator
Takes utterance specification provided by controller and produces utterance to be spoken
KNOWLEDGE
Task Knowledge
action decomposition
general expectations
theorems for goal completion
production of locative descriptions
Dialog Knowledge
linguistic realizations of task expectations
discourse structure
User Knowledge

Figure 2
The system architecture.

The domain processor. This is the primary application-dependent portion of the system. It contains much of the information about the application domain. It receives from the controller a request for the next suggested goal to be undertaken, and it returns to the controller its suggestion along with expected results from attempting the test. It then receives the results of the interaction and appropriately updates its data structures.

In the implemented system, the domain processor assists in electronic equipment repair and contains a debugging tree that organizes the debugging task. The debugging tree has as its root a node representing the whole device to be debugged and other nodes representing all of the subsystems. It is organized using the "part of" relationship with each node that represents a part of a subsystem connected as a child node below that subsystem.

```
                      Device
            /            |            \
     Subsystem1   Subsystem 2   Subsystem 3
```

For example, in the implemented system, the top level device is a circuit called the RS111; its subsystems are the power circuit, the T1 and T2 circuits, and the LED circuit. Their subsystems are wires, switches, transistors, and so forth. The lowest level of this tree is the atomic element that can be addressed in a dialog.

Each node has as its primary constituents a set of specifications and a set of flags representing its state. The specifications have three parts, giving the observation to be made, the conditions to be satisfied before making the observation, and the actions to be taken depending on what is observed. An example of one of the specifications for the LED is as follows:

Observation: Observe the behavior of the LED.

Conditions: The switch must be on, and the control must be set to 10.

```
Actions: IF the LED is alternately displaying a 1 and 7 with frequency greater than
            once per second THEN assert the specification is satisfied
         ELSE IF the LED is not on
            THEN assert the battery is suspicious, in which case the power circuit
            is suspicious
         ELSE IF the LED is on but not blinking
            THEN assert that the transistor circuits are suspicious
         ELSE IF the LED is blinking but not alternately displaying a 1 and a 7
            THEN assert that the LED circuit is suspicious
         ELSE IF the LED is damaged
            THEN assert that the LED device should be replaced
         ELSE IF ...
```

The actions may be to assert that the specification is checked, to set suspicion flags on other nodes in the tree, or to replace parts.

The other major component of a node is a set of status flags for the subsystem represented by the node. One flag indicates whether the subsystem is checked, unchecked, partially checked, or suspicious. Other flags give for each individual specification its status and a counter for the number of iterations that the specification has been checked.

The domain processor algorithm chooses the node (subsystem) with the greatest suspicion and the specification on that node with greatest suspicion and sends it to the dialog controller for possible checking. When two nodes or specifications are tied for being most suspicious, finer-grained criteria are used to break the tie.

The algorithm is designed to guarantee that false information that may be entered into the tree will be eventually found. This is done by allowing the iterations counter on each specification to reduce its effective level of suspicion. Thus in a problematic debugging situation, specifications with the smallest count will be checked again and again until all have been checked the same number of times. Additional checking will then make the rounds of all specifications. If erroneous information has been entered after any observation, that observation will eventually be repeated, enabling progress and guaranteeing ultimate success.

The set of possible observations provides the situation-specific and situation-related expectations discussed in the section on expectations. Thus, in the example listed above where the LED is being observed, the expectations are as follows:

(1) the LED is alternately displaying a 1 and 7 with frequency greater than once per second

(2) the LED is not on

(3) the LED is on but not blinking

(4) the LED is blinking but not alternately displaying a 1 and a 7

(5) the LED is damaged

etc.

These are passed to the controller at the time of the request for the LED observation.

The reasoning system. The IPSIM system receives as input goals to be proven and commands to start, stop, and furnish information. It yields as output theorems that are proven and status reports on the proof in progress. Its processing follows the usual mechanisms of Prolog-style theorem-proving, and it is modeled by the ZmodSubdialog routine given above. It uses the rules in the knowledge base described below.

The knowledge base. This is the repository of information about task-oriented dialogs. This includes the following general knowledge about the performance of actions and goals:

(1) Knowledge about the decompositions of actions into substeps.

(2) Knowledge about theorems for proving completion of goals.

(3) Knowledge about the expectations for responses when performing an action.

There is also general task knowledge about completing locative descriptions. General dialog knowledge includes knowledge about the linguistic realizations of task expectations as well as discourse structure information maintained on the current dialog. Finally, there is also knowledge about the user that is acquired during the course of the dialog. Note that the predefined information of this module is easily modified without requiring changes to the dialog controller.

Linguistic interface. This system receives the spoken inputs from the user and returns spoken outputs. We will examine first the voice input system.

The inputs to the voice input system for each utterance are the set of expectations from the dialog controller and the speech utterance from the user. The output from this system is a GADL meaning representation.

The core of the processor is a syntax-directed translator (Aho and Ullman 1969) with rules of the form $A \rightarrow w_1{:}w_2$, where w_1 should be thought of as an ordinary context-free grammar right-hand side. If the terminal string w can be generated by the grammar rules of the from $A \rightarrow w_1$, then the analogous derivation using the rules $A \rightarrow w_2$ will produce the translated output in GADL form. As an example, suppose the utterance "no wire" has been received and the following rules are in the system grammar:

```
StartState → A : A
A → TIS NOT WIRE : assertion(NOT,state(exist,WIRE,present))
TIS → :
WIRE → DET WIRE2 : WIRE2
DET → :
NOT → no : false
WIRE2 → wire : wire(+,+)
```

Then the derivation of the source string yields the meaning string as shown.

StartState	StartState
A	A
TIS NOT WIRE	assertion(NOT,state(exist,WIRE,present))
NOT WIRE	assertion(NOT,state(exist,WIRE,present))
NOT DET WIRE2	assertion(NOT,state(exist,WIRE2,present))
NOT WIRE2	assertion(NOT,state(exist,WIRE2,present))
no WIRE2	assertion(false,state(exist,WIRE2,present))
no wire	assertion(false,state(exist,wire(+,+),present))

That is, the meaning representation from the input "no wire" is

assertion(false,state(exist,wire(+,+),present)).

The dialog controller, of course, will provide an expectation for each received input. In the current example, assume the machine has previously output "there should be a wire from terminal 102 to terminal 104." Then the expectations would be

assertion(true,state(exist,wire(102,104),present))

assertion(false,state(exist,wire(102,104),present))

assertion(true,do(user,achieve(state(exist,wire(102,104),present))))

question(yn,do_action(achieve(state(exist,wire(102,104),present))))

assertion(true,comprehension)

assertion(false, comprehension)

question(wh,location(102,_))

question(wh,location(104,_))

question(wh,how_to_do(achieve(state(exist,wire(102,104),present))))

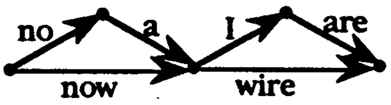
The selected meaning of the incoming utterance will be the least-cost match between an output for the translation grammar and the expectations. The mechanisms for finding this least-cost match will be described next.

The cost C of selecting a given expectation is a function of two parameters: (1) the utterance cost U, which measures the distance of the perceived voice signal from a grammatical utterance as defined by the system grammar, and (2) the expectation cost E, which measures the degree of locality of the selected expectation with respect to the current subdialog. The utterance cost U will be small if the voiced signal precisely matches a token sequence generated by the system grammar. The expectation cost E will be small if the meaning of the utterance as generated by the translation grammar precisely matches an expected meaning for the currently active subdialog. Thus the total cost can be represented as

$$C = f(U, E)$$

where the exact nature of the function f is a problem to be solved.

In this project, it was assumed that the speech recognizer would provide a graph of alternate guesses of the current input. Thus it might pass the following to the parser after a user had spoken "no wire."



The parser then searches for paths through the graph that match as closely as possible grammatical inputs. In searching for a path, the parser may delete or insert words to achieve a match. Each such edit operation has an associated cost, depending on the significance of the word being edited. Some words, such as "not" or major nouns, make a large difference in sentence meaning; other words, such as articles, may not carry significant meaning in a given context. U is the sum of the edit costs required to traverse the graph path and match a grammatical input. In the example, the path no-a-wire matches the grammatical "no wire" with the deletion of only an article "a."

The computation of E was simple in our project. A low value was assigned for all expectations at the current subdialog. The sequential levels of more distant subdialogs each were given higher expectation costs.

The original hypothesized combining function for U and E was

$$C = \beta U + (1 - \beta)E$$

where β is a weighting factor between 0 and 1. A β near 0 will tend to prefer matches to local expectation, regardless of the values of U; a β near 1 will place most weight on getting a good match between the input graph and a grammatical string, regardless of the value of E. Our experimentation, as described in Hipp (1992), indicated that β should be near 1. This supports the intuition that definitive source data at the time of the utterance should be the preferred evidence regardless of expectation. Consequently, the usefulness of the expectation is for selecting between grammatical utterances derived from the perceived voice signal that have minimal utterance cost. Reflecting this experimentally determined result, the cost computation was revised to

$$C = \begin{cases} E & \text{if } U = U_{\min} \\ \infty & \text{Otherwise} \end{cases} \qquad *$$

where $U_{\min}$ is the smallest observed utterance cost for the given utterance. This function selects the meaning with minimum utterance cost and uses expectation to break ties. A big advantage to using this form comes from the fact that any partial parse that exceeds the currently known minimum can be abandoned immediately at great savings in computation time.

The details of the minimization algorithm are given in Hipp (1992). It follows some of the ideas of Aho and Peterson (1972), Levinson (1985), and Lyon (1974), and will not be described here. It finds optimum answers in less than two seconds' time for most utterances of lengths used in the environment of our system when running on a Sun Sparc Station 2.

The voice output system will not be discussed here. It receives a GADL specification for an output and some parameters regarding the statement context, and uses a grammar to generate the desired word sequence. It uses the context information to adapt outputs to their environment and sends the sequence to a DECtalk system for voicing.

8. Some Implementation Details

The system has been implemented on a Sun 4 workstation with the majority of the code written in Quintus Prolog. The parser is coded in C. Speech recognition is performed by a Verbex 6000 user-dependent connected-speech recognizer running on an IBM PC, and the vocabulary is currently restricted to 125 words. The users are required to begin each utterance with the word "verbie" and end with the word "over." The Verbex machine acknowledges each input with a small beep sound. These sentinel interactions help to keep the user and machine in synchronization. The grammar used by the parser consists of 491 rules and 263 dictionary entries. The dictionary entries define insertion and deletion costs for individual words as well as substitution costs for phonetically similar words (such as "which" and "switch").

The dialog system exclusive of the parser and error correction code consists of about 17,000 lines of Prolog (and this includes some comments) apportioned as follows: Dialog Controller procedural mechanisms (including IPSIM) 15%, Dialog Controller knowledge base 11%, Domain Processing procedural mechanisms 25%, Domain Processing knowledge base 14%, Linguistic Interface including much language generation code 30%, miscellaneous 5%.

The implemented domain processor was loaded with a model for a particular circuit assembled on a Radio Shack 160-in-One Electronic Project Kit. The model was complete enough to solve any problem of the circuit that involved missing wires. For example, if the system were asked to debug the circuit with no wires, it would systematically discover each missing wire and request that the user install it.

The speech output was done with a DECtalk (trademark of Digital Equipment Corp.) DTCO1 text-to-speech converter.

9. Testing the System

A reasonable test of the theory and implementation described here is to bring human subjects to the laboratory and determine whether they can converse sufficiently well with the machine to effectively solve problems. The purpose of the testing was to gather general statistics on system performance and timing, to study the effects of mode, and to judge the human factors issues, learnability, and user response. The hypotheses were that the system would function acceptably, that with a reasonable amount of user training, machine directive mode would yield longer completion times and less complex verbal behaviors than a more passive mode, and that users would respond positively to using the system. This section describes the design of the tests and the results obtained.

9.1 Experimental Design

Three experimental sessions were used for each subject. The first was to train the subject and register subject pronunciations on the Verbex machine. The second session was a data-gathering test in which the subject could attempt up to ten problems with the dialog system locked in either directive or declarative mode. The third session allowed the subject to attempt up to ten additional problems. It was similar to the second except that the system was placed in the mode that was not used in session 2 (either declarative or directive). Experimentation was thus to be limited to just two modes even though four were operative.

Eight subjects were recruited from computer science classes. They were selected on the criteria that they (1) have demonstrated problem-solving skills by having successfully completed a computer science course and having enrolled in another, (2) not have excessive familiarity with artificial intelligence or natural language processing as would occur, for example, if they had had a course on one of these topics, and (3) not be an electrical engineering major (in which case they could probably repair the circuits without aid). They were told they would receive $36.00 for participating in the three-part experiment. All selected subjects were used and all collected data are reported regardless of the level of success achieved.

Session 1 introduced the subjects to the voice equipment and required that they speak at least two examples of each of the 125 vocabulary words. They then were asked to speak 239 sentences to train the system for coarticulation. Repetitions in either of these sessions were used as needed to obtain acceptable recognition rates. Next the subjects were told about the dialog system and its functions and capabilities in brief and simple terms. They were given the basic rules on how to speak to the system, including the need for carefully enunciated speech, the requirement for verbie-over bracketing, the importance of hearing the acknowledging beep, and special requirements for stating numbers. They were told not to direct any comments to the experimenter; however, the experimenter would occasionally give them help, as will be described below. The subjects were asked to listen to and repeat four sentences spoken by the DECtalk system; this exercise was repeated until they overcame any difficulties in understanding. They were shown the target LED displays and given suggestions on how to successfully describe such displays to the system; specifically, the user should tell what they see present on the display (as in "the top of a seven is displaying") and not describe what does not appear (as in "the bottom of the seven is missing"). The subjects were provided with a list of the allowed vocabulary words and charts on a poster board suggesting implemented syntax if they wished to use it. Finally, they were given four practice problems and allowed to try solving them with the machine operating in directive mode. The complete session lasted up to two and one half hours.

Session 2 was scheduled for three or four days later. It began with a reorientation, 60 practice sentences on the speech recognizer, and some review questions on the general instructions. If this session was in directive mode, the subjects were told the system would act like a teacher and that they should follow its instructions. If this session was in declarative mode, they were told the system would act like an assistant so that they could control the dialog, and they were given an example of a short interaction so that they could observe the kind of control that can be achieved. Then they were released to do up to ten problems.

Session 3 was scheduled for three or four days after the second session. Appropriate instructions were given to change the subject expectations to the new mode, and ten more sample problems were given. Finally, the subjects were asked to fill in a short form and describe their reactions to using the system.

An important issue in such tests, as has been observed elsewhere (Biermann, Fineman, and Heidlage 1992), is the problem of giving the subject sufficient error messages to enable satisfactory progress. Users may wander aimlessly in their behaviors without some guidance when things go wrong. If the input speech is discrete with a pause after every word, an automatic system can confirm words individually and give the user adequate feedback (Biermann et al. 1985). But with connected speech, the system cannot easily pinpoint the source of errors and may not provide satisfactory guidance. The user may receive an unexpected response from the system and then speak again but with increased volume or nonstandard vocabulary; this may yield worse machine responses and even more extreme behavior from the user. Our answer to this problem was to post the experimenter nearby and to allow him or her to give the subject several different standard error messages if they were needed. The experimenter was allowed to deliver any of the following messages if certain criteria were met:

1. Due to misrecognition your words came out as ______. (Most misrecognitions were corrected automatically and thus resulted in no such messge. This message was given if the interpreted meaning contradicted the intended meaning or referenced the wrong object.)
2. Please be patient. The system is taking a long time to respond.
3. The system is ready for your next utterance. (Or other synchronization warnings.)
4. Please remember to start/end utterances with verbie/over.
5. Recognition is indicated by a beep.
6. The word ____ is not in the vocabulary.
7. (a number) must be spoken as digits.
8. Please restrict your utterances to one sentence.
9. Please keep your tone/volume/rhythm similar to the way you trained.
10. Please focus on interaction with the computer. (In case of a comment to the experimenter.)
11. Please follow the computer's guidance. (After three repetitions caused by subject's refusal to cooperate.)

Statistics were kept on the number of such messages that were delivered during the test sessions, as reported below.

9.2 Test Problems

The circuit to be repaired was a multivibrator circuit constructed on a Radio Shack 160 in One Project Kit. It contained twenty wires and used a number of components on the board: a switch, potentiometer, light-emitting diode (LED), battery, and two transistors. Its correct behavior was to alternately display a 1 and a 7 on the LED with the rate of alternation being adjustable by the potentiometer. For the purposes of the experiment, failures were introduced by removing one or two wires. The first eight problems for the two sessions were matched in difficulty as well as possible in order to give balance between sessions and to prevent a varying difficulty from overshadowing important effects. The last two problems were repeats of the practice problems from the first session.

9.3 Test Dialog System

The dialog system being tested was the version that was operative at the time of the test, mid February, 1991. At that time, it was running on a Sun 4 machine, which caused

Table 1
Experimental results for eight subjects operating at two levels of machine initiative; declarative and directive. Numbers in parentheses are for the first eight problems only; the last two problems were repeats of the practice problems from Session 1.

	Declarative Mode	Directive Mode
Number of problems attempted	75	66
Number of problems diagnosed	66	61
Number of problems completed	60	58
Average diagnosis time	199.9 (225.0)	258.9 (277.3)
Average completion time	270.6 (303.8)	511.3 (541.0)
Average number of utterances per dialog	10.7 (12.0)	27.6 (28.8)
Average subject response time (sec.)	17.0	11.8
Average number of utterances per minute	2.3	3.1
Percent of nontrivial utterances	62.9	39.7
Average length of nontrivial utterances	5.4	4.8
Number of different words used	100	84
Number of utterances	1011	1829
Number of experimenter interactions	174	100
Number of misrecognition interactions	118	69
Recognition rate (parser)	75.3	85.0
Recognition rate (Verbex)	44.3	53.1
Overall recognition rate (parser)	81.5	
Overall recognition rate (Verbex)	50.0	

significant problems with execution time. A few responses during the experiment were as slow as 10 seconds or in some cases as much as 30 seconds, which hampered the flow of the interaction. These slow responses were primarily due to the computational costs of parsing long utterances containing many misrecognized words. The system was later enhanced by moving it to Sparc-2 machine.

The ability of the system to respond to silence as a legitimate input was disabled because it had earlier confused our pilot subjects. If the user was silent for a period of time, the system patiently waited for his or her input. A small number of grammar omissions and other minor system shortcomings were noticed in the early subjects and fixed for later subjects.

Later versions of the system, such as the one on our demonstration tape (Hipp and Smith 1991), included some error message capabilities that were not available in the experiment. Those would have led to substantially better performance if they could have been used.

9.4 Results

The results from the experiment are summarized in Table 1.

Each entry in the table will be explained:

Number of problems attempted. Each subject had the opportunity to do 10 problems at each initiative level. Therefore if time constraints had not existed, a total of 80 problems would have been attempted in each mode. However, subjects progressed slowly enough so that the two-hour session (session 2 or 3) ended before all 10 were attempted in some cases.

Number of problems diagnosed. The diagnosis of a problem involved discovering what wires were missing. However, substantial more interaction was needed to correctly fix the problem and to test the circuit to be sure its complete functioning had been restored.

Number of problems completed. A dialog was completed when the machine was satisfied that the repair was complete and had signed off by saying "good-bye" to the user. Most of the failures to complete were simply the result of limited time. If a dialog exceeded 10 minutes without a successful diagnosis or 15 minutes without a successful completion, and there was no expectation that progress would occur in the next couple of minutes, the dialog was halted and scored as a failure. The percent of problems completed might have been nearly 100 percent if unlimited time had been available.

Average diagnosis time. The elapsed time from the beginning of the dialog to discovery of the missing wire(s).

Average completion time. The elapsed time from the beginning of the dialog to the completion of the task.

Average number of utterances per dialog. The average number of utterances by the subject per dialog.

Average subject response time. The average elapsed time (in seconds) from the computer's utterance to the subject's response. It is believed that this is a better indicator for the "speech rate" of a subject than average utterances per minute because the number of utterances per minute is dependent on the speed of the computer system as well, and long utterances by the subject required long response times for the computer.

Average number of utterances per minute. The average number of utterances per minute spoken by the subject on all dialogs, even those that were not completed.

Percent of nontrivial utterances. The percent of utterances spoken by the subject that included more than one word and the bracketing words (verbie and over).

Average length of nontrivial utterances. The average number of words in a nontrivial utterance by the subject not including the bracketing words.

Number of different words used. The total number of different words used by the subjects that were implemented in the system.

Number of utterances. The total number of utterances spoken by all subjects in all dialogs.

Number of experimenter interactions. The total number of times the experimenter spoke to a subject during the dialogs.

Number of misrecognition interactions. The number of experimenter interactions that were caused by misrecognitions. This figure shows how many of the interactions specified in the previous row were due to the single cause of misrecognition. The message in this case was "Due to misrecognition, your words came out as ...."

Recognition rate (parser). The percentage of subject utterances that were correctly interpreted by the parser after error correction and the use of expectation.

Recognition rate (Verbex). The percentage of subject utterances that were correctly recognized word-for word by the speech recognition system.

Overall recognition rate. The recognition rate obtained by combining data across both modes.

After subjects completed session 3, they were given a form to gather their reactions to the use of the system. Table 2 summarizes their answers on a scale from 1 (representing "strongly disagree") to 5 (representing "strongly agree"). Additional information about user responses, including their extensive remarks, is given in Smith (1991).

9.5 Test Data Analysis

9.5.1 General results. Overall system functioning was good. All subjects were able to use the system, and they solved the problems at the rates of 80 and 88 percent (that is 60 out of 75 trials and 58 out of 66 trials) in the declarative and directive

Table 2
User reactions to system use.

	Subjects Using Directive then Declarative Modes	Subjects Using Declarative then Directive Modes
Easy to learn to use	4.5	4.2
Enjoyed using	4.2	3.9
Easy to use	3.8	4.0
Tiring	3.8	4.0
System had too much control initially	3.0	2.6
System had too much control at end	2.5	4.1

modes, respectively. Most of the failures to finish were from time outs rather than any ultimate system failure. Subject speech rates were around two to three utterances per minute, but this might have risen to four to six sentences per minute if the system response time had been better. (This higher speech rate occurred in an earlier system built in this lab (Biermann, Fineman, and Heidlage 1992), and might also be possible with our current faster version of this system.) The powerful error correction from the nearest neighbor algorithm using expectation improved overall recognition from 50.0 to 81.5 percent. The number of experimenter interactions was about 1 for every 18 user utterances in the directive mode, and 1 for every 6 user utterances in the more complex declarative mode.

The effects of mode were clear and as predicted. During session 1, subjects had gained substantial information about how to debug this circuit, so the extremely pedantic directive mode was too detailed for them in most cases. They functioned much better in declarative mode, in which the machine stated relevant facts but allowed the user to lead the interaction. The completion times and numbers of utterances were far smaller for the declarative mode. In addition, the complexity of the sentences and the time required for the user to respond was larger for the declarative mode. The directive mode involves far more questions from the machine with only one-word answers, yielding faster responses and higher recognition rates. The directive mode requires far fewer experimenter interactions to help the subject through the task.

Users responded positively to the system and definitely noticed the effects of the mode change.

The posters for providing syntax assistance were not used extensively. Based on exit interviews with subjects, two of them did not use them at all during the formal experimental sessions (sessions 2 and 3). Three subjects used them only for a specific situation (such as notifying the system that the circuit was working), while the remaining three subjects used the charts for assistance with more than one class of utterances (such as voltage statements and LED description statements). However, even these subjects did not continually refer to the charts, but only used them infrequently.

Additional study of the data yielded some interesting information about the subjects' rate of learning of the system. One might wonder how users' behaviors improved with time. The total number of problems attempted in both modes in session 2 was 67, and this rose to 74 in session 3. The rate of success rose from 79 percent to 88 percent. It is also interesting to speculate how long it would be necessary to keep the experimenter at hand to give error messages. The number of experimenter interactions dropped from 162 in session 2 to 112 in session 3.

Table 3
The statistical results for the eight problems.

		Completion Time		Utterances		% Non Trivial	
Prob.	n	t	p value	t	p value	t	p value
1	5	.686	—	3.340	.05*	−3.619	.05*
2	5	−.018	—	1.170	.40	−3.267	.05*
3	4	1.547	.30	3.633	.05*	−3.441	.05*
4	4	2.971	.10	4.586	.02*	−3.118	.10
5	4	1.342	.30	4.557	.02*	−2.281	.20
6	7	3.136	.05*	4.516	.01**	−1.874	.20
7	5	2.145	.10	3.783	.02*	−2.665	.10
8	4	1.876	.20	3.235	.05*	−1.811	.20

Another phenomenon often noticed in the speech recognition world is the "sheep–goats" dichotomy that seems to divide users. Some users seem to function well with voice recognition equipment ("sheep") while others ("goats") have considerable difficulties. We noticed this in that four of our subjects were able to attempt all 80 problems given them and solved 79 of the 80. Their recognition rates tended to be in the high 80s or low 90s. The much lower reported average figures given above are the result of averaging in performance figures from the less successful class of users.

A final issue of interest is the level of software reliability achieveable by such a system. It has been observed in previous research systems that the code is sufficiently complex and the level of development in a research laboratory sufficiently incomplete that substantial numbers of software failures do occur in such tests. For example, Damerau (1981), Miller, Herschman, and Kelly (1978) on the LADDER system, and Biermann, Ballard, and Sigmon (1983) reported software failures at the rates of approximately 5, 2, and 2 percents, respectively. In the current test, of the 2,840 utterances that were processed, only one resulted in such a software failure. The reasons for the high reliability were (1) the quality of the code was high and (2) the nature of the design that finds a nearest neighbor to an input string among expected inputs is one in which high reliability is much more easily obtained.

9.5.2 Statistical Analysis. Although the summary results provide a strong indication that there are behavioral differences as a function of initiative, there are two major problems in demonstrating the statistical significance of these differences: (1) only four subjects completed a high enough percentage of the dialogs to enable a within-subject comparison of these differences across all problems, and (2) there were large sample variances in the data.

To alleviate these problems, behavioral differences were analyzed separately for each problem. This was possible because of the balancing of problems between sessions according to problem type. That is, problem "k" of each session had the same erroneous LED display. Consequently, the system's diagnosis procedure was virtually identical for problem "k" in each session. The only difference came in identifying the missing wire or wires that caused the circuit failure. The missing wires were different in each session.

Three performance criteria were examined: (1) completion time, (2) number of utterances spoken, and (3) percentage of nontrivial utterances. By examining the difference in a subject's performance on problem "k" as a function of initiative, a paired *t* test (Larsen and Marx 1981) could be conducted. The null hypothesis would say that the average of the paired differences is 0. Table 3 shows the results. For example, for problem 1, there were five subjects who completed the dialog in both sessions, and their paired difference for each criterion is obtained by the following formula:

"directive mode value" − "declarative mode value"

For example, the first subject spoke 18 utterances in directive mode for problem 1, but only 11 utterances when in declarative mode, a paired difference of +7. The set of paired differences for a given problem provides the sample data for the test statistic.

The *t* statistic and the individual observed significance levels are shown for each phenomena, one row per problem. In the results for number of utterances spoken, all problems but problem 2 showed an observed significance in user behavior as a function of initiative at the $p = 0.05$ level or less. There are also strong trends toward statistically significant behavioral differences with respect to completion time and nontrivial utterances for several of the problems. One difficulty with respect to completion time is the fact that the implementation of the parsing algorithm was not yet optimized during the experiment, and the more prevalent nontrivial utterances of declarative mode would require a longer time to parse. Note that while almost all of the significance levels support the hypothesis of a real difference, because the problems are interrelated, and the tested subjects overlap from problem to problem, we cannot claim overwhelming statistical significance. Nevertheless, we believe these results provide a strong indication that our system demonstrates variable initiative behavior, and that user behavior differs according to the level of initiative.

10. Theoretical Issues from the Literature

Grosz and Sidner (1986) have given a high-level theory of dialog. The theory specifies three components, the linguistic, intentional, and attentional structures, and describes their nature and relationships to each other. However, their theory leaves a whole variety of issues undetermined; without a full implementation, the question of its applicability remains unanswered. This project shows a successful instantiation of these ideas and provides some verification of their correctness. In fact, it presents a working speech dialog system that follows their theory in most of its details. In doing so, the project clarifies the nature of the three components and increases our understanding of them.

The linguistic component is the actual sequence of dialog interactions, and its most significant characteristic is that it can be naturally divided into what Grosz and Sidner call *segments*. (We have used the term "subdialog" instead of "segment" because we feel the latter term connotes contiguity, which may be misleading.) These segments are semantically coherent subparts of the dialog, and their union constitutes the whole dialog. An individual segment may be a contiguous sequence of interactions, or it may be broken into several sequences, as described in earlier sections of this paper. Many researchers have studied the properties of such segments, including linguistic and intonational markers to delineate them (Hirschberg and Litman 1993; Hirschberg and Pierrehumbert 1986), the resolution of noun phrases within the context of a segment (Grosz 1978; Reichman 1985), and their semantic self-consistency (Hobbs 1979). Our theory provides an automatic mechanism for participating in dialogs, for following segments initiated by the user, for initiating segments on its own, and for properly utilizing segment semantics for many language-processing purposes, such as noun phrase resolution, error correction in speech recognition, and so forth.

The intentional component specifies the purpose of the dialog. Grosz and Sidner (1986) use the notation DP and DSP to stand for "discourse purpose" and "discourse segment purpose." These entities correspond to the predicate goals that our system poses and then builds the dialog around. Our system, in fact, constructs a set of partial proofs and these are our instantiation of the intentional component. Grosz and Sidner introduce the relation of "dominance" between two DSPs; one DSP dominates another if the other "is intended to provide part of the satisfaction" of the first. This relation corresponds to the natural precedence in our system that a Horn clause proof gives to the predicates in the proof tree. Grosz and Sidner emphasize that the number of possible discourse purposes must be infinite in a dialog system, an assertion that we agree with. Our predicates allow for arbitrarily complex nesting of functional structures. As a trivial example, in our theory applied to the block-stacking world, the DSP buildtower (A on B on C on ... on X) is a legal goal regardless of how many blocks are specified to be stacked. Another issue they discuss is the possibility of multiple simultaneous goals. Thus the speaker may wish to both convey information and impress the hearer with his or her intelligence. This is a problem we have not attempted to deal with in our theory, and we do not offer any solutions to it. Our system deals with multiple goals but each subdialog addresses only one at a time.

The attentional structure is a stack in the theory of Grosz and Sidner and in many other theories as well (Litman and Allen 1987). The current incoming sentence is matched to local expectation, the entities at the top of the stack, and its meaning will be processed with respect to these structures in focus unless some kind of inconsistency occurs. If the sentence does not have meaning within this context, the stack may be pushed to introduce a new topic, or it may be popped and processing can occur within the next context in the priority. In our theory, the local expectation or focus is represented by the set of predicates that are related by our Prolog-style rules to the currently active predicate or DP. If no successful match is made, then other nonlocal but recently active DPs are tried.

The fact that our representation of local focus is a set of predicates on the proof tree (the intentional structure) may seem to contradict the Grosz–Sidner assertions that the focus-space hierarchy and the intentional structure should not be conflated. However, a closer look reveals that our focus stack exists independently of the intentional structure even though the items on the stack are listed on the intentional structure. The focus structure may be thought of as a stack of pointers; those pointers give addresses on the intentional structure, but the independence of the two entities is not compromised.

An example given by Grosz and Sidner (which in turn was borrowed from L. Polanyi and R. Scha) illustrates many of these points.

D1: John came by and left the groceries
D2: *Stop that you kids*
D3: and I put them away after he left

Figure 3 shows our representation of the interaction after the utterance D1. Two goals are currently active, DSP=describe prepare dinner and DSP=reduce stress; their partial proof trees are shown. The focus stack contains the description of getting food as its top entry, and previous topics have been pushed down to lower levels. The speaker's priority system, however, in the next instant has pushed the goal "reduce stress" into the foreground with the associated utterance D2, as shown in Figure 4. Here the stack receives an additional entry, a pointer to the control-children DSP. With the control-children DSP achieved, the focus stack can pop back to the previous interaction and continue, as shown in Figure 5. Actually, we deviate from Grosz and Sidner here and keep the recent subdialog active near the top of the stack. It may be returned to; for

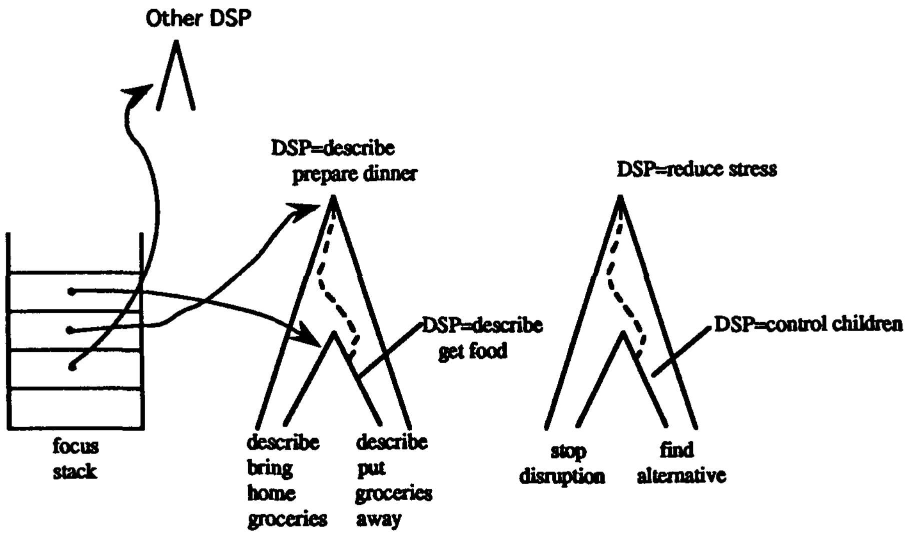


Figure 3
Intentional and attentional structures after "John came by and left the groceries."

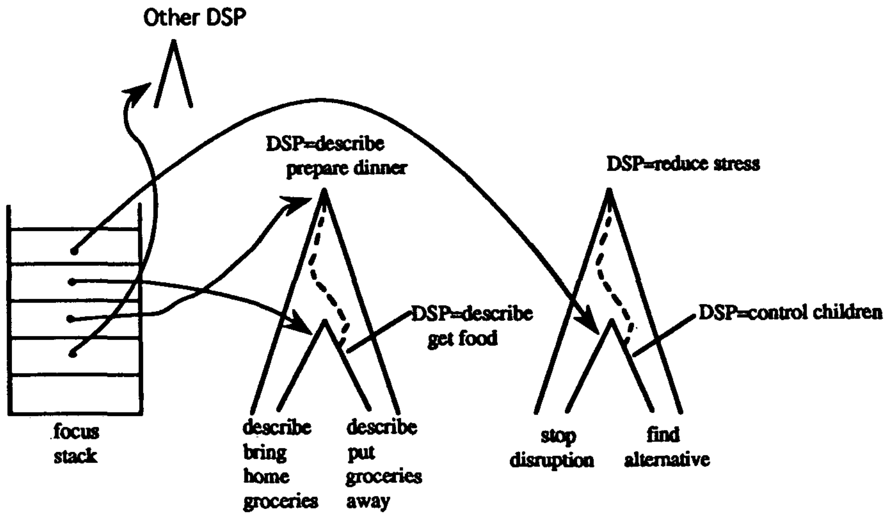


Figure 4
The speaker's priority intervenes to reduce stress: "Stop that you kids."

example, the next utterance could be

D4: Have you finished your homework?

We do this because we do not want our system to have to look far to find the appropriate referents in any recently active subdialog.

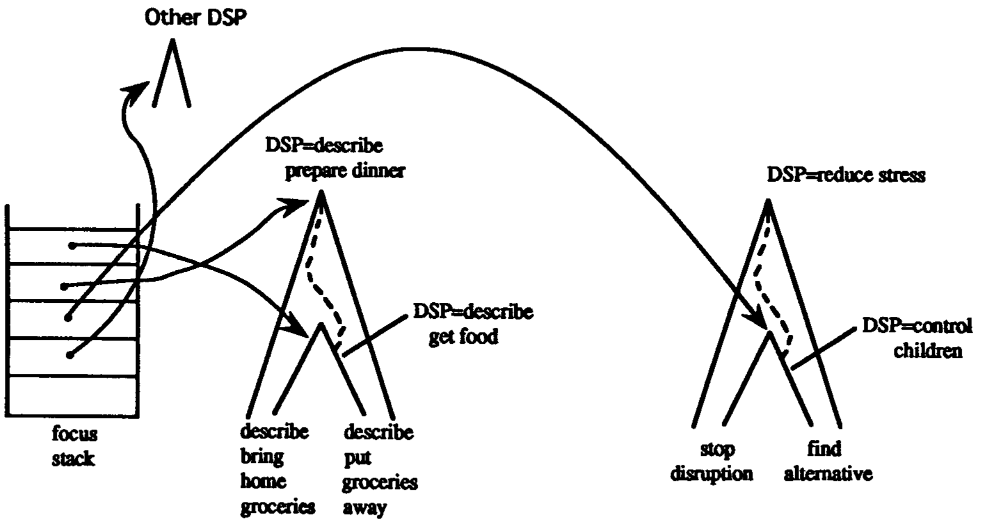


Figure 5
Returning to the DSP=describe get food intention: "and I put them away after he left." The recent focus on DSP=control children is retained on the focus stack.

Litman and Allen (1987) introduce the concept of a "discourse plan" that meshes with the domain plans of the traditional literature (Fikes and Nilsson 1971). The idea of discourse plans is that they operate on domain plans and act as meta operators. But they still are plans and are interpreted by the same mechanism as traditional planning systems. Discourse plans fit into three classes: the continue, clarification, and topic shift classes. As each incoming utterance arrives, one of these discourse plans comes into action to interpret that utterance in terms of the existing domain plans. A continue class discourse plan relates the utterance to the existing domain plan as a continuation of that domain plan. A clarification class plan relates the utterance to the existing domain plan as a clarification of that domain plan. A topic shift plan rejects any relationship to the current domain plan and introduces a new domain plan. Our system has corresponding behaviors, but they are coded into the interpreter for the Prolog-style rules. (Litman and Allen comment that the discourse plans are a small set and that they are domain-independent, so we find their inclusion in the interpreter to be a realistic choice.) Continuations and clarifications are handled as standard mechanisms of our interpreter. Topic shifts are handled by the mechanism that seeks matches of an incoming meaning with other subdialogs if the meaning has no local match.

Webber and Baldwin (1992) have examined the phenomenon called "context change" in discourse, which refers to the movement from subdialog to subdialog (or from segment to segment). Their discussion focuses on the concept of the *working set,* which specifies "the set of entities in the discourse context." As discourse proceeds, this working set is updated and it provides at each point in time the set of objects for definite noun phrase resolution. They describe two mechanisms for updating the working set; one, called context change by *entity introduction,* adds entities to the working set as they appear in the discourse. The other is called context change by *event simulation,* and it assumes that operators within the reasoning system add and delete items from the working set as a side effect of planning. In our system, the analog to the working set is the set of objects, relationships, and actions specified by the rules in the current subdialog. However, our system does not increment or decrement its working set in a gradual manner, one object at a time or one noun phrase at a time. When a subdialog is invoked, all of its objects, relationships, and actions come into focus at once; when the subdialog is abandoned, all of these entities are demoted to nonprimary but still near-at-hand availability. Our system does not typically change subdialog merely because an unexpected entity has been introduced. The specification of a new entity without any sentential context would probably confuse our system; it looks for utterances that will be meaningful in the current subdialog and it changes subdialog when an incoming utterance fails to have meaning locally but does have meaning in another available subdialog.

The semantic interpretation of sentences in realistic situations sometimes can be enabled by *abduction* mechanisms, as described by Hobbs et al. (1988). Abduction is needed when a literal interpretation of incoming tokens will not match the items or relationships in the existing database; it provides a mechanism to "coerce" a specific and literal meaning into a meaning that accounts for the sentence context. For example, Hobbs et al. (1988) address the issue of finding a referent for the implied fluid in the first sentence below.

Flow obstructed. Metal particles in lube oil filter.

According to Hobbs et al. (1988), the second sentence will require proving there exists x lube-oil (x), and the problem is to find a logical connection between the fluid of sentence one and the "lube oil." Their solution is to propose the identity

$$\forall x\ ((\text{fluid}\ (x) \wedge \text{etc}_1\ (x)) \equiv \text{lube-oil}\ (x))$$

where $\text{etc}_1(x)$ specifies a set of additional characteristics of x needed to ensure the identity. These properties $\text{etc}_1(x)$ need not be known; they are what Hobbs et al. call "assumable," and the act of making the assumption is the abductive act central to the "coercion" being carried out. This identity and the associated assumption make it possible to equate the fluid of the first sentence with the lube oil of the second sentence. The mechanism of our system achieves a similar result but via different means. Some currently active subdialog, which might or might not be at the highest level on the focus stack, would contain the lube oil and a variety of its characteristics and relationships. As incoming sentences arrive, their interpretations would be matched to the objects and relationships in the focus spaces using the minimum distance algorithm according to the stack priority until an acceptable match is found. In this example, there would be a focus space with an object "lube oil" that has the property of being a "fluid" that can "flow" and so forth. The words in the given sentences would naturally match to these without any special mechanisms. In summary, the abduction takes care of the situation where the semantic model is inadequate, and our design attempts to avoid such inadequacies.

A number of important contributions have been made in the area of user modeling (Kobsa and Wahlster 1988; Rich 1988; Paris 1988; McCoy 1988). One of the most comprehensive is the "General User Modeling Shell" (GUMS), described by Finin (1989). He lists five important features for user models:

(1) Separate knowledge base. The user model is kept as a separate module.

(2) Logical representation. The user model knowledge is encoded so as to be directly usable in inferential processes.

(3) Declarative knowledge. The user knowledge should be in a declarative rather than procedural form.

(4) Support for abstraction. The user knowledge should be able to represent generalizations. For example, it should be possible to represent information about classes of users as well as individuals.

(5) Multiple use. The user model knowledge should be represented in a form such that it can be reasoned about as well as reasoned with.

An important feature of this model is a hierarchical structure for stereotypes about users. Each level in the structure assumes a set of "definite" knowledge and a set of "default" knowledge. The given user in an interaction is placed somewhere on the hierarchy according to the best information. Then, as new information is gathered, he or she can be moved to a more general level (less definite information) if the user's knowledge is in contradiction with the required definite knowledge of that level.

In addition to a user's stereotype, an individual model of the user will also be developed. As specific information is acquired about the user, the system will begin to rely more on the individual model and less on the stereotype model.

Our system supports the first three features described by Finin. These are side effects of the Prolog-style representation. We could easily support the ideas related to classes of users and the hierarchy of stereotypes, but they were not addressed in this project.

We do not know of any systems that claim to support nontrivial variable initiative as described in our system. However, there is some literature on the topic. Whittaker and Stenton (1988) propose a definition of dialog control based on the utterance type of the speaker: question, assertion, command, prompt. In the first three, the speaker usually has control unless the utterance is a direct response to a question. In the last, control is usually being relinquished. They note that there is a close relationship between topic shift and control shift, and thus dialog control is useful in identifying the discourse structure.

Kitano and Van Ess-Dykema (1991) extend the plan recognition model of Litman and Allen (1987) to consider variable initiative dialog. They note that the two participants may have different domain plans. Consequently, there must be speaker-specific plans as well as the joint plans proposed in the Litman and Allen model. This separation of plans enables a more flexible plan recognition process. In addition, they extend the initiative control rules of Whittaker and Stenton to consider utterance content. They note that when a speaker makes an utterance that is relevant to his or her speaker-specific domain plan, then that speaker has dialog control. Their observation that there are speaker-specific plans or goals is crucial to our proposed model for participating in variable initiative dialogs. Selecting the next subdialog to be entered may require selecting between differing computer and user goals. This selection process must be a function of the current level of dialog initiative.

11. Summary

A voice-interactive dialog architecture that achieves simultaneously a variety of behaviors believed to be necessary for efficient human-machine dialog has been developed. Goal-oriented behavior is supplied by the theorem-proving paradigm, and the missing axiom theory provides the machinery for voice interactions. Subdialogs and movement between them is implemented with an interruptible theorem prover that maintains a set of partially completed proofs and can work on the most appropriate one at any given time. A user model is provided by a continuously changing set of rules that are referenced during theorem proving either to enable or to inhibit voice interaction. Variable initiative is made possible by variable types of processing by the input and output routines and by restricting or releasing the ability to interrupt to a new subdialog. Expectation is associated with individual subdialogs, is compiled from domain and dialog information related to each specific output, and is used to improve voice recognition and enable movement between subdialogs.

While this system simultaneously achieves behaviors that have been studied individually by a variety of earlier researchers, it cannot be reasonably characterized as an amalgamation of other systems. The behaviors are achieved by a new mechanism, namely the system-controlled Prolog-style inference machine, and the contribution of the work is in the demonstration that these behaviors can be done with this mechanism and in its description of how to do it. The contribution of other researchers has been to investigate the individual issues related to achieving dialog so that as we faced them, we had ideas about what to do using our mechanisms.

The implementation of the system shows the viability of the architecture and provides information regarding achievable behaviors in real-time voice dialogs. Eight subjects were able to solve 84 percent of attempted problems and reported a positive experience in doing it.

A number of important unsolved problems need attention to enable further progress. For example, an automatic mechanism is needed for setting the appropriate level of initiative as an interaction proceeds. Also, techniques are required that will enable the system to better diagnose user problems so that adequate messages can be provided automatically rather than depending on the experimenter. And the system, as it stands, can probably be improved greatly by increasing its knowledge base and refining its mechanisms.

The dialog theory described here has been aimed at implementing voice interactions. However, the theory could be used in other contexts using menus and multimedia or other interaction styles. This is another area for future work.

Appendix A: Trace of ZmodSubdialog Processing the 22-Sentence Example Dialog from Section 3

```
 1.  1   Retrieve goal from dialog controller
               T1_circuit_Test2(V)
 2.  1   Enter subdialog with that goal
 3.  2         Create subdialog data structures
 4.  2         Grab next available rule from knowledge
                    T1_circuit_Test_2(V) ← set(knob,10),measurevoltage(121,34,V)
 5.  2         Unify with current goal
 6.  2         This is a general rule
 7.  2         Store its right hand side as goals
 8.  2         Grab next one and enter a subdialog
                    set (knob,10)
 9.  2         Enter subdialog with that goal
10.  3              Create subdialog data structures
11.  3              Grab next available rule from knowledge
                         set(knob,Y)←find(knob),adjust(knob,Y)
12.  3              Unify with current goal
```

```
13. 3        This is a general rule
14. 3        Store its right hand side goals
15. 3        Grab next one and enter a subdialog
                find(knob)
16. 3        Enter subdialog with that goal
17. 4           Create subdialog data structures
18. 4           Grab next available rule from knowledge
                   find(knob)
19. 4           Unify with current goal
20. 4           Trivially answers question
21. 4           Exit subdialog
    Current rule: set(knob,Y)←find(knob),adjust(knob,Y)
22. 3        Grab next right hand side and enter a subdialog
                adjust(knob,10)
23. 3        Enter subdialog with that goal
24. 4           Create subdialog data structures
25. 4           Grab next available rule from knowledge
                   Y ← usercan(Y),vocalize(Y)
26. 4           Unify with current goal
27. 4           This is a general rule
28. 4           Store its right hand side goals
29. 4           Grab next one and enter a subdialog
                   usercan(adjust(knob,10))
30. 4           Enter subdialog with that goal
31. 5              Create subdialog data structures
32. 5              Grab next available rule from knowledge
                      usercan(adjust(knob,X))
33. 5              Unify with current goal
34. 5              Trivially answers question
35. 5              Exit subdialog
    Current rule: Y←usercan(Y),vocalize(Y)
36. 4           Grab next right hand side and enter a subdialog
                   vocalize(adjust(knob,10))
37. 4           Enter subdialog with that goal
38. 5              Create subdialog data structures
39. 5              Grab next available rule from knowledge
                      vocalize(Y)
40. 5              Unify with current goal
41. 5              Goal is to vocalize
42. 5              Execute verbal output
                      Put the knob to one zero.
43. 5              Record expectation A
                      affirmative
                      negative
                      user put knob to 10
                      where is knob
                      etc.
44. 5              Receive response
                      OK.
```

```
45.  5                    Record implicit, explicit meanings for response
46.  5                    Transfer control depending on response
47.  5                    Successful response: Return
48.  5                    Exit subdialog
     Current rule: Y ← usercan(Y),vocalize(Y)
49.  4                Exit subdialog
     Current rule: set(knob, Y) ← find(knob),adjust(knob, Y)
50.  3             Exit subdialog
     Current rule: T1_circuit_Test2(V) ← set(knob,10), measurevoltage(121, 34, V)
```

And so forth. The rest of the 330-step trace for processing the example 22-sentence dialog is available from the third author on request.

Acknowledgments

We would like to acknowledge collaboration with Robert Rodman during the early years of this project. In particular, we acknowledge his student Robin Gambill, who wrote the sentence-generation software for the system. Dania Egedi wrote some of the grammars for recognition and helped with the subject testing. Ruth S. Day advised us on the design of the human factors testing, and Robert M. Hoekstra suggested the statistical tests. The National Science Foundation supported the work under Grants IRI 88-03802 and IRI 92-21842. Duke University supported the work under a special grant authorized by Dr. Charles E. Putman.

We are very appreciative of the careful reading and suggestions provided by anonymous reviewers of this paper.

References

Aho, Alfred V., and Peterson, Thomas G. (1972). "A minimum distance error correcting parser for context free languages." *SIAM Journal on Computing*, 1(4), 305–312.

Aho, Alfred V., and Ullman, Jeffrey D. (1969). "Properties of syntax directed translations." *Journal of Computer and System Sciences*, 3(3), 319–334.

Allen, James F. (1983). "Recognizing intentions from natural language utterances." In *Computational Models of Discourse*, edited by M. Brady and R. C. Berwick., 107–166. MIT Press.

Allen, James F., and Perrault, C. Raymond (1980). "Analyzing intention in dialogues." *Artificial Intelligence*, 15(3), 143–178.

Allen, James; Guez, Stephen; Hoebel, Louis; Hinkelman, Elizabeth; Jackson, Keri; Kyburg, Alice; and Traum, David (1989). "The discourse system project." Technical Report 317, University of Rochester, New York.

Biermann, Alan W.; Ballard, Bruce W.; and Sigmon, Anne H. (1983). "An experimental study of natural language programming." *International Journal of Man-Machine Studies*, 18, 71–87.

Biermann, Alan W.; Fineman, Linda; and Heidlage, John F. (1992). "A voice- and touch-driven natural language editor and its performance." *International Journal of Man-Machine Studies*, 37, 1–21.

Biermann, Alan W.; Rodman, Robert; Rubin, David; and Heidlage, John F. (1985). "Natural language with discrete speech as a mode for human to machine communication." *Communication of the ACM*, 18(6), 628–636.

Carberry, Sandra (1988). "Modeling the user's plans and goals." *Computational Linguistics*, 14(3), 23–37.

Carberry, Sandra (1990). *Plan Recognition in Natural Language Dialogue*. MIT Press.

Carbonnel, N., and Pierrel, J. M. (1988). "Task-oriented dialogue processing in human–computer voice communication." In *Recent Advances in Speech Understanding and Dialog Systems*, edited by H. Niemann, M. Lang, and G. Sagerer, 74–107. Springer-Verlag.

Chin, David N. (1989). "KNOME: Modeling what the user knows in UC." In *User Models in Dialog Systems*, edited by Alfred Kobsa and Wolfgang Wahlster, 74–107. Springer-Verlag.

Cohen, Robin, and Jones, Marlene (1989). "Incorporating user models into expert systems for educational diagnosis." In *User Models in Dialog Systems*, edited by Alfred Kobsa and Wolfgang Wahlster, 313–333. Springer-Verlag.

Damerau, Fred J. (1981). "Operating statistics for the transformational question answering system." *Computational Linguistics*, 7(1), 30–42.

Erman, Lee D.; Hayes-Roth, Frederick; Lesser, Victor R.; and Reddy, D. Raj (1980). "The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty." *ACM Computing Surveys*, 12, 213–253.

Fikes, Richard E., and Nilsson, Nils (1971). "STRIPS: A new approach to the application of theorem proving to problem solving." *Artificial Intelligence 2*, 2(3/4), 189–208.

Fink, Pamela E., and Biermann, Alan W. (1986). "The correction of ill-formed input using history-based expectation with applications to speech understanding." *Computational Linguistics*, 12(1), 13–36.

Finin, Timothy W. (1989). "GUMS: A General User Modeling Shell." In *User Models in Dialog Systems*, edited by Alfred Kobsa and Wolfgang Wahlster, 411–430. Springer-Verlag.

Grosz, Barbara J. (1978). "Discourse analysis." In *Understanding Spoken Language*, edited by D. E. Walker, 235–268. North Holland.

Grosz, Barbara J., and Sidner, Candace L. (1986). "Attentions, intentions, and the structure of discourse." *Computational Linguistics*, 12(3), 175–204.

Hipp, D. Richard (1992). A new technique for parsing ill-formed spoken natural-language dialog. Doctoral dissertation, Duke University, Durham, North Carolina.

Hipp, D. Richard, and Smith, Ronnie W. (1991). *A demonstration of the "Circuit Fix-It Shoppe"* [Twelve-minute videotape]. Department of Computer Science, Duke University, Durham, North Carolina.

Hirschberg, Julia, and Litman, Diane (1993). "Empirical studies on the disambiguation of cue phrases." *Computational Linguistics*, 19(3), 501–530.

Hirschberg, Julia, and Pierrehumbert, Janet B. (1986). "The intonational structuring of discourse." In *Proceedings, 24th Annual Meeting of the Association for Computational Linguistics*. New York. 136–144.

Hobbs, Jerry R. (1979). "Coherence and coreference." *Cognitive Science*, 3, 67–90.

Hobbs, Jerry R.; Stickel, Mark; Martin, Paul; and Edwards, Douglas (1988). "Interpretation as abduction." In *Proceedings, 26th Annual Meeting of the Association for Computational Linguistics*. 95–103.

Kautz, Henry A. (1991). "A formal theory of plan recognition and its implementation." In *Reasoning about Plans*, edited by J. F. Allen, H. A. Kautz, R. N. Pelavin, and J. D. Tenenberg, 69–125. Morgan Kaufmann.

Kitano, Hiroaki, and Van Ess-Dykema, Carol (1991). "Toward a plan-based understanding model for mixed-initiative dialogues." In *Proceedings, 29th Annual Meeting of the Association for Computational Linguistics*. 25–32.

Kobsa, Alfred, and Wahlster, Wolfgang, editors. (1988). Special Issue on User Modelling. *Computational Linguistics*, 14(3).

Kobsa, Alfred, and Wahlster, Wolfgang (1989). *User Models in Dialog Systems.* Springer-Verlag.

Larsen, Richard J., and Marx, Morris L. (1981). *An Introduction to Mathematical Statistics and Its Applications.* Prentice-Hall.

Lehman, Jill Fain, and Carbonell, Jaime G. (1989). "Learning the user's language: A step towards automated creation of user models." In *User Models in Dialog Systems*, edited by Alfred Kobsa and Wolfgang Wahlster, 163–194. Springer-Verlag.

Levinson, Steven E. (1985). "Structural methods in automatic speech recognition." In *Proceedings of the IEEE*, 73(11), 1625–1650.

Linde, Charlotte, and Goguen, J. A. (1978). "Structure of planning discourse." *J. Social Biol. Struct.*, 1, 219–251.

Litman, Diane J., and Allen, James F. (1987). "A plan recognition model for subdialogues in conversations." *Cognitive Science*, 11(2), 163–200.

Lyon, Gordon (1974). "Syntax-directed least errors analysis for context-free languages." *Communications of the ACM*, 17(1), 3–14.

McCoy, Kathleen F. (1988). "Reasoning on a highlighted user model to respond to misconceptions." *Computational Linguistics*, 14(3), 52–63.

Miller, H. G.; Hershman, R. L.; and Kelly, R. J. (1978). "Performance of a natural language query system in a simulated command control environment." U.S. Naval Report NOSC ACCAT.

Moody, Terence S. (1988). *The effects of restricted vocabulary size on voice interactive discourse structure.* Doctoral dissertation, North Carolina State University, Raleigh, NC.

Morik, Katharina (1989). "User models and conversational settings: Modeling the user's wants." In *User Models in Dialog Systems*, edited by Alfred Kobsa and Wolfgang Wahlster, 364–385. Springer-Verlag.

Mudler, J. and Paulus, E. (1988). "Expectation-based speech recognition." In *Recent Advances in Speech Understanding and Dialog Systems*, edited by H. Niemann,

M. Lang, and G. Sagerer, 473–477. Springer-Verlag.

Novick, David G. (1988). *Control of mixed-initiative discourse through meta-locutionary acts: A computational model.* Doctoral dissertation, University of Oregon.

Paris, Cecile L. (1988). "Tailoring object descriptions to a user's level of expertise." *Computational Linguistics*, 14(3), 64–78.

Polanyi, Livia, and Scha, Remko J. H. (1983). "On the recursive structure of discourse." In *Connectedness in Sentence, Discourse and Text*, edited by K. Ehlich and H. van Riemsdijk, 141–178. Tilburg University.

Pollack, Martha E. (1986). "A model of plan inference that distinguishes between the beliefs of factors and observers." In *Proceedings, 24th Annual Meeting of the Association for Computational Linguistics*, 207–214.

Reichman, Rachel (1985). *Getting Computers To Talk Like You and Me.* MIT Press.

Rich, Elaine (1989). "Stereotypes and User Modelling." In *User Models in Dialog Systems*, edited by Kobsa, Alfred, Wahlster, and Wolfgang. Springer-Verlag. 35–51.

Seneff, Stephanie (1992). "TINA: A natural language system for spoken language applications." *Computational Linguistics*, 18(1), 61–86.

Smith, Ronnie W. (1991). *A computational model of expectation-driven mixed-initiative dialog processing.* Doctoral dissertation, Duke University, Durham, NC.

Smith, Ronnie W., and Hipp, D. Richard (1994). *Spoken Natural Language Dialog Systems: A Practical Approach.* Oxford University Press.

Walker, Donald E. (1978). *Understanding Spoken Language.* North-Holland.

Walker, Marilyn, and Whittaker, Steve (1990). "Mixed initiative in dialogue: An investigation into discourse segmentation." In *Proceedings, 28th Annual Meeting of the Association for Computational Linguistics*, 70–78.

Webber, Bonnie L., and Baldwin, Breck (1992). "Accommodating context change." In *Proceedings, 30th Annual Meeting of the Association for Computational Linguistics*, 96–103.

Whittaker, Steve, and Stenton, Phil (1988). "Cues and control in expert-client dialogues." In *Proceedings, 26th Annual Meeting of the Association for Computational Linguistics*, 123–130.

Young, Sheryl R. (1990). "Use of dialogue, pragmatics and semantics to enhance speech recognition." *Speech Communication*, 9, 551–564.

Young, Sheryl R.; Hauptmann, Alexander G.; Ward, Wayne H.; Smith, Edward T.; and Werner, Philip (1989). "High level knowledge sources in usable speech recognition systems." *Communications of the ACM*, 32(2), 183–194.

Zue, Victor; Glass, James; Phillips, Michael; and Seneff, Stephanie (1989). "The MIT SUMMIT speech recognition system, a progress report." In *Proceedings, DARPA Speech and Natural Language Workshop.* Philadelphia, 21–23.