Speech-Graphics Dialogue Systems

Alan W. Biermann, Michael S. Fulkerson, Greg A. Keim
Duke University
{awb,msf,keim}@cs.duke.edu

1 A Theory of Dialogue

The central mechanism of a dialogue system must be a planner (Allen et al., 1994; Smith et al., 1995; Young et al., 1989) that seeks the dialogue goal and organizes all behaviors for that purpose. Our project uses a hybrid Prolog-like planner (Smith and Hipp, 1994) which first attempts to prove the top-most goal and then initiates interactions with the user when the proof cannot easily be achieved. Specifically, it attempts to discover key *missing axioms* in the proof that prevent its completion and that may be attainable with the help of the user. The purposes of the interaction are to gather the missing information and to eventually achieve the top-most goal.

Once the structure of the system is settled, a variety of desirable behaviors for realistic dialogue can be programmed. These include subdialogue behaviors, variable initiative, the ability to account for a user model, the use of expectation for error correction purposes, and the ability to handle multimedia input and output. Each of these is described in the following paragraphs.

1.1 Subdialogue behaviors

Traditional analyses of human-human dialogue decompose sequences into *segments* which are locally coherent and which individually address their own subgoals in the overall dialogue structure. (Hobbs, 1979; Reichman, 1985; Grosz and Sidner, 1986; Lochbaum, 1991). Such a segment is opened for a specific purpose, may involve a series of interactions between participants, and may be closed having successfully achieved the target subgoal. Such a segment may be interrupted for the purpose of achieving a new, locally discovered, subgoal or for approaching a different goal. It may also fail to achieve success and be abandoned. Typical dialogues involve repeatedly opening such segments, pursuing one subgoal, jumping to another, returning to a previous subgoal and so forth until the highest level goal is achieved or abandoned.

The Prolog-like proof tree enables this kind of behavior because the dialogue segments can be built around the explicit subgoals of the proof tree. Control for the search can be governed by the domain dependent characteristics of the subproofs. The ordinary Prolog depth first search is not used and, instead, control can pass from subgoal to subgoal to match the segmental behavior that is normal for such dialogues.

1.2 Variable initiative

The primary facility needed for variable initiative is the ability either to control the movements between subgoals (dialogue segments) or to release control and to follow the user's movements (Guinn, 1995; Kitano and Ess-Dykema, 1991; Novick, 1988; Walker and Whittaker, 1990). Controlling the movement requires that the system have domain information available to guide decisions concerning which directions may be good to take. Having made these decisions, the system then jumps to the associated subdialogs and follows its plan to completion. Releasing control to the other participant involves matching incoming utterances to expected interactions for the various available subgoals and following the user to subgoals where matches are found. This is called *plan recognition* in the literature and has been the object of much study (Allen and Perrault, 1980; Litman and Allen, 1987; Pollack, 1986; Carberry, 1988; Carberry, 1990). Mechanisms for both managing movement between subgoals and deciding when to release control to the other participant, including extensive analyses of their effectiveness, are given in (Smith and Hipp, 1994; Guinn, 1995; Guinn, 1996).

1.3 Accounting for the user model

Efficient dialogue requires that the knowledge and abilities of the other participant be accounted for (Kobsa and Wahlster, 1989). When the system pro-

vides information to the user, it is important that it present the new information at the appropriate level. If the system describes details that are already known to the user, he or she will become demoralized. If the system fails to give needed information, the user will cease to function effectively. The Prolog theorem proving system provides a natural means for encoding and using the user model without major additional mechanisms. The missing axiom discovery mechanism simply selects subdialogues for interaction at the levels in the proof tree where the user has knowledge and these levels are where the interaction occurs (Smith et al., 1995).

1.4 Expectation for the purposes of error correction

Because all interactions in a given subdialogue are occurring in the context of the associated subgoal, the actual vocabulary and syntax that are locally appropriate may be anticipated (Young et al., 1989). Thus, for example, if the system has asked the user to measure a certain voltage, the Prolog theorem proving tree will include locally the possibilities that the user has responded successfully (as "I read six volts"), that the user has asked for clarification ("at which terminal?"), that the user needs instruction ("how do I measure that?"), or that the user has failed to satisfy the request ("no"). The error correction mechanism looks for an expected input that has a low Hamming distance (weighted) from the actual recognized input and chooses the best match to the input that it will respond to. If the match does not exceed a specified threshold, the system could look for matches on other recent subdialogues to determine whether the user is attempting to move to another subject. If no match is found, the system could also ask for a repeat of the spoken input. In tests of the Circuit Fixit Shoppe, the Hamming distance algorithm alone corrected an utterance level error rate from 50 percent down to 18.5 percent in a series of 141 dialogues (Hipp, 1992; Smith and Gordon, 1996).

1.5 Multimedia input and output capabilities

The original version of this architecture envisioned only speech in and speech out as the communication media. Speech input (such as "The voltage is six volts") was translated to predicated form (such as `answer(measure(t17,t202,6))`) and turned over to the theorem proving mechanism. Similarly, outputs from Prolog were converted to strings of text that were enunciated by a speech synthesizer. In recent years, however, our project (Biermann and Long, 1996) has experimented with multimedia grammars that convert full multimedia communication to and from the internal predicate form. The following sections describe our method.

2 Multimedia Grammars

We designed a multimedia grammar made up of a series of operators that relate media syntax and semantics. Each operator accounts specifically for a syntactic item and simultaneously executes code in the semantic world which is appropriate for that syntax. For example, in a programming domain where one might refer to the lines of code on the screen, a useful operator is **line** which finds the set of all lines on the screen within the current region of focus. Other operators find other sets (associated with nouns), find subsets of sets (as with the adjective "capitalized"), select out individuals (as with an ordinal), specify relationships (as with containment), and call for changes on the screen (as with "delete"). An important characteristic of such operators is that their syntactic and semantic portions are specified by a general purpose language (C++) so that they can manipulate any media or semantic objects that the designer may address. While our prototype system has used only spoken and text English and graphical pointing (highlighting or arrows), the approach could conceivably involve full graphical capabilities, mechanical devices, or other input-output media. Our approach is in contrast with the methods of (Feiner and McKeown, 1993; Wahlster et al., 1993) where communications are split into several media and then those media are coordinated for presentation to the user. Other work on multimedia communication is surveyed in (Maybury, 1993).

The multimedia grammar is demonstrated in the generation of the phrase "the fifth character in this line" with highlighting of a specified line as given in Figure 2. The domain is Pascal tutoring and the phrase specifies a particular character that the system wishes to comment on. An example of this type of reference from the actual system is shown in Figure 1.

Such a grammar can be used either for generation or input. In the generation mode, the target meaning is known (it is a particular character "1" in the example) and a sequence of operators is to be found that can achieve the target meaning. The associated syntax becomes the output to be presented to the user ("the fifth character in this line" (with pointer) in the example). In the parsing mode, the target syntax is known and a sequence of operators is desired that can account for the syntax. These operators will then compute the meaning for the ut-

Operator	Syntax	Semantics	Complexity
line	line	[begin] [writln('Hello');] [end.]	C_{line}
this (with pointer)	this line (with pointer)	[writln('Hello');]	$C_{this\_pointer} * \log(n)$
in	in this line (with pointer)	writln('Hello');	C_{in}
character	character in this line (with pointer)	[w][r][i][t][l][n]...	C_{char}
ordinal	fifth character in this line (with pointer)	[l]	$C_{ord} * 5$
the	the fifth character in this line (with pointer)	l	C_{art}

Figure 2: The operator grammar generating syntax to select an item on the screen.

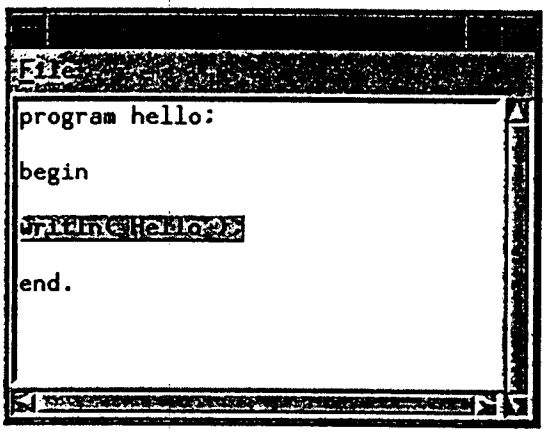

Figure 1: A screen from the Duke Programming Tutor: "There is an error at the fifth character in this line."

terance. Our project uses this grammar for output only because we have separately invested major efforts in the error correction system that has not been merged with the multimedia grammar.

This grammar, of course, has the ability to generate a variety of outputs: "this character" (with pointer), "the tenth character", "the fifth character in the second line", "this character in the second line" (with pointer), etc. A mechanism needs to be devised that will select among these choices and that will also prune the search to avoid unnecessary computation. Our system uses complexity numbers as shown in Figure 2 and seeks a minimum complexity utterance. The methodology is experimental and assigns a complexity constant to each operator. The pointer complexity is also multiplied by $\log(n)$ where n is the number of items that are being distinguished from. The intuition here is that a geometrical mechanism centers on the highlighted item. The ordinal is multiplied by v, the value of the ordinal, on the intuition that the user may actually count out the number specified by the ordinal. The actual values of the constants are obtained by training continuously as the user operates the system. This is explained in the next section.

3 Learning User Preferences

The complexity constants (C_{line}, C_{art}, etc.) for generation are learned and continuously updated during normal operation. The system gathers feedback from the user via any means the designer may choose and seeks a set of generation constants that optimize user satisfaction. In a test of the system (Biermann and Long, 1996), the feedback mechanism was simply the time required for the user to respond. A quick response was thus recorded as encouragement to continue the current type of generation and a long response acted to encourage the system to experiment with other values for the constants and seek a new optimum. The specifics of the learning algorithm employed are explained in (Long, 1996).

The graph in Figure 3 illustrates this process by tracking two of these constants through a sample run of the system. The system begins with a bias towards highlighting, evidenced by its lower relative value as compared to that of using ordinals. However,in the middle of the run, the user begins taking a long time to respond to the use of highlighting. Eventually, this drives the system to try ordinals, to which the user responds more quickly. This has the effect of lowering the constant for ordinals, thereby

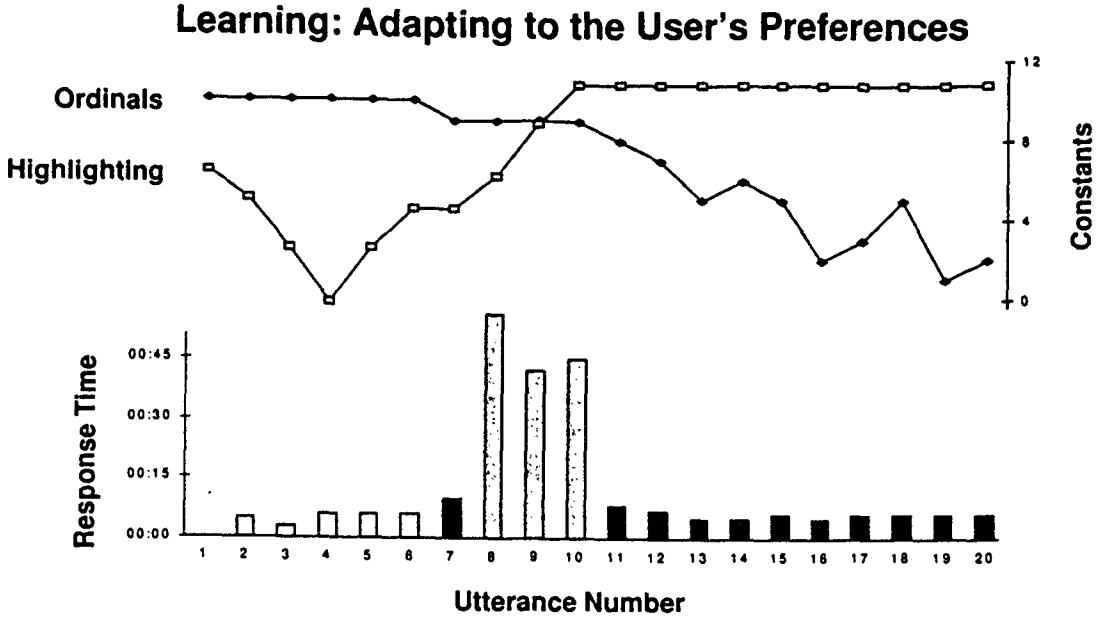

Figure 3: Adapting to the user's preferences.

making it the prefered output mode. Note that the algorithm also has an exploration parameter, which is the probability that it will choose a mode other than what is currently prefered. This allows the algorithm to periodically test modes that it might otherwise avoid, and explains why the system used ordinals for the seventh response, despite the higher constant.

4 Building Dialogue Systems

Our most recent voice dialogue system incorporates many of the ideas outlined above. The Duke Programming Tutor allows students in the introductory Computer Science course to write and debug simple programs, communicating with the system using voice, text and selection with the mouse. The system can respond with debugging or tutorial information, presented as a combination of speech, text and graphics. In the fall of 1996, 15 Duke undergraduates used the Duke Programming Tutor in place of their regular weekly lab. These sessions lasted about half an hour, and students received less than 3 minutes of instruction about how to use the system. For most students, this was only the second or third time they had debugged a program.

While constructing this system, we often wanted to add modules to explore new ideas: an animated face, the machine learning of the output mode preferences, a novel dialogue control algorithm, etc. As we struggled through integrating each of these new modules and discovering their dependencies on other parts of the existing system, we found ourselves wishing for a standardized framework—a communication and architectural infrastructure for voice dialogue systems. And while the CSLU Toolkit (Sutton et al., 1996) already promises rapid development of voice system applications, it and other commercial systems rely primarily on finite state models of dialogue, which may be insufficient for modeling complex domains or posing some research questions. A complete set of dialogue application programmer interfaces (APIs) would reduce system development time, lead to increased resource sharing and allow more accurate system and component evaluation (Fulkerson and Keim, 1997).

The high level architecture we envision uses *messages* to communicate content, and *events* to describe meta and control information. For example, SPEECHIN, a speech recognition module, might generate events such as *SpeechStart* or *SpeechStop*, and also produce a message containing a recognized utterance. This research effort will focus on understanding and formalizing these communication languages, so that they are not only powerful enough to capture the dialogue information we can obtain today, but also extensible enough to convey novel pieces of the human/machine interaction allowed by future developments. In order to test these ideas,

we are currently converting modules in our existing system to allow experimentation with various communication languages and architectures.

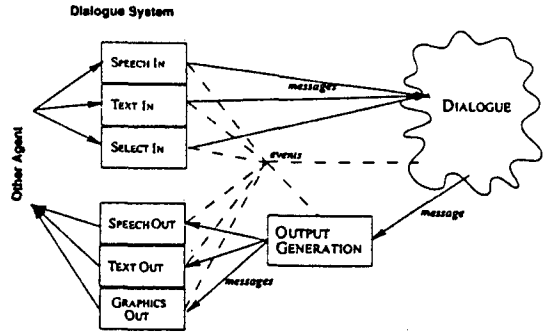

Figure 4: A multimodal dialogue system using messages and events.

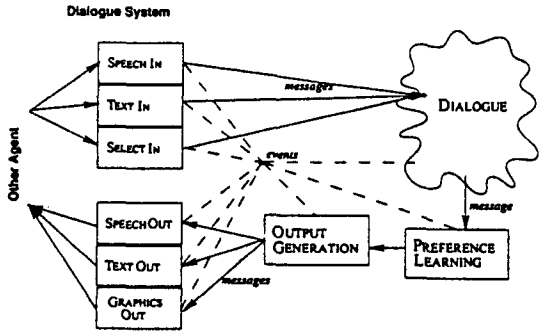

Figure 5: Adding learning module to an existing system.

Consider a hypothetical multimodal dialogue system that was constructed according to the above guidelines, as illustrated in Figure 4. The message output from dialogue processing contains a predicate form of some content to be communicated, and a set of modes in which this can be presented. The output generation module takes this message as input, and generates a response based on this information, choosing the mode randomly from what is allowed in the message. In many systems, adding a mechanism for learning the user's preferences might involve adding code to a number of modules. In the system we've just described however, the process is much easier. In Figure 5, we see that the learning algorithm can be inserted between the dialogue processing and output generation modules. It receives events generated by other modules, and uses timings between output and input events to calculate the user's response time. It then modifies the message from dialogue to allow only user's current prefered modes, and passes it on to output generation. Note that the API would not only make development faster and easier, it would also allow multiple learning algorithms to be tested in a particular domain. This type of component evaluation *within the context of a system* is currently much harder to accomplish.

5 Summary

We have discussed a mechanism for building dialogue systems, and how one might achieve useful behaviors, such as handling subdialogues, allowing variable initiative, accounting for user differences, correcting for errors, and communicating in a variety of modes. We discussed the Duke Programming Tutor, a system that demonstrates the integration of many of these ideas, which has been used by a number of students. Finally, we presented our ongoing project to make designing, constructing and evaluating new dialogue systems faster and easier.

6 Acknowledgements

This research is supported by the Office of Naval Research grant N00014-94-1-0938, the National Science Foundation grant IRI-92-21842 and a grant from the Research Triangle Institute, which is funded in part by the Army Research Office. Other individuals who have contributed to the Duke Programming Tutor include Curry Guinn, Zheng Liang, Phil Long, Douglas Melamed and Krishnan Rajagopalan.

References

J. F. Allen and C. R. Perrault. 1980. Analyzing intention in dialogues. *Artificial Intelligence*, 15(3):143–178.

James F. Allen, Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio, and David R. Traum. 1994. The TRAINS project: A case study in building a conversational planning agent. Technical Report TRAINS Technical Note 94-3, The University of Rochester, September.

A. W. Biermann and P. M. Long. 1996. The composition of messages in speech-graphics interactive systems. In *Proceedings of the 1996 International Symposium on Spoken Dialogue*, pages 97–100, October.

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

Sandra Carberry. 1990. *Plan recognition in natural language dialogue.* ACL-MIT Press series in natural language processing. MIT Press, Cambridge, Massachusetts.

S.K. Feiner and K.R. McKeown. 1993. Automating the generation of coordinated multimedia explanations. In M.T. Maybury, editor, *Intelligent Multimedia Interfaces*, pages 113–134. AAAI/MIT Press.

Michael F. Fulkerson and Greg A. Keim. 1997. Development of a component level API for voice dialogue systems. *In submission.*

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

Curry I. Guinn. 1995. *Meta-Dialogue Behaviors: Improving the Efficiency of Human-Machine Dialogue–A Computational Model of Variable Inititive and Negotiation in Collaborative Problem-Solving.* Ph.D. thesis, Duke University.

C. I. Guinn. 1996. Mechanisms for mixed-initiative human-computer collaborative discource. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 278–285.

D. R. Hipp. 1992. *A New Technique for Parsing Ill-formed Spoken Natural-language Dilaog.* Ph.D. thesis, Duke University.

J. R. Hobbs. 1979. Coherence and coreference. *Cognitive Science*, 3:67–90.

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

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

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

K.E. Lochbaum. 1991. An algorithm for plan recognition in collaborative discource. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 33–38.

P. M. Long. 1996. Improved bounds about on-line learning of smooth functions of a single variable. In *Proceedings of the 1996 Workshop on Algorithmic Learning Theory.*

M.T. Maybury, editor. 1993. *Intelligent Multimedia Interfaces.* AAAI/MIT Press.

D. G. Novick. 1988. *Control of Mixed-Initiative Discourse Through Meta-Locutionary Acts: A Computational Model.* Ph.D. thesis, University of Oregon.

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

R. Reichman. 1985. *Getting computers to talk like you and me.* The MIT Press, Cambridge, Mass.

Ronnie W. Smith and Steven A. Gordon. 1996. Pragmatic issues in handling miscommunication: Observations of a spoken natural language dialog system. In *AAAI Workshop on Detecting, Repairing, and Preventing Human–Machine Miscommunication in Portland, Oregan.*

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

Ronnie W. Smith, D. Richard Hipp, and Alan W. Biermann. 1995. An arctitecture for voice dialog systems based on prolog-style theorem proving. *Computational Linguistics*, 21(3):281–320, September.

Stephen Sutton, David G. Novick, Ronald Cole, Pieter Vermeulen, Jacques de Villiers, Johan Schalkwyk, and Mark Fanty. 1996. Building 10,000 spoken dialogue systems. In *Proceedings of the Fourth International Conference on Spoken Language Processing*, pages 709–712, October.

Wolfgang Wahlster, Elisabeeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, and Thomas Rist. 1993. Plan-based integration of natural language and graphics generation. *Artificial Intelligence*, 63:387–427.

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

S. R. Young, A. G. Hauptmann, W. H. Ward, E. T. Smith, and P. Werner. 1989. High level knowledge sources in usable speech recognition systems. *Communications of the ACM*, pages 183–194, August.