MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Evaluating the Contribution of Discourse Theory to an Interactive System

Charles Rich

TR93-18 September 1993

Abstract

An experiment is proposed which begins to systematically explore the potential contribution of discourse theory to the design of interactive systems. The experiment involves implementing and comparing two versions of a very simple system for planning air travel itineraries. One version of the system includes facilities based on discourse theory, the other does not. Neither system uses natural language. The long-term goal of this research is to develop a toolkit that can be used to add discourse capabilities to any interactive system.

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Mitsubishi Electric Research Laboratories, Inc.; an acknowledgment of the authors and individual contributions to the work; and all applicable portions of the copyright notice. Copying, reproduction, or republishing for any other purpose shall require a license with payment of fee to Mitsubishi Electric Research Laboratories, Inc. All rights reserved.

Copyright © Mitsubishi Electric Research Laboratories, Inc., 1993
201 Broadway, Cambridge, Massachusetts 02139

Publication History:–

1. First printing, MN93-18, Sept 1993

1 Introduction

Both intuition and anecdotal evidence suggest that the application of discourse theory has the potential of improving the design of interactive computer systems. This paper discusses some of the difficulties in evaluating this potential and proposes an implementation experiment that begins to explore the question in a more systematic way. By discourse theory, I mean to refer generally to the body of research that provides computational models of the extended communication between two or more agents in a shared context, and especially to work such as [Grosz and Sidner, 1990; Lochbaum *et al.*, 1990; Grosz and Kraus, 1993], in which an important part of the shared context is a collaborative task.

It seems intuitive in many situations to view an interactive computer system as an agent with which the user collaborates. From this point of view, I would argue that such systems will be easier for people, especially novices, to use if they obey the same conventions and rules of discourse that people are familiar with from everyday experience. (This is analogous to the argument that direct manipulation computer interfaces are successful largely because they take advantage of users' experience with manipulating the physical world.)

Some of the terminology of discourse theory has already made its way into the practice of designing interactive systems. It has recently become quite common (see, for example, the [Maybury, 1993] collection) to describe a system-user interaction in terms of communication acts such as *request*, *confirm*, *inform*, and so on. Unfortunately, it is very difficult to tell exactly which parts of discourse theory have been applied in these systems and what the contribution has been to the overall system usability.

The current state of the art in general-purpose tools for human-computer interaction is graphical user interface toolkits, such as Motif. While these toolkits provide a rich set of appearance options, they provide very shallow semantics for the dynamics of interaction.

The long-term goal of this research is to develop a general-purpose collection of subroutines, data structures, and specifications (i.e., a toolkit) that application programmers can easily use to add discourse capabilities such as the following to any interactive application:

- clarification of the current or an earlier communication action
- returning to a subtask which was abandoned or set aside (e.g., while more information was being gathered)
- mixed initiative (i.e., the system may sometimes suggest an action to be performed next)
- summary of the current state of the collaborative problem solving process

One of the biggest methodological problems in advancing toward the goal of a general-purpose discourse toolkit is the fact that, if you look at any particular application, improving the discourse capabilities of the system is hard to separate from improving the underlying application-specific competence or the graphical presentation. For example, when users complain that a document preparation system or VLSI design tool should be "more intelligent" or "easier to use," what they may really want most is a better algorithm for breaking pages or a better graphical presentation of design constraints. The methodological challenge is thus to demonstrate the benefit of applying discourse theory without getting bogged down in the specifics of a particular application. The implementation experiment proposed in the next section attempts to meet this challenge.

2 A Proposed Experiment

The basic idea of the experiment is to implement and compare two versions of a system. The *base* version of the system is a conventional interactive application. The *extended* version of the system has the same internal algorithms and graphical presentation, but adds capabilities for maintaining and using a representation of discourse context. (A few extra buttons may be added to the interface for discourse-specific functions.)

The task performed by the system should be simple enough so that a high degree of competence and a state-of-the-art graphical presentation can be implemented for the base system with minimal effort. The task should also be simple enough so that building a formal task model, which is required for the discourse extensions, is not overwhelming.[1]

The following are key features of this experiment:

- It is clear what is being held constant and what is being varied in the comparison between systems.
- It is clear what part of discourse theory is being applied and its interaction with the rest of the system (details in Section 2.3).
- Natural language understanding is not involved.

The last point above is important for both practical and theoretical reasons. As a practical matter, natural language understanding, even in a limited setting, is a very difficult problem in its own right. Including a natural language understanding

[1] A personal note. From my experience with research on an intelligent software assistant [Rich and Waters, 1990], I am particularly concerned with this point. It is very easy for the effort of formalizing a task/domain to overwhelm the other goals of the research, especially in the early stages.

module in the base system would push the implementation effort well beyond the minimal level.

From a theoretical point of view, if this form of experiment is successful (i.e., if the extended system is better than the base system), it will also help demonstrate that discourse theory addresses the *content* of collaborative communication at a very fundamental level—with respect to which English utterances, mouse clicks, or changes in the shape or color of an icon can simply be viewed as alternative presentations. This is an important step toward the goal of a discourse toolkit that can be used with any interactive application, i.e., not just those that involve natural language.

2.1 The Base System

The proposed task for the base system, namely planning air travel itineraries, is motivated mostly by the public availability[2] of a large transcribed corpus of telephone conversations between travel agents and customers. Careful analysis of this corpus has been extremely helpful for understanding how the general principles of discourse theory are instantiated in this particular task. This task also satisfies the requirements of minimal implementation effort for the base system and minimal task modeling effort.

The basic operation of the base system, shown in Figure 1, is very simple: a filter determined by user constraints such as desired airline, destination, arrival time, seat type, etc., is used to select a set of candidate flights from the database of all flights. (For multi-leg itineraries, the system can be thought of as a filter on the appropriately sized cross-product of flights.)

The architecture of the base system, shown in Figure 2, follows current good design practice by enforcing a clear separation between the application logic/state and the user interface. The application is in effect a simple state machine controlled through a direct manipulation interface. Many other systems, such as spreadsheets

[2]From Patti Price at SRI International.

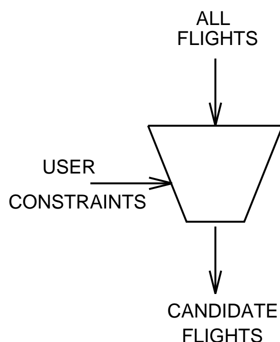


Figure 1. Basic operation of the base system.

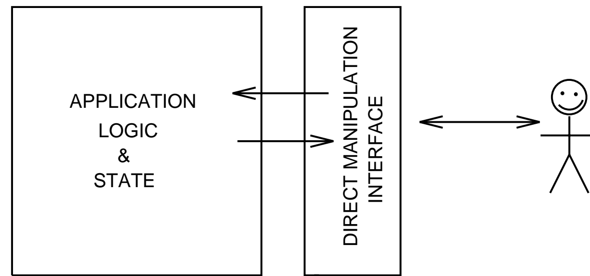


Figure 2. Architecture of the base system.

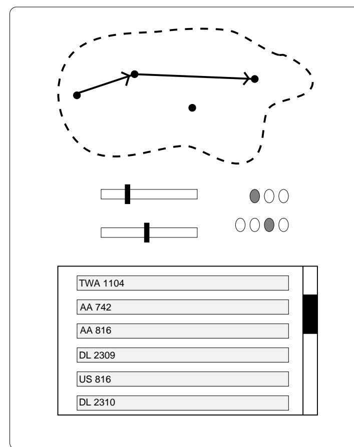


Figure 3. Presentation graphics of the base system.

and design tools, can be viewed as having this architecture. The direct manipulation interface will be implemented using a standard graphical toolkit.

A rough sketch of the base system's presentation graphics is shown in Figure 3. Again, the goal here is to apply the best current design practice. The top half of the screen is the direct manipulation interface to the user constraints. The complete state of the system, i.e., all the constraints, is visible and changeable using the appropriate graphic devices, such as a map for specifying cities, buttons for choices, sliders for ranges, and so on. The bottom half of the screen is a scrollable window that lists the candidate flights (itineraries), one per line. Some easily readable layout for each line, such as a color-coded horizontal time line, would be desirable.

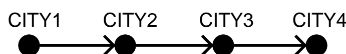
A typical scenario of system use would be as follows. The user first clicks on the map to specify the origin, connecting city, and final destination for a trip. A long list of itineraries immediately appears in the candidate window. She then modifies the default settings for various other constraints, such as desired airline and arrival times, thereby reducing the number of candidates to two. Finally, she decides which itinerary she likes best and books it (this final selection step is not supported by the system).

2.2 Limitations of the Base System

The system described above sounds pretty good at first blush—certainly a lot easier than using printed airline guides! However, I think users will be dissatisfied with it in a number of ways. Below are some guesses about the kinds of difficulties that will arise, based partly on analysis of the transcribed conversations between travel agents and customers.

The most obvious difficulty using the base system arises when the candidate window becomes empty, i.e., when there are no flights satisfying the current constraints. At this point, a natural thing for the user to do is to return to a previous state of the system in which the candidate window was not empty. (The alternative using the base system as described thus far is blind search, i.e., randomly changing constraints until some candidates appear.) A simple extension to the base system which seems to address this difficulty is to maintain a chronological history of all previous system states and to provide a direct manipulation interface, such as a stack of notecards, for choosing a previous state.

Unfortunately, this kind of backtracking often forces the user to throw away useful work that has already been done. To illustrate, consider a four-city itinerary, such as the following:[3]



Suppose you started entering constraints working forward from City1 to City2 to City3, and arrive at an impasse (empty candidate list) while entering the constraints on the third leg of the trip (City3 to City4). At this point suppose you want to retract the constraints on the first leg, but leave the constraints on the second leg alone. Even with a complete state history, there is no easy way to achieve this effect—you have to individually change each constraint on the first leg to its earlier value. The problem is that the system was never in the state you wish to go to. The solution to this problem, as we will see in the next section, is to represent not just the state of the base system, but also the state of the discourse.

Another difficulty with the base system is that the user constraints are all set independently. From looking at the transcribed conversations, it is clear that there are often dependencies between constraints. For example, in four-city itinerary above, if you enter 7 p.m. as the latest desired arrival time at City2, it follows that the earliest desired departure time (including the miminum allowance for changing planes) is 7:30 p.m. You should not have to separately enter this 7:30 constraint. Unfortunately, the dependency between these two constraints cannot simply be built into the base system, because the direction of the dependency varies according to whether you are working forward or backward. If you are working backward from City4, then the dependency goes the other way. The solution to this problem, as with the backtracking problem, lies in adding an explicit representation of the discourse state.

[3]This is chronological vs. dependency-directed backtracking ([Winston, 1992], Chapter 14).

2.3 Discourse Context

The central concept in discourse theory is the *discourse context*. In general, the discourse context includes intentional state, attentional state, and linguistic structure [Grosz and Sidner, 1986]. In this experiment, a representation of only the intentional part of the discourse context will be added to the base system, using a formalism based on *shared plans* [Grosz and Sidner, 1990].

A shared plan represents the state of a collaborative discourse in terms of relationships between the goals and actions of the agents involved. For example, there are basically two problem solving strategies (*recipes*) that people use for planning multi-leg air travel itineraries: working forward from the originating city or working backward from the final destination. Each of these recipes has the same subgoals (selecting a flight for each leg), but in a different order. The shared plan for this task specifies which recipe is being used, which subgoals have been achieved so far, and which user and system actions contributed to each subgoal.

The shared plan representation provides direct solutions to both of the difficulties described in Section 2.2. Since each constraint setting action is linked to the subgoal of selecting a flight for a particular leg, it is possible to retract the constraints related to any leg regardless of the chronological order in which the constraints were entered. In this way the system can backtrack to new combinations of partial previous states. The constraint dependency problem is solved by letting the recipe specified in the shared plan control which way the dependency goes.

The shared plan representation also begins to support the general capabilities outlined for a discourse toolkit in Section 1. For example, when a user returns after being interrupted in the middle of working on a complicated trip, an automatically generated description of the current shared plan might serve as a good summary of what was going on. To support mixed initiative, the system might suggest actions that will contribute to unachieved subgoals in the current recipe.

2.4 The Extended System

Figure 4 shows the architecture of the extended system. Dashed lines indicate the discourse extensions and relationships. There are two components in the discourse extensions: the discourse manager (the procedural component) and the discourse context (which contains an implementation of the shared plan representation discussed in the preceding section).

As can be seen in the figure, the application logic part of the base system needs to be modified to send to the discourse manager a description of each system action as it is performed. Similarly, the direct-manipulation interface needs to be modified to produce a discourse-level description of each user action as it is performed. The formalism for these action descriptions will be based on Sidner's artificial negotiation language [Sidner, 1992].

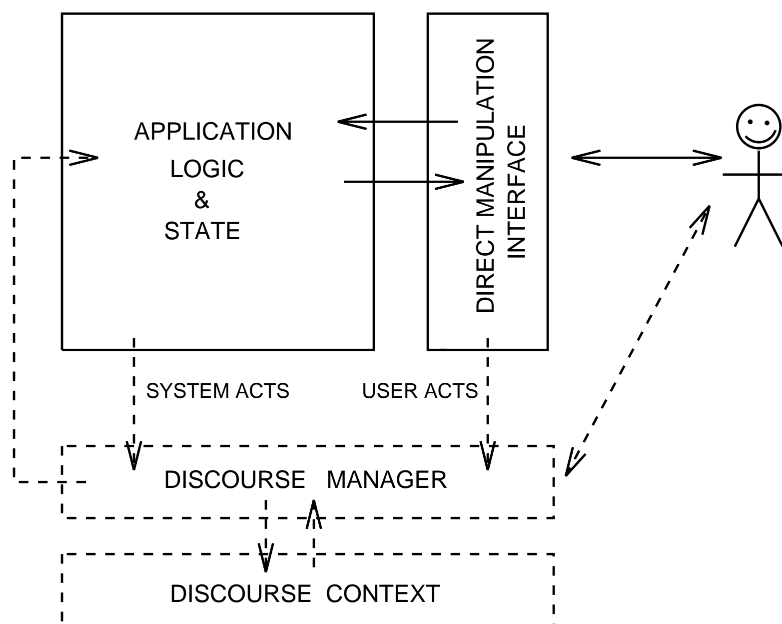


Figure 4. Architecture of extended system (extensions shown in dashed lines).

For example, when the user clicks on an icon representing a two-valued constraint, the interface should report that the user has proposed that the value of the constraint be reversed. When the application logic actually changes the value of the constraint, it should report a system action of accepting that proposal.

The discourse manager is responsible for updating the discourse context based on incoming information about system and user actions. The details of the operation of the discourse manager are somewhat sketchy at this point, since this is the component for which the most basic research needs to be done.

The discourse manager may also need to exchange information directly with the user. For example, the easiest way to establish which recipe is being used (e.g., forward vs. backward) would be to add a set of buttons to the interface labelled with the options.[4] The discourse manager may also need access to the user display to print out a summary of the current problem solving state.

Finally, the discourse manager needs to be able to update the state of the application, such as to automatically set constraints that depend on other constraints, as discussed above.

2.5 Evaluation

The evaluation part of the experiment has not yet been designed. The basic idea of the evaluation is to have a number of people use the two systems to plan itineraries

[4]The alternative is to try to recognize which recipe is being used by looking only at the actions performed by the user. This is *plan recognition* and is well-known to be very difficult in general.

given verbal descriptions such as the following:

> You are a Boston-based sales representative planning a trip to visit customers in Chicago, Denver, and San Francisco next week. You would prefer to leave on Wednesday morning, but can leave on Tuesday night, if necessary. Your customer in Denver is only available between 11 a.m. and 3 p.m. on Thursday. You would prefer to fly as much as possible on American Airlines, as you have almost enough frequent flier miles to qualify for a free trip this summer. You absolutely must be home by 5 p.m. on Friday in order to make it to your son's piano recital.

The base and extended systems can then be compared in terms of measures such as performance time, quality of solution, and user satisfaction.

As a practical matter, it makes sense to pursue the evaluation in two phases. In the first phase, which is informal and involves a small number of users, the goal will be mainly to get ideas about how to improve the extended system. If this first phase is promising, a second more formal phase may be undertaken with more careful controls and a larger number of subjects for statistical significance.

3 Toward a Discourse Toolkit

As well as being a convenient architecture for this experiment, Figure 4 might also serve as the framework for a discourse toolkit. To consider this possibility, we first need to shift our point of view from the end user of an interactive system to the application programmer who is the "user" of the toolkit during system development.

3.1 Modularity

From point of view of the application programmer, one of the most important properties of a toolkit is its modularity. Ideally, a discourse toolkit should provide a narrow, well-specified interface and should not interfere with the normal operation of the underlying application.

The conventional *undo* facility, despite its limitations (it only supports chronological backtracking), is a good example of modularity. The toolkit-level specification of *undo* simply provides a function to be called by the application whenever a reversible unit of action has been completed. (The programmer gets to decide the granularity of the reversible units.) The information passed from the application to the *undo* facility is simply a pointer to an application procedure that reverses the current action. All that the *undo* facility does is store the current procedure pointer (or perhaps a stack of them) and invoke the current procedure when requested by the user. A benefit of this kind of modularity from the end user's point of view is that the functionality of the *undo* command is consistent across applications.

A discourse toolkit should provide the same kind of modularity as *undo*, but for more powerful discourse-related capabilities. Figure 4 suggests that it may be possible to achieve this goal by sprinkling calls to the discourse manager throughout the application logic and direct manipulation interface code, without otherwise interfering with the operation of the application. The experimental implementation of the extended air travel system will shed some light on how well this idea works out in practice.

3.2 Task Modeling

In order to make use of a discourse toolkit, an application programmer will have to build a formal model of the collaborative task being performed by the system and user. Discourse theory provides certain metalevel operators, such as *propose*, *acknowledge*, *accept*, *reject*, which can be built into the language of a toolkit. However, the content of what is being proposed, acknowledged, etc., depends on the specific task being performed. For example, in the air travel task above, the content of a proposal might be "select TWA as the airline of the flight from Boston to Chicago."

Representing this task-specific content requires building up a formal vocabulary of:

- objects (cities and flights),
- relationships (flights have an airline attribute),
- actions (select an airline for the flight from Boston to Chicago),
- goals (plan an itinerary from Boston to San Francisco), and
- recipes (working forward).

The air travel task in this paper has intentionally been restricted in order to make this formal model easy to construct. For a real application, constructing a formal task model can be quite demanding, especially for the average application programmer, who has little experience with modeling or specification technology of any kind.

Defining objects and relationships as needed above is very similar to the kind of data modeling that goes on in modern data base systems. Defining the actions in a task model is quite similar to specifying software procedures. The goals and recipes part of a task model is more abstract than what is usually formalized in current software practice, except for in expert or knowledge-based systems.

On the one hand, task modeling can be thought of as an unfortunate hidden cost of applying discourse theory. On the other hand, the need for an explicit task model should be no surprise. From an AI point of view, what the task model does is add a measure of reflection—"self-awareness," if you like—to a system. Reflection is a well-known technique for improving the performance of a problem-solving system.

From a software engineering point of view, the task model can be thought of as part of the general trend towards improving software by making more of the programmer's design intentions explicit.

It should also be noted that the task model constructed for a discourse toolkit does not have to be as complete as it would have to be if the whole application were being implemented using a goal-directed AI problem solver. From the toolkit's point of view, how the application decides what action to perform next is a "black box." All the toolkit needs to do is to represent the results of that decision.

4 Conclusion

If the extended system described above does in fact turn out to support more flexible and fluent interaction than the base system, I hope it will encourage further effort toward applying discourse theory to the design of interactive systems. A first step in this direction would be to abstract and generalize the mechanisms developed in the experimental system to the class of applications that share the same architecture. Research might then move on to a more generally applicable discourse toolkit.

Acknowledgement

Many of the ideas in this paper arose from discussions with Candace L. Sidner.

References

[Grosz and Kraus, 1993] B. J. Grosz and S. Kraus. Collaborative plans for group activities. In *Proc. 13th Int. Joint Conf. Artificial Intelligence*, Chambery, France, 1993.

[Grosz and Sidner, 1986] B. J. Grosz and C. L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.

[Grosz and Sidner, 1990] B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions and Communication*, chapter 20, pages 417–444. MIT Press, Cambridge, MA, 1990.

[Lochbaum *et al.*, 1990] K. E. Lochbaum, B. J. Grosz, and C. L. Sidner. Models of plans to support communication: An initial report. In *Proc. 8th National Conf. on Artificial Intelligence*, pages 485–490, Boston, MA, July 1990.

[Maybury, 1993] M. T. Maybury, editor. *Intelligent Multimedia Interfaces*. AAAI Press, Menlo Park, CA, 1993.

[Rich and Waters, 1990] C. Rich and R. C. Waters. *The Programmer's Apprentice.* Addison-Wesley, Reading, MA and ACM Press, Baltimore, MD, 1990.

[Sidner, 1992] C. L. Sidner. Using discourse to negotiate in collaborative activity: An artificial language. In E. Simoudis, editor, *AAAI-92 Workshop on Cooperation Among Heterogenous Agents*, San Jose, CA, July 1992.

[Winston, 1992] P. H. Winston. *Artificial Intelligence, Third Edition.* Addison-Wesley, Reading, MA, 1992.