

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

## **Adding a Collaborative Agent to Graphical User Interfaces**

Charles Rich, Candace L. Sidner

TR96-11 May 1996

### **Abstract**

We have implemented a collaborative agent toolkit called Collagen and used it to build a software agent that collaborates with the user of a direct-manipulation graphical interface by following the rules and conventions of human discourse. One of the main results is an interaction history that is segmented according to the structure of the agents' and users' goals, without requiring the agent to understand natural language. Superseded by TR97-21.

*Ninth ACM Symposium on User Interface Software and Technology, Seattle, WA, November, 1997, pp. 21-30.*

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., 1996  
201 Broadway, Cambridge, Massachusetts 02139



## Adding a Collaborative Agent to Graphical User Interfaces

Charles Rich      Candace L. Sidner\*

TR-96-11    May 1996

### Abstract

We have implemented a collaborative agent toolkit called Collagen and used it to build a software agent that collaborates with the user of a direct-manipulation graphical interface by following the rules and conventions of human discourse. One of the main results is an interaction history that is segmented according to the structure of the agent's and user's goals, without requiring the agent to understand natural language.

*To appear in Ninth Annual Symposium on User Interface Software  
and Technology (UIST'96), Seattle, WA, November 1996.*

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 Information Technology Center America; 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 Information Technology Center America. All rights reserved.

Copyright © Mitsubishi Electric Information Technology Center America, 1996  
201 Broadway, Cambridge, Massachusetts 02139

---

\*Lotus Development Corporation

**Publication History:-**

1. First printing, TR-96-11, May 1996

# Adding a Collaborative Agent to Graphical User Interfaces

*Charles Rich*  
MERL—A Mitsubishi Electric  
Research Laboratory  
201 Broadway  
Cambridge, MA 02139 USA  
+1-617-621-7507  
rich@merl.com

*Candace L. Sidner*  
Lotus Development Corporation  
55 Cambridge Parkway  
Cambridge, MA 02142 USA  
+1-617-693-7737  
csidner@lotus.com

## ABSTRACT

We have implemented a collaborative agent toolkit called Collagen and used it to build a software agent that collaborates with the user of a direct-manipulation graphical interface by following the rules and conventions of human discourse. One of the main results is an interaction history that is segmented according to the structure of the agent's and user's goals, without requiring the agent to understand natural language.

**KEYWORDS:** Agent, collaboration, discourse, window sharing, direct manipulation, SharedPlan

## 1 INTRODUCTION

Current interactive systems can be difficult to use: the order in which things must be done is often inflexible, it's hard to recover from mistakes, and each system has its own interaction conventions. Although many factors contribute to these difficulties, we believe that the essence of the problem is not in the user interface as viewed over a single interaction, but rather in the lack of support for the user's problem solving *process*, especially over extended periods of time. The overall goal of this research is to therefore to support this process by applying principles of human collaboration and discourse.

One of our underlying assumptions is that a human-computer interface that embodies human discourse rules and conventions will be easier for people to learn and use than one that does not. We are encouraged in this assumption by the success of direct-manipulation graphical user interfaces, which we take to be due in large part to users' preexisting familiarity with the manipulation of real objects.

We are presently working toward our goal within the "software agent" paradigm, specifically, by adding a collaborative software agent to graphical user interfaces.

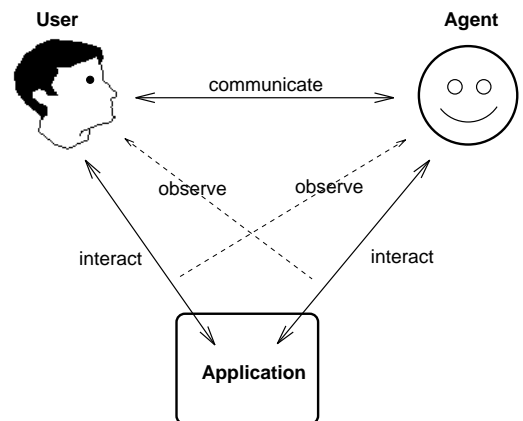


Figure 1: Collaborative interface agent paradigm.

Software agents are currently a new research area without precise definition. Roughly speaking, a software agent is an autonomous software process which interacts with humans as well as with elements of its software environment, such as the operating system, application programs, and other agents.

The crux of this paper is the automatically segmented history of an interaction between a user and a software agent shown in Figure 5 and its discussion in Section 7. The sections leading up to Section 7 cover the motivation, methodology, and theoretical background of the work. The sections following Section 7 describe the underlying representations and algorithms. All of the examples in the paper are from a system implemented in Common Lisp and running in real time (a few seconds per interaction).

## 2 COLLABORATIVE INTERFACE AGENT

Our version of the software agent paradigm, which we term a *collaborative interface agent* (see Figure 1), mimics the relationships that hold when two humans collaborate on a task involving a shared artifact, such as two mechanics working on a car engine together or two computer users working on a spreadsheet together. This implies, as shown in the figure, that the software agent is able to communicate with and observe the actions of the human user and must itself be able to interact with the

application program being used to perform the shared task. Furthermore, the agent interacts with shared application through the same graphical interface used by the human in a way that can be observed by the human user. This approach facilitates the reuse of existing applications and supports collaboration by making it easy for the user to know what the agent is doing.

Most current work on similar software agents is very application-specific. Our goal has been to develop a general-purpose collection of algorithms, data structures, and specifications (i.e., a toolkit) that application programmers can easily use to add a collaborative agent to any application program. The key components of this toolkit called *Collagen*<sup>1</sup> (for *Collaborative agent*) concern:

- grouping the steps of an interaction into segments based on their purposes,
- modelling how the purposes of segments relate to each other and to the overall goals of the collaboration, and
- communication between the user and the agent about mutual beliefs and the division of labor.

It is possible to achieve substantial application independence in these areas based on the results of more than two decades of research on collaborative discourse in which the computational structures we use have been validated across a range of human tasks. By adding application-specific knowledge and rules to these generic underlying discourse structures, we can implement collaborative agents for particular applications that help users by suggesting what to do next, summarizing the current state of the problem solving process, and performing delegated tasks.

### 3 METHODOLOGY

Our basic methodology has been to choose a specific application program to serve as a test for our approach. We wanted this application program to be more complex than the typical research toy, but less complex than a full commercial program. We also wanted the application interface to be a good example of the current state of the art, i.e., a pure direct-manipulation interface where all of the underlying state of the application is graphically visible and modifiable. The next section describes the air travel planning application we implemented to serve this purpose.

To our test application, we then added a collaborative interface agent as shown in Figure 1, being very careful to keep the application-specific parts of the agent separate from the generic parts. This paper is a first report on the implementation and behavior of this collaborative agent.

After some additional work described in the conclusion, we plan a pilot user study, in which the performance of users solving problems using the test application alone is compared with their performance with the collaborative

agent added. We plan to examine such factors as the ease with which users accomplish the given tasks, the speed of solution, and their overall satisfaction.

Finally, a key aspect of our approach is that we are *not* using natural language understanding technology. This is important for both practical and theoretical reasons. As a practical matter, natural language understanding even in this limited setting is a very difficult problem in its own right, which we would like to sidestep for the moment. From a theoretical point of view, we want to emphasize that discourse theory addresses the *content* of collaborative communication at a very fundamental level, regardless of what language is used.

As the internal semantic (i.e., content) representation of the communication between the user and agent, we use an artificial language developed by Sidner [17], and translate messages from this internal representation into English sentences for the user to read using simple string templates (see Section 8).

### 4 TEST APPLICATION

Figure 2 shows the interface to the air travel planning system we implemented using the Garnet [15] graphics package. (For the discussion in this section, please ignore the two overlapping windows in the upper-right and lower-left corners.) The test application provides a direct-manipulation interface to an airline schedule database and a simple constraint checker. By pressing buttons, moving sliders, and so on, the user can specify and modify the geographical, temporal, and other constraints on a planned trip. The user can also retrieve and display possible itineraries satisfying the given constraints.

A typical problem to be solved using this application is the following:

You are a Boston-based sales representative planning a trip to visit customers in Dallas, 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 attend your son's piano recital.

In order to get some initial intuitions about the problem-solving process in this domain, we asked seven visitors and staff members at our laboratory to solve this and similar problems using the test application and recorded their behavior via informal notes and the logging facilities we built into the system. A typical problem-solving session lasted about 15 minutes and entailed about 150 user actions (mouse clicks).

In a typical session, the user begins by clicking on the route map to specify the origin, order of layover cities,

<sup>1</sup>Collagen is a fibrous protein that occurs in vertebrates as the chief constituent of connective tissue.

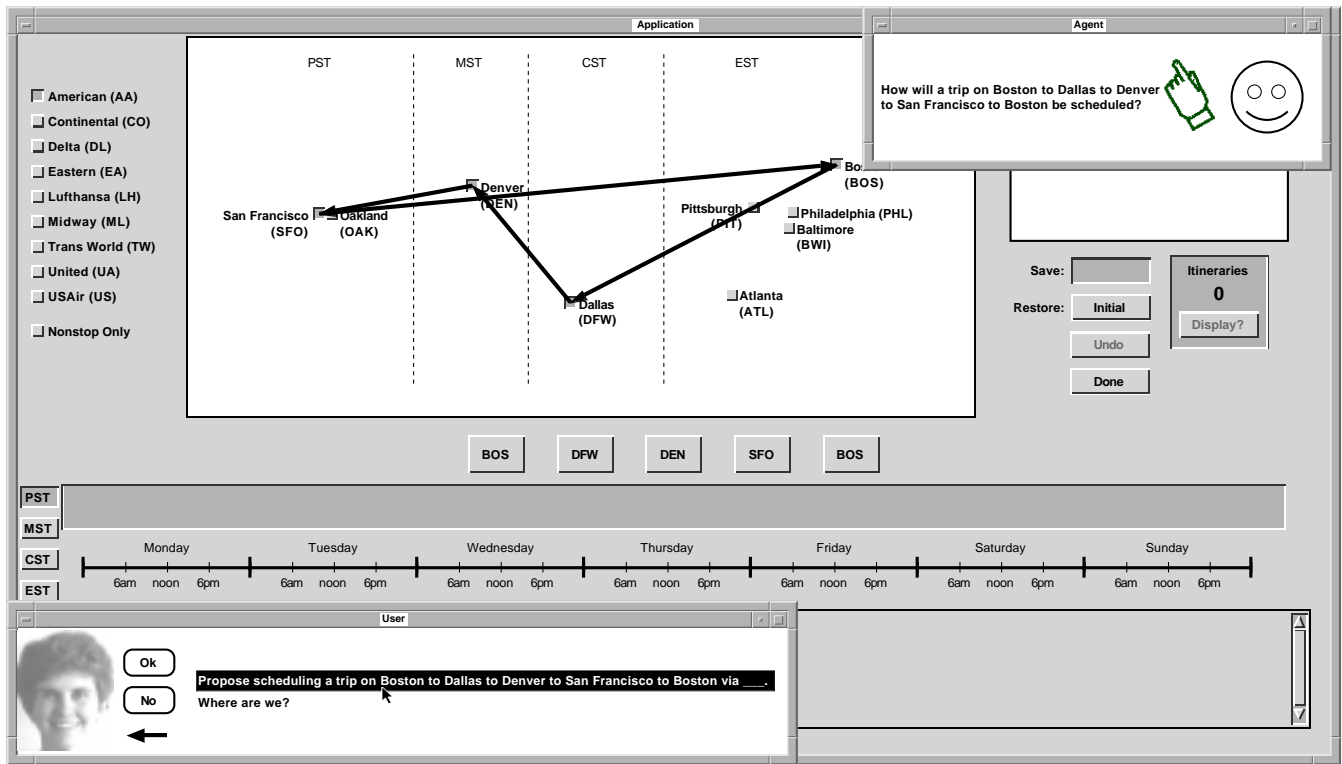


Figure 2: Test application with overlapping home windows.

and final destination for the trip. Next, users typically manipulate some of the small rectangles labelled with city names. These city “interval” bars (see Figure 6) can be inserted into the horizontal slider area below the map and moved and stretched to specify latest arrival and earliest departure times at each city. Users can also restrict the airlines used by setting the buttons to the left of the route map.

Whenever the user enters or changes a constraint, the number of possible itineraries is automatically recomputed from the data base of all flights and displayed in the box labelled Itineraries. By pressing the Display? button in that box, the user can view all the possible itineraries laid out along the same timeline as the interval bars in the large scrollable area at the bottom of the screen (see Figure 6).

In general, users find that displaying more than 5 to 10 itineraries is too much information. If there are more than this many, they typically add further constraints or look only at the first few itineraries displayed. The system does not allow display when there are zero (over-constrained) or more than 100 (under-constrained) itineraries.

The main difficulties users experienced using the test application (or at least the ones we paid most attention to) were various forms of getting stuck and getting lost. Users had trouble knowing what to try next when they had over- or under-constrained their trip. They also had trouble keeping track of which combinations of

routes and constraints they had already examined. Although the test application does provide an Undo button and a “snapshot” facility (via the Save button and Restore menu), these were not flexible enough. For example, the Undo button, as is typical in current interfaces, forces you to undo actions in reverse chronological order. Sometimes you want to undo an earlier action without undoing the intervening actions. Similarly, sometimes you want to restore only part of a previous state.

One of our conclusions from these sessions is that the users can productively be viewed as “designing” itineraries (sequence of flights). As is typical in design tasks, the strategies used included information seeking (e.g., to see what components—flights—are available with various properties), constraint satisfaction (e.g., arrival and departure time preferences), cost reduction (e.g., travel time), searching, backtracking, trade-offs, etc. All of these showed up in simple ways using the test application.

Another important property of these scenarios is that the problem statements are only partially formalizable within the system. (There may also be more than one right answer.) To take an example from the problem statement in this section, notice that the test application does not provide a representation to specify in advance (even if you could) just how much travel inconvenience you would put up with in order to accumulate more frequent-flyer miles on American. This kind of incompleteness is typical of design and many other tasks for which people use interactive systems.

## 5 WINDOW SHARING

The first step in adding a collaborative interface agent to the test application is to establish the basic communication, observation, and interaction channels required by the paradigm shown in Figure 1. This is achieved using a window-sharing layer implemented in the X Window System and described in detail elsewhere [16].

A key concept in this window-sharing layer is the *home window*. The user and software agent each have a small dedicated window that is used for communication between them. The home windows start out in the corner locations shown in Figure 2; the user may move them to different screen locations in the usual ways provided by the window system.

Each home window contains an identifying face and has an associated *cursor*. The user's cursor is his usual mouse pointer. The agent's cursor is the pointing hand icon shown in its home window. The agent uses this hand to point and click on the shared application window just like the user's mouse pointer. The agent's eyes blink periodically to indicate that its process is still running. The home windows also shrink and expand as they are used. For example, after the user has chosen from her communication menu in Figure 2, both home windows return to their default configurations shown below.

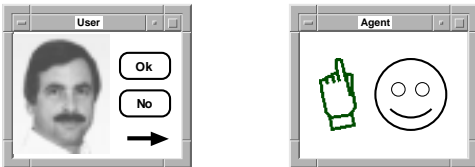


Figure 3: Default home window configurations.

To support asynchronous real-time interaction, the application and each home window is serviced by a separate process. Thus even when the agent has asked a question, the user is free to continue clicking on the application window instead of answering. Furthermore, using a distributed window system like X, the agent and user processes may run on a different machine than the application process. (Whether or not to run the agent process in the same address space as the application is an engineering tradeoff that depends on the application.)

Returning now to Figure 1, let us account for how each of the arrows is realized:

- Communication from the agent to the user is achieved by printing English text in the agent's home window, as illustrated in Figure 2.
- Communication from the user to the agent is achieved by the user selecting from a menu as illustrated in Figure 2. Internally, a message in the artificial discourse language is transmitted to the input buffer of the agent process.
- The user interacts with the application in the usual way, modifying the state of the application with her cursor and "querying" it with her eyes.

- The agent modifies the state of the application with its cursor (see discussion of UnGUI module in [16]) and queries it using the programming interface (API) provided by the application.
- The user observes the agent's actions by watching the agent's cursor.
- The agent observes the user's actions by virtue of a generic layer in the application that mirrors all logical actions into the input buffer of the agent process.

## 6 COLLABORATION AND DISCOURSE

*Collaboration* is a process in which two or more participants coordinate their actions toward achieving shared goals. Most collaborations between humans involve communication. *Discourse* is a technical term for an extended communication between two or more participants in a shared context, such as a collaboration.

### SharedPlans

Much is known about the structure of human collaboration and discourse. In this work, we make use specifically of two interrelated theories due to Grosz and Sidner [6, 7] and extensions by Grosz and Kraus [5] and Lochbaum [12, 13]. Their theory predicts that, for successful collaboration, the participants need to have mutual beliefs<sup>2</sup> about the goals and actions to be performed and the capabilities, intentions, and commitments of the participants. The formal representation of these aspects of the mental states of the collaborators is called a *SharedPlan*.

As an example of a SharedPlan in the air travel domain, consider the collaborative scheduling of a trip wherein participant A (e.g., the user) knows the constraints on travel and participant B (e.g., the software agent) has access to a database of all possible flights. To successfully complete the collaboration, A and B must mutually believe that they:

- have a common goal (to find an itinerary that satisfies the constraints);
- have agreed on a sequence of actions (a *recipe*) to accomplish the common goal (e.g., choose a route, specify some constraints on each leg, search for itineraries satisfying the constraints);
- are each capable of performing their assigned actions (e.g., A can specify constraints, B can search the database);
- intend to do their assigned actions; and
- are committed to the overall success of the collaboration (not just the successful completion of their own parts).

In Collagen, we have implemented data structures and algorithms, described in more detail below, for representing and manipulating goals, actions, recipes, and SharedPlans.

<sup>2</sup>A and B mutually believe p iff A believes p, B believes p, A believes that B believes p, B believes that A believes p, A believes that B believes that A believes p, and so on. This is a standard philosophical concept whose infinite formal definition is not a practical problem.



```

A: "Replace the pump and belt please."
  [
    B: "OK, I found a belt in the back."
      [
        B: "Is that where it should be?"
        A: "Yes."
      ]
    B: Removes belt.
    B: "It's done."
  ]
  [
    A: "Now remove the pump."
    ...
    A: "First you have to remove the flywheel."
    ...
    A: "Now take off the base plate."
    B: "Already did."
  ]

```

Figure 4: Segments in a human discourse.

Several important features of collaboration should be noted here.

First, participants do not usually begin a collaboration with all of the conditions above “in place.” They typically start with only partial knowledge of the shared environment and the other participants and use communication as well as individual information gathering to determine the appropriate recipe to use, who should do what, and so on.

Second, notice that SharedPlans are recursive. For example, the first step in the recipe mentioned above, choosing a route, is itself a goal upon which A and B might collaborate.

Finally, planning (coming to hold the beliefs and intentions required for the collaboration) and execution (acting upon the current intentions) are usually interleaved for each participant and among participants. Unfortunately, there is currently no generally accepted domain-independent theory of how people manage this interleaving. (The current best candidates for a generic theory are the so-called belief/desire/intention frameworks, such as [2].) Collagen therefore does not currently provide a generic framework for execution. Another way of saying this is that we only provide a generic framework for *recording* the order in which planning and execution occur, but *not* for *deciding* how to interleave them.

### Discourse Segments

The concept of discourse segments is at the very foundation of discourse theory. A tremendous amount of analysis of discourses from a range of human interactions has resulted in the view that discourse has a natural hierarchical structure. The elements of this hierarchy are called *segments*. The existence of segments can be seen in everything from pitch patterns (in spoken discourse) to the way that pronouns are interpreted. As we will see in Section 7 below, automatic segmentation has there-

fore been our first milestone in adding discourse facilities to human-computer interaction.

A simple example of segments in a human collaborative discourse is shown in Figure 4, which is adapted from [8]. In this discourse, participant A is instructing participant B how to repair an air compressor. The toplevel segment and three embedded segments are indicated by the brackets and indentation shown (further subsegments are elided). In Grosz and Sidner’s theory, the segment structure of a discourse is accounted for by assigning a *purpose* to each segment, such that each segment’s purpose contributes to successful collaboration on the parent segment’s purpose via the conditions of the SharedPlan described above.

For example, the purpose of the toplevel segment in Figure 4 is to replace the pump and belt, which is the common goal of the collaboration. The purpose of the first subsegment is to remove the belt, which is one of the steps in the recipe for replacing the belt. The purpose of the first subsubsegment is to identify a parameter of the removal action, i.e., the belt to be removed. The purpose of the second subsegment is to remove the pump, which is also one of the steps in the recipe for the toplevel purpose.

Notice in Figure 4 that our analysis of the discourse includes not only the participants’ utterances, but also their actions (e.g., B removes belt). This is particularly appropriate in a context, such as collaborative interface agents, where all actions on the shared artifact are mutually observable.

## 7 SEGMENTED INTERACTION HISTORY

Figure 5 shows the segmented history of an interaction between a user and our collaborative agent solving the problem posed at the start of Section 4. The contents of the figure were automatically generated by the agent as a printout of the data structures built during the interaction by the algorithms described in Section 9; only extra spacing and authors’ comments in italics have been added. The embedding of segments is indicated by indentation. The start of each segment is indicated by a line in brackets [...] containing a description of the discourse segment purpose.

Other interactive systems also maintain histories. However, most such histories are “flat” or, if they do have structure, it is a reflection of the nesting of dialog boxes, rather than, as is the case here, the user’s problem solving process.

Notice that each event in the history is either a communication, represented by an English gloss in quotes "...", or the description of an application-level manipulation (result of a mouse click). Both the user and agent perform both communication and manipulation acts and the initiative moves back and forth between the two.

The interaction begins with the user choosing one of a set of default high-level goals. After the agent accepts the user’s proposal, the user pauses with her cursor on

[Scheduling a trip on Boston to Dallas to Denver to San Francisco to Boston]

1 USER: "Propose scheduling a trip."  
2 AGENT: "Ok."  
*(user pauses at home)*

[User identifying route of scheduling a trip as Boston to Dallas to Denver to San Francisco to Boston]

3 AGENT: "What is route of scheduling a trip?" *(Figure 7 here)*  
4 USER: Add Boston to the route. *(implicit acceptance)*  
5 USER: Add Dallas to the route.  
6 USER: Add Denver to the route.  
7 USER: Add San Francisco to the route.  
8 USER: Add Boston to the route.

[Interruption where user adding American airline] *(incomplete recipe)*  
9 USER: Add American airline.  
*(user pauses at home)*

[User proposing scheduling a trip on Boston to Dallas to Denver to San Francisco to Boston via working backward]

10 AGENT: "How will a trip on Boston to Dallas to Denver to San Francisco to Boston be scheduled?" *(Figures 2, 8 here)*  
11 USER: "Propose scheduling a trip on Boston to Dallas to Denver to San Francisco to Boston via working backward."

[User working on San Francisco to Boston leg]

12 USER: "Propose working on San Francisco to Boston leg."  
13 USER: Add Boston final interval with arrival Sun 11:30p.  
14 USER: Change Boston final interval to arrival Fri noon.  
15 USER: Add San Francisco stopover with arrival Wed 1:55a departure Wed 9:55a.  
16 USER: Change San Francisco stopover to arrival Thu 6:00p departure Fri 1:57a.  
*(user pauses at home)*

[User working on Denver to San Francisco leg]

17 AGENT: "Propose working on Denver to San Francisco leg."  
... *(elide working on rest of legs)*  
*(user stuck - pauses at home with zero itinerary count)*

[Agent adding United airline]

36 AGENT: "Propose I add United airline."  
37 USER: "Ok."  
38 AGENT: Add United airline.

[User displaying and selecting itineraries]

39 USER: Display 4 itineraries.  
40 USER: Select itinerary Wed AA/1099 Wed AA/313 Thu UA/343 Fri AA/928.  
41 USER: Done. *(only user can know when done)*

Figure 5: Automatically segmented interaction history with authors' comments added in italics.

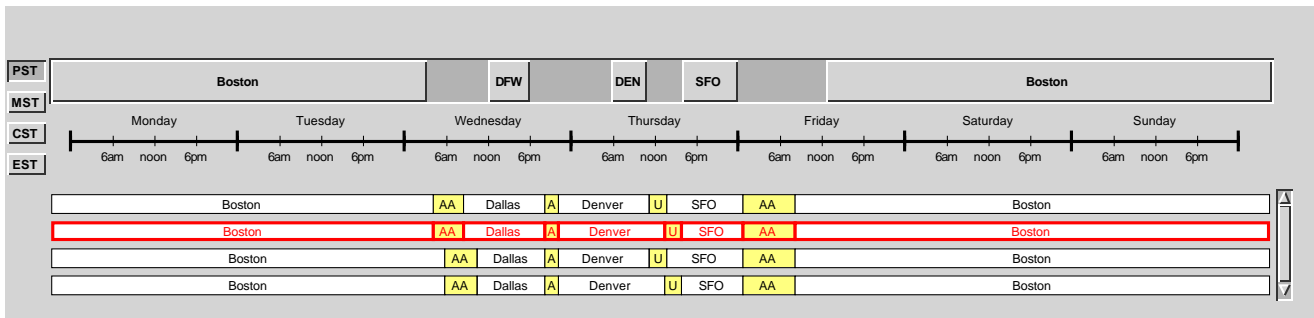


Figure 6: Interval bars and displayed itineraries at end of session above.

her home page. In a face-to-face conversation, there are many cues, such as pauses, eye contact, voice pitch, and body position, that help signal when a speaker has ended her “turn.” In this system, we have introduced a—we hope intuitive—artificial gesture for this purpose, namely, leaving the cursor motionless in the home window for more than a second.

In response to the user’s pause, the agent takes the initiative (event 3) and proposes a subgoal which contributes to the current goal by identifying an unspecified parameter, the route. The user implicitly accepts the agent’s proposal by starting to act on this subgoal (event 4).

The segments of the interaction history are computed incrementally. So, for example, if instead of adding Boston to the route following event 3, the user had selected “Where are we?” from her communication menu, the following would be printed in the agent’s home window:

```
[Scheduling a trip]
1 USER: "Propose scheduling a trip."
2 AGENT: "Ok."
   [User identifying route of scheduling a trip]
3 AGENT: "What is route of scheduling a trip?"
```

Figure 7: Where are we?

Event 9 illustrates where the agent’s current recipe knowledge is incomplete. The user’s manipulation is analyzed as an interruption, i.e., a segment that does not contribute to its parent, because none of the agent’s recipes for scheduling a trip have a step for specifying airlines. Clearly, these recipes need to be improved. However, we take the view in general that the agent’s knowledge will never be complete and it therefore must deal gracefully with unexpected events.

The agent currently has two recipes, gleaned from our observations of real users, for scheduling a trip on a given route: working forward and working backward. The working-forward recipe works on the legs of the trip in order starting with the first leg; the working-backward recipe starts with the last leg.

Figure 2 shows the moment in the interaction after the agent’s question in event 10 when the user is about to answer by choosing what will become event 11 from the communication menu in her home window. Unlike conventional pop-up menus, the user at this point could have just ignored the agent’s question and continued clicking on the application window instead.

Notice that the only other entry in her menu at this point is the generic “Where are we?”. The main reason we believe that menus can be workable here is because the generation algorithm described in Section 9 only produces the relatively small number of communication actions that are expected in the current discourse context.

Notice also that the user’s menu choice in Figure 2 has

a blank at the end where the name of the recipe is supposed to go. Once a communication choice is made, the user is presented with another menu, such as the one below, to choose from the allowable values for any unspecified elements.

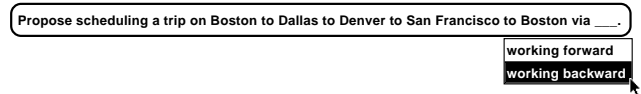


Figure 8: Choosing a recipe.

Given that the user chose the working-backward recipe, the interaction continues with the first step of the recipe, namely working on the San Francisco to Boston leg. Working on a leg means using the cursor to slide and stretch the city interval slider bars (see Figure 6).

Events 17 and 36 are two examples of the agent’s response to a situation in which it appears the user does not know what to do next. The agent’s response in event 17 results from an application-independent strategy, namely, to propose the next step in the current recipe.

The agent’s response in event 36 is application-specific. As mentioned in Section 4, one of the main ways that users get stuck is by over-constraining their itinerary causing the itinerary count to become zero. The agent’s response here is to suggest reducing constraints by adding United to the set of allowable airlines. The agent did not choose this particular constraint change at random—it used its ability to access the test application’s API to find a constraint change that would in fact increase the number of itineraries. This is a good example of where the capabilities of one collaborator are usefully different from those of another.

The interaction ends with the user displaying the four possible itineraries in the scrolling window (see Figure 6) and selecting one. Notice that the last event is technically a manipulation rather than a communication, because there is a special “Done” button on the application interface.

This interaction history is an important first milestone in our overall research program. The underlying data structures (described in Section 9 below) will support many additional collaborative capabilities. As just one example, we are working on a better “undo” facility that works at the level of segments.

## 8 ARTIFICIAL DISCOURSE LANGUAGE

The most fundamental concept in Collagen’s internal representation is an *act*. An act is formed by applying a constructor function (act type) to the required *parameters*. The first two parameters of every act type are the time the act is performed and which participant (or set of participants) performs the act. Many act types, such as scheduling a trip, are abstract; they are essentially goals which may be achieved in several alternative ways (i.e., via different recipes).

A *basic* act is an act that is directly executable and not further decomposable. For example, the direct manipulations on the test application are basic acts.

Sidner’s artificial discourse language [17] defines a collection of constructors for basic communication acts, such as proposing, retracting, accepting, and rejecting proposals. This section provides a rough sketch of the language and our use of it. As a syntactic convention below, application-independent parts of the language are written in capitals, terms specific to the test application are in lower case, and variables are in *italics*.

Our current implementation includes only two basic communication act types: PFA (propose for accept) and AP (accept proposal). An area of future work is to implement Sidner’s other act types, such as retracting, rejecting, and counter-proposing, which are required to support collaborative negotiation.

```
PFA(t, participant1, belief, participant2)
```

The semantics of PFA are roughly: at time *t*, *participant*<sub>1</sub> believes *belief*, communicates his belief to *participant*<sub>2</sub>, and intends for *participant*<sub>2</sub> to believe it also. If *participant*<sub>2</sub> responds with an AP act, e.g., “Ok”, then *belief* is mutually believed.

Sidner’s language at this level is very general—the proposed *belief* may be anything. For communicating about collaborative activities, we introduce two application-independent operators for forming beliefs about acts: SHOULD(*act*) and RECIPE(*act*, *recipe*).

Below are examples of how this language represents some of the communication acts in Figure 5. In each example, we show the internal representation of the act followed by the English gloss that was automatically produced. Glosses are produced by a straightforward recursive substitution process using a set of string templates associated with each act type.

```
PFA(36,agent,SHOULD(add-airline(t,agent,ua)),user))
36 AGENT: "Propose I add United airline."
```

Notice below that a present participle template is used when the participant performing an act is unspecified.

```
PFA(1,user,SHOULD(schedule(t,who,route)),agent)
1 USER: "Propose a trip be scheduled."
```

Questions arise out of the embedding of PFA acts as shown below (*route* is a constructor for route expressions).

```
PFA(11,agent,
  SHOULD(PFA(t1,user,
    RECIPE(schedule(t2,who,
      route(bos,dfw,den,sfo,bos)),
      recipe), agent)),
  user)
```

```
11 AGENT: "How will a trip on Boston to Dallas to
  Denver to San Francisco to Boston be scheduled?"
```

```
PFA(12,user,RECIPE(schedule(t,who,route(...)),
  working-backward),agent)
```

```
12 USER: "Propose scheduling a trip on ...
  via working backward."
```

There is presently no way for users to construct an arbitrary communication; they can only choose from a menu of communication acts generated by the algorithm described in the next section. We are interested in seeing how far we can push this approach. However, should we decide to do so, it would be straightforward to provide an interface, such as a syntax-directed editor, for constructing arbitrary messages in the artificial language.

## 9 DISCOURSE PROCESSING IN COLLAGEN

This section provides an overview of the generic discourse processing algorithms and data structures that underlie the collaborative agent. Since the goal of this work is to use well-established human discourse algorithms, readers are referred to the referenced literature for more details.

Discourse processing in Grosz and Sidner’s framework [6, 7] has three interacting components:

- The linguistic structure of a discourse includes the grouping of utterances into segments, as well as the use of anaphora, tense, and cue words. We use the artificial language above to represent this kind of linguistic information.
- The attentional state of a discourse captures the shifting focus of attention of the participants. We represent attentional state as a *focus stack* of discourse segments.
- The intentional structure of a discourse corresponds to the current partial status of the participants’ Shared-Plans. We represent this information as a *recipe tree*, which is a reimplementaion of Lochbaum’s rgraph structure [12]. Each node of a recipe tree is a common goal; the children of the node are acts that contribute to it.

As can be seen in Figure 9, the agent currently uses only one focus stack and recipe tree, which represents mutually believed information. When we start to work on negotiation, we will need to distinguish between mutual beliefs and the individual beliefs of the agent and user.

Our key discourse processing algorithm, *discourse interpretation*, is a reimplementaion of Lochbaum’s rgraph augmentation algorithm [12]. The *discourse generation* algorithm is essentially the inverse of the interpretation algorithm.

### Discourse Interpretation

The input to discourse interpretation is a (basic) manipulation or communication act that has been observed or received by the agent, i.e., queued in its input buffer. The main job of the algorithm is to see if the current act can be viewed as contributing to the current discourse purpose, i.e., the purpose of the top segment on the focus stack. This breaks down into five main cases.

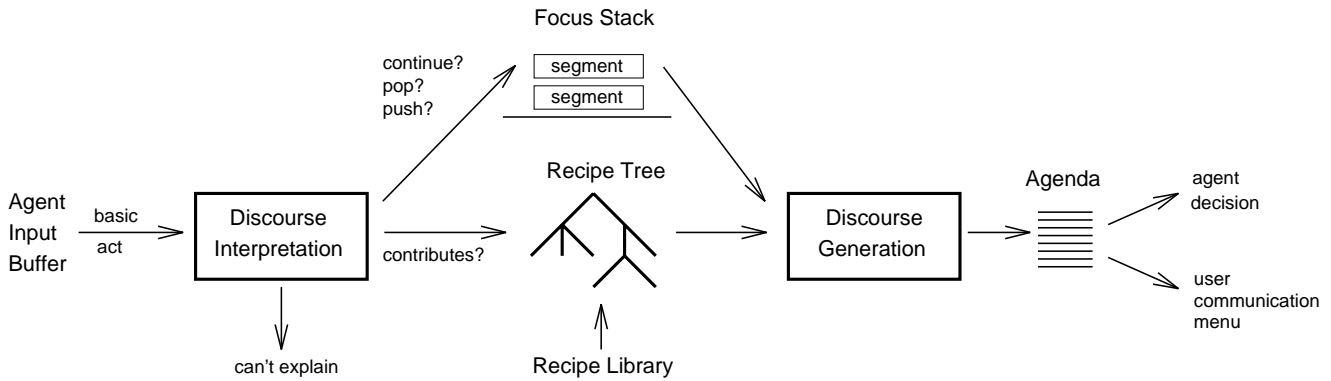


Figure 9: Discourse processing algorithms and data structures in Collagen.

The current act either:

- directly achieves the current purpose,
- is one of the steps in a recipe for the current purpose (this may involve retrieval from the recipe library),
- identifies the recipe to be used to achieve the current purpose,
- identifies who should perform the current purpose or a step in the current recipe, or
- identifies an unspecified parameter of the current purpose or a step in the current recipe.

If one of these cases obtains, the current act is added to the recipe tree and the current segment. Furthermore, if this act completes the achievement of the current discourse segment purpose, the focus stack is popped. The last three cases above are instances of a larger class of explanations that Lochbaum [13] calls “knowledge preconditions.”

If the current act is a communication of the form

PFA(..., ..., SHOULD(...), ...)

it is interpreted as initiating a new segment which is pushed onto the focus stack regardless of whether the proposed act contributes to the current purpose. This is part of the modelling of interruptions, i.e., subsegments whose purpose does not contribute to the purpose of the parent segment.

Finally, acts that cannot be understood in any of the ways described above are turned over to a rudimentary exception-handling mechanism, which at the moment uses some heuristics to try to distinguish between interruptions and abandonments of the current purpose. Further research is needed in this area.

It is tempting to think of discourse interpretation as plan recognition, which is known to be exponential in the worst case [10]. However, this misses a key property of normal human discourse, namely that speakers work very hard (even to the extent of providing redundant information [20]) to make sure that their conversational partners understand their intentions without a large cognitive search. Notice that the only search involved in the algorithm above is through the steps of

the current recipe or all known recipes for the current discourse purpose (and this is *not* done recursively).

### Discourse Generation

The discourse generation algorithm looks at the current focus stack and recipe tree and produces a prioritized *agenda* of (possibly partially specified) acts which would contribute to the current discourse segment purpose. The communication acts in this agenda which could be performed by the user are used to construct user communication menus like the one shown in Figure 2. The agent also uses this agenda as input to its decision about what to do next. For example, it may propose that the user perform a high priority act on the agenda.

Priorities are assigned to acts in the agenda based on some simple application-independent heuristics. For example, the highest priority is assigned to unexecuted steps of a recipe, all of whose parameters have been specified and all of whose predecessors in the recipe have been executed.

## 10 RELATED WORK

This work lies at the intersection of many threads of related research in user interface, linguistics, and artificial intelligence. It is unique, however, in its combination of goals and techniques.

Our concept of an interface agent is closest to the work of Maes [14], although she uses the term “collaborative” to refer to the sharing of information between multiple software agents. Cohen [4] has also developed interface agents without collaborative discourse modelling. Terveen [19] has explored providing intelligent assistance through collaborative graphical manipulation without explicit invoking the agent paradigm.

Cohen [3] and Jacob [9], among others, have explored discourse-related extensions to direct manipulation that incorporate anaphora and make previous context directly available. However, most work (e.g., [11, 21]) on applying human discourse principles to human-computer interaction have assumed that natural language understanding will be applied to the user’s utterances.

The two systems we know of that are overall closest in spirit to our own are Stein et al.'s MERIT [18] and Ahn et al.'s DenK [1]. MERIT uses a different discourse theory and compiles it into a finite-state machine representation, which is less flexible and extensible. DenK has the goal of providing a discourse-based agent, but has not yet modelled collaboration.

## 11 CONCLUSION

We view the current state of this research as providing a new conceptual platform upon which we and others can now build higher. Our short-term implementation goals include:

- history-based transformations, such as undoing, returning to, or replaying segments,
- adding communication acts to support negotiation,
- and additional application-specific recipes.

With these and a few other improvements in place, we plan to begin initial user testing.

In the longer term, some of the directions we hope to see explored include:

- agents that operate remotely in space and time (e.g., on the Internet), which will entail relaxing the requirement for complete mutual observability and increasing the level of discussion between the agent and user about future actions, and
- generalizing Collagen to support multiple human and software agents (the underlying window sharing layer and the SharedPlan formalism have already been generalized in this way).
- communicating with the software agent in spoken language, taking advantage of the discourse generation algorithm to reduce the search space for speech recognition,

Most current user interface design uses the familiar tools of menus, buttons, dialogue boxes, and so on, to implement the following “collaborative” principle: tell the user something simple and constrain her options to respond. As we have illustrated in this paper, true collaboration is a much richer process. The discourse capabilities of our software agent provide this kind of richness by structuring the interaction and its history and letting users make their purposes more explicit. We hope this work will inspire new efforts in user interface design, with or without software agents.

## REFERENCES

1. R. Ahn et al. The DenK-architecture: A fundamental approach to user-interfaces. *Artificial Intelligence Review*, 8:431–445, 1994-5.
2. M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, November 1988.
3. P. Cohen. The role of natural language in a multimodal interface. In *Proc. ACM Symposium on User Interface Software and Technology*, pages 143–149, Monterey, CA, November 1992.
4. P. Cohen et al. An open agent architecture. In O. Etzioni, editor, *Software Agents, Papers from the 1994 Spring Symposium, SS-94-03*, pages 1–8. AAAI Press, Menlo Park, CA, March 1994.
5. B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 1996. In publication.
6. B. J. Grosz and C. L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.
7. 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.
8. B. J. Grosz [Deutsch]. The structure of task-oriented dialogs. In *IEEE Symp. on Speech Recognition: Contributed Papers*, pages 250–253, Pittsburgh, PA, 1974.
9. R. J. K. Jacob. Natural dialogue in modes other than natural language. In R.-J. Beun, M. Baker, and M. Reiner, editors, *Dialogue and Instruction*, pages 289–301. Springer-Verlag, Berlin, 1995.
10. H. Kautz. A circumscriptive theory of plan recognition. In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions and Communication*, chapter 6, pages 105–133. MIT Press, Cambridge, MA, 1990.
11. L. Lambert and S. Carberry. A tripartite plan-based model of dialogue. In *Proc. 29th Annual Meeting of the ACL*, Cambridge, MA, 1991.
12. K. E. Lochbaum. Using collaborative plans to model the intentional structure of discourse. Technical Report TR-25-94, Harvard Univ., Ctr. for Res. in Computing Tech., 1994. PhD thesis.
13. K. E. Lochbaum. The use of knowledge preconditions in language processing. In *Proc. 14th Int. Joint Conf. Artificial Intelligence*, pages 1260–1266, Montreal, Canada, August 1995.
14. P. Maes. Agents that reduce work and information overload. *Comm. ACM*, 37(17):30–40, July 1994. Special Issue on Intelligent Agents.
15. B. Meyers et al. Garnet: Comprehensive support for graphical, highly-interactive user interfaces. *IEEE Computer*, 23(11):71–85, November 1990.
16. C. Rich. Window sharing with collaborative interface agents. *ACM SIGCHI Bulletin*, 28(1):70–78, January 1996. Also published as MERL Technical Report 95-12.
17. C. L. Sidner. An artificial discourse language for collaborative negotiation. In *Proc. 12th National Conf. on Artificial Intelligence*, Seattle, WA, August 1994.
18. A. Stein and E. Maier. Structuring collaborative information-seeking dialogues. *Knowledge-Based Systems*, 8(2-3):82–93, April 1995.
19. G. Terveen, D. Wroblewski, and S. Tighe. Intelligent assistance through collaborative manipulation. In *Proc. 12th Int. Joint Conf. Artificial Intelligence*, pages 9–14, Sydney, Australia, August 1991.
20. M. A. Walker. Redundancy in collaborative dialogue. In *14th Int. Conf. on Computational Linguistics*, 1992.
21. N. Yankovich. Talking vs. taking: Speech access to remote computers. In *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 275–276, Boston, MA, April 1994.