COLLAGEN: When Agents Collaborate with People

Charles Rich, Candace L. Sidner

TR96-16   July 1996

Abstract

We take the position that autonomous agents, when they interact with people, should be governed by the same principles that underlie human collaboration. These principles come from research in computational linguistics, specifically collaborative discourse theory, which describes how people communicate and coordinate their activities in the context of shared tasks. We have implemented a prototype toolkit, called Collagen, which embodies collaborative discourse principles, and used it to build a collaborative interface agent for a simple air travel application. The potential benefits of this approach include application-independence, naturalness of use, and ease of learning, without requiring natural language understanding by the agent. Superseded by TR97-21.

First International Conference on Autonomous Agents, Marina del Rey, CA, February, 1997, pp. 284-291 and reprinted in M. Huhns and M. Singh, editors, Readings in Agents, Morgan Kaufmann, San Francisco, CA, 1997, pp. 117–124

COLLAGEN: When Agents Collaborate with People

Charles Rich   Candace L. Sidner*

TR-96-16   July 1996

Abstract

We take the position that autonomous agents, when they interact with people, should be governed by the same principles that underlie human collaboration. These principles come from research in computational linguistics, specifically collaborative discourse theory, which describes how people communicate and coordinate their activities in the context of shared tasks. We have implemented a prototype toolkit, called Collagen, which embodies collaborative discourse principles, and used it to build a collaborative interface agent for a simple air travel application. The potential benefits of this approach include application-independence, naturalness of use, and ease of learning, without requiring natural language understanding by the agent.

To appear in First Int. Conf. on Autonomous Agents, Marina del Rey, CA, February 1997.

*Lotus Development Corporation

Publication History:–

1. First printing, TR-96-16, July 1996

COLLAGEN: When Agents Collaborate with People

Charles Rich
MERL–A Mitsubishi Electric
Research Laboratory
201 Broadway
Cambridge, MA 02139 USA
rich@merl.com

Candace L. Sidner
Lotus Development Corporation
55 Cambridge Parkway
Cambridge, MA 02142 USA
csidner@lotus.com

Abstract

We take the position that autonomous agents, when they interact with people, should be governed by the same principles that underlie human collaboration. These principles come from research in computational linguistics, specifically collaborative discourse theory, which describes how people communicate and coordinate their activities in the context of shared tasks. We have implemented a prototype toolkit, called Collagen, which embodies collaborative discourse principles, and used it to build a collaborative interface agent for a simple air travel application. The potential benefits of this approach include application-independence, naturalness of use, and ease of learning, without requiring natural language understanding by the agent.

Introduction

The current explosion of work on autonomous agents is primarily driven, as it should be, by the excitement of finding useful results in specific applications. In parallel, however, we also need to be developing a foundation of application-independent principles and techniques. This paper reports on a principled approach to the part of an autonomous agent which interacts with—and collaborates with—people.

We take the position that agents, when they interact with people, should be governed by the same principles that underlie human collaboration. Our motivation for this position is the assumption that a style of interaction which embodies familiar rules and conventions will be easier for people to learn and use than one that does not. This is similar to the arguments that motivated the development of direct-manipulation graphical user interfaces to take advantage of users' preexisting familiarity with the manipulation of real objects.

To find the principles underlying human collaboration, we appeal to research on collaborative discourse, specifically the SharedPlan work of Grosz & Sidner (1986, 1990), Grosz & Kraus (1996), and Lochbaum (1994, 1995). This work provides us with a well-specified computational theory that has been empirically validated across a range of human tasks. Applying this theory to autonomous agents poses a number of challenges, including:

- applying the theory without requiring natural language understanding by the agent,
- embodying the application-independent algorithms and data structures in a toolkit,
- and providing a modular description for application-specific information.

Our solutions to these problems are the focus of this paper. To develop these solutions, we have implemented both a prototype toolkit called *Collagen*[1] (for *Coll*aborative *agen*t) and an air travel application using it. Because this paper concentrates on Collagen, it will by necessity be sketchier on the prototype application. Whenever possible, general issues will be illustrated in the air travel domain; a short sample session is also presented at the end of this paper. Readers are referred to (Rich & Sidner 1996a, 1996b) for more details on the air travel application system.

Collagen and the air travel application have both been implemented in Common Lisp. All of the examples in this paper are from our demonstration system, which runs in real time (maximum of a few seconds per interaction).

Collaboration and Discourse

This section provides a brief overview of the theory on which Collagen is based. Readers are referred to the referenced literature for more detail.

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

Figure 1 shows the structure of a generic collaboration between a software agent and a human user. Theories of collaboration and discourse tell us about the

[1]Collagen is a fibrous protein that occurs in vertebrates as the chief constituent of connective tissue.

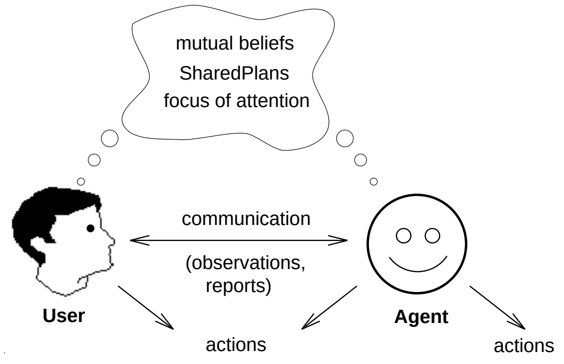


Figure 1: Collaborating with an agent.

structure and function of the "stuff" inside the clouds, which constitutes part of the human user's mental state and is computationally represented inside the agent. At this level of abstraction, this model applies to autonomous agents of all kinds.

Notice that communication between the user and agent includes observations and/or reports of their actions. It is clear that reports of actions, e.g., "I have done X," are a kind a communication. We also include direct observations of actions as a kind of communication when, as is often the case in close collaboration, both participants know and intend that their actions are observed. For example, in our example application, all agent and user actions are mutually observable through a direct-manipulation graphical interface (see Figure 3). Collagen also supports collaborations in which actions are only reported, e.g., where an agent is performing actions at a remote network site.

SharedPlans

Grosz & Sidner's (1990) theory predicts that, for successful collaboration, the participants need to have mutual beliefs[2] 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 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).

[2]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 philosphical concept whose infinite formal definition is not a practical problem.

Collagen provides data structures and algorithms for representing and manipulating goals, actions, recipes, and SharedPlans.

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 a collaboration) and execution (acting upon the current intentions) are usually interleaved for each participant and among participants.

Focus of Attention

In Grosz & Sidner's (1986) theory, the shifting focus of attention in a discourse is represented by a *focus stack* of discourse segments. A *segment* is a contiguous sequence of communication acts that serve some purpose. For example, a question and answer sequence constitutes a discourse segment whose purpose is (usually) to achieved shared knowledge of some fact. Segments are often hierarchically embedded. For example, a question/answer segment may include a question clarification subsegment.

The SharedPlan and focus stack representations are connected through discourse segment purposes: each discourse segment is associated with a SharedPlan for its purpose.

In the natural flow of a collaboration, new segments and subsegments are created, pushed onto the focus stack, completed, and then popped off the stack. Sometimes, participants also interrupt each other, abandon the current segment (purpose) even though it is not complete, or return to earlier segments.

Discourse Algorithms

The two key algorithms in Collagen are discourse interpretation, which is a reimplementation of Lochbaum's (1994) rgraph augmentation algorithm, and discourse generation, which is essentially the inverse of interpretation.

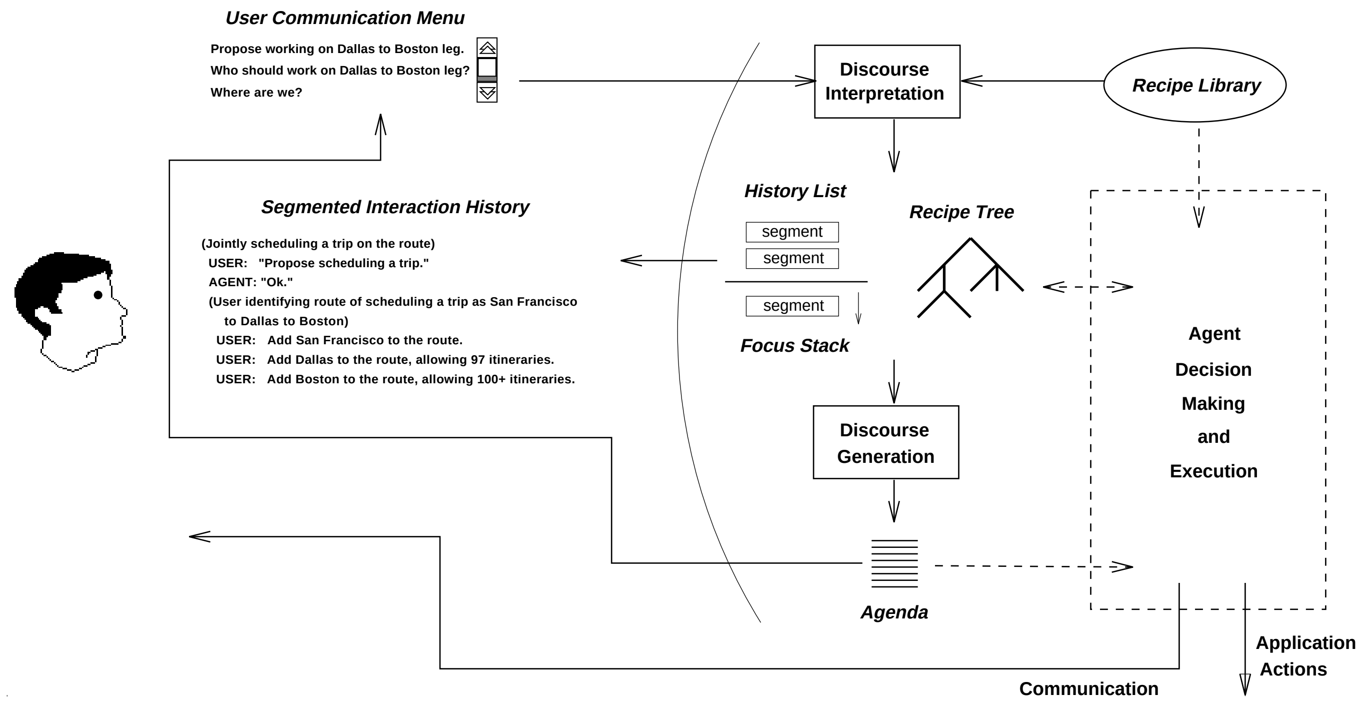

User Communication Menu
Propose working on Dallas to Boston leg.
Who should work on Dallas to Boston leg?
Where are we?
Discourse Interpretation
Recipe Library
History List
segment
segment
segment
Focus Stack
Recipe Tree
Segmented Interaction History
(Jointly scheduling a trip on the route)
USER: "Propose scheduling a trip."
AGENT: "Ok."
(User identifying route of scheduling a trip as San Francisco to Dallas to Boston)
USER: Add San Francisco to the route.
USER: Add Dallas to the route, allowing 97 itineraries.
USER: Add Boston to the route, allowing 100+ itineraries.
Agent Decision Making and Execution
Discourse Generation
Agenda
Application Actions
Communication

Figure 2: Collagen architecture.

The main job of *discourse interpretation* is to see how the current communication or observed action can be viewed as contributing to the current discourse purpose, i.e., the purpose of the top segment on the focus stack. For example, if the current purpose is to jointly schedule a trip, and the user proposes a route, the agent interprets this as the first step in the scheduling recipe. The intepretation algorithm also takes care of deciding when to push a new segment on the stack or pop the current segment.

It is tempting to think of discourse intepretation as plan recognition, which is known to be exponential in the worst case (Kautz 1990). However, this misses a key property of normal human discourse, namely that the participants work hard to make sure that their partners understand their intentions without a large cognitive search. In order to explain an observed or reported act, Collagen only searches through the steps of the current recipe or all known recipes for the current goal (and this is *not* done recursively). As we will see below, the agent relies on the user to communicate enough so that it can follow what is going on without having to do arbitrary plan recognition.

The *discourse generation* algorithm looks at the current focus stack and associated SharedPlan and produces a prioritized *agenda* of (possibly partially specified) actions which would contribute to the current discourse segment purpose. For example, if the current purpose is to jointly schedule a trip, the agenda includes an action in which the agent asks the user to propose a route. In general, the agenda contains communication and other actions by either the user or agent, which would advance the current problem-solving process.

Collagen

Collagen is a toolkit that embodies a set of conventions for collaborative discourse in the same sense that, for example, the Motif toolkit embodies a set of conventions for the graphical design of user interfaces. Generally speaking, what Collagen provides is a standard mechanism for maintaining the flow and coherence of agent-user interaction. In addition to saving implementation effort, using such a toolkit provides consistency across applications, and to the extent that the conventions are based on good principles, leads to applications that are easier to learn and use.

Even when using a toolkit, you still must provide a lot of application-specific information. After discussing the generic architecture of Collagen below, we will focus on how this application-specific information is provided and used.

Architecture

Figure 2 is a blow-up of Figure 1 to show the internal architecture of the part of the agent that interacts with the user. As commented earlier, the architecture at this level is applicable to all kinds of agents. In addition to the components shown in this figure, Collagen also includes a layer of special windowing facilities (Rich 1996) to support the interface agent paradigm (see later section).

The first thing to notice about the architecture in Figure 2 is that the agent's decision making and execution is a "black box." We are not trying to provide a toolkit for building a complete agent. At the heart of the agent there may be a rule-based expert system, a neural net, or a completely ad hoc collection of code. What we are providing are mechanisms for this black box to use when communicating with the user. Said another way, Collagen provides a generic framework for recording and communicating the decisions made by the agent (and the user), but not for *making* them.

User Presentations

From the user's point of view, a collaborative agent built using Collagen presents itself in three ways. All of these presentations are in natural language, which is generated from an internal artificial language (described below) by simple string template substitutions.

First (see bottommost arrow in the Figure 2), the user sees direct communications from the agent. The content of such a communication could be a question, an answer, a proposal, a report, etc. The exact method by which such communications are presented to the user depends on the application. For interface agents, Collagen provides a particular style of overlapping windows; other applications could simply print into a predefined message area.

Second (see topmost arrow in Figure 2), the user communicates to the agent by selecting from the dynamically-changing *user communication menu*, which is computed from the current discourse agenda. What we are doing here is using expectations generated by discourse context to replace natural language understanding. The user is not allowed to make arbitrary communications, but only to select from communications expected by the discourse interpretation algorithm. Thus, unlike usual ad hoc menu-driven interaction, the user menu in Collagen is systematically generated from an underlying model of orderly discourse. The choices in the user communication menu are simply the subset of the discourse agenda which are communication acts that may be performed by the user (plus a few special entries, such as "Where are we?" to request printout of the segmented history described below). An example of a user communication menu in our example application can be see in the lower left corner of Figure 4.

Finally (see middle arrow Figure 2), the user can request a printout of parts of the agent's internal discourse state in the form of a *segmented interaction history* (see example Figure 5). Segmented interaction histories are one of Collagen's most useful features. They are like log files, except that they are hierarachically structured and include not only primitive actions, but also the user's and agent's higher-level goals and intentions. The most basic function of the segmented interaction history is to orient the user. For example, if the user needed to leave her computer in the middle of working on a problem and returned after a few hours (or days), the history would help her reestablish where in the problem solving process she was.

The segmented interaction history also serves as a menu for history-based transformations, such as returning to earlier points in the collaboration or replaying earlier segments in a new context. See (Rich & Sidner 1996b) for more about history-based transformations.

Inside the Agent

The left side of Figure 2 shows the components of Collagen inside the agent. At the top of the figure, the *discourse interpretation* module receives selections from the user communication menu. Even though the user menu is presented in natural language, there is no natural language understanding required here, since each entry in the menu is associated with the artificial language expression from which it was generated.

The interpretation module updates the agent's internal discourse state representation according to the rules of discourse theory and the specific content of the user's communication. The agent's internal discourse state consists of the focus stack described earlier (shown growing downward in the figure), the *history list*, which records toplevel segments that have been popped off the stack, and the *recipe tree*, which is a concrete representation of some of the mutual beliefs in SharedPlans. The recipe library, which is also an input to discourse interpretation, will be discussed in its own section below.

The *discourse generation* module constantly updates the agenda as the discourse state changes.

Notice that the agent communication arrow at the bottom of Figure 2 originates in the agent decision making box. Only application-specific code can decide what the agent should actually say (or do) at any particular time. However, the Collagen architecture does provide resources which this application-specific agent code can use to support intelligent assistance. Collagen provides software interfaces (API's) for the recipe library, the discourse state representation, and the agenda.

For example, the agent in our air travel application consults the recipe tree as part of determining the best suggestion to make when the user has added too many constraints to her trip. Also, our example agent often chooses one of the entries on the current discourse agenda as its next action.

Task Modelling

In order to use Collagen, an agent developer must provide a formal model of the collaborative task(s) being performed by the agent and user. Defining this model is very similar to what is called "data modelling" in database or "domain modelling" in artificial intelligence (Brodie, Mylopoulos, & Schmidt 1982). It also overlaps with modern specification practices in software engineering, although the goals and recipes in a task model for collaborative discourse include more abstract concepts than are usually formalized in current software practice, except for in expert or knowledge-based systems.

On the one hand, task modelling 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 artificial intelligence 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 capturing more of the programmer's design rationale in the software itself. Also, since the agent is not required to use the task model alone for its decision making and execution, the model only needs to be complete enough to support communication and collaboration with the user.

In the remainder of this section, we discuss and illustrate some of the issues in building task models, starting with the artificial discourse language and then moving on the recipe library.

Artificial Discourse Language

As the internal representation for user and agent communication acts, we use Sidner's (1994) artificial discourse language. Sidner defines a collection of constructors for basic act types, such as proposing, retracting, accepting, and rejecting proposals. Our current implementation includes only two of these act types: `PFA` (propose for accept) and `AP` (accept proposal).

`PFA`(t, *participant*$_1$, *belief*, *participant*$_2$)

The semantics of `PFA` are roughly: at time t, *participant*$_1$ believes *belief*, communicates his belief to *participant*$_2$, and intends for *participant*$_2$ to believe it also. If *participant*$_2$ 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 actions: `SHOULD(`*act*`)` and `RECIPE(`*act*,*recipe*`)`.

The rest of the belief sublanguage is application-specific. For example, to model our air travel application, we defined appropriate object types (e.g., cities, flights, and airlines), relations (e.g., the origin and destination of a flight), and goal/action constructors (e.g., scheduling a trip, adding an airline specification). We can imagine automatically extracting some of these definitions from declarations in the implementation code for the agent.

Below are examples of how some of the communications in our example application are represented in the artificial discourse language. In each example, we show the internal representation of the communication followed by the English gloss that is produced by a straightforward recursive substitution process using string templates associated with each operator. Italicized variables below denote parameters that remain to be bound, e.g., by further communication.

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

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 scheduling a trip."
```

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?"
```

Recipe Library

At its most abstract, a *recipe* is a resource used to derive a sequence of steps to achieve a given goal (the objective of the recipe). Although very general, application-independent recipes exist, such as divide and conquer, we are primarily concerned here with application-specific recipes.

In our implementation, a recipe is concretely represented as a partially ordered sequence of act types (steps) with constraints between them. The recipe library contains recipes indexed by their objective. There may be more than one recipe for each type of objective.

The recipe library for the example application contains 8 recipes defined in terms of 15 different goal or action types. It is probably about half the size it needs to be to reasonably cover the application domain.

Recipes with a fixed number of steps are easily represented in our simple recipe formalism. However, in working on our example application, we quickly discovered the need for more complicated recipes whose step structure depends on some parameters of the objective. For example, two common toplevel recipes for scheduling a trip are working forward and working backward. The working-forward recipe works on the legs of a trip in order starting with the first leg; the working-backward recipe starts with the last leg. In both cases, the number of steps depends on the length of the route.

Rather than "hairing up" our recipe representation as each difficult case arose, we decided instead to provide a general-purpose procedural alternative, called *recipe generators.* Recipes such as working forward/backward are represented in Collagen as procedures which, given an objective, return a recipe. A predicate can also be associated with a recipe to test whether it is still applicable as it is being executed.

A related category of application-specific procedures in the recipe library are *recipe recognizers.* These are primarily used for the bottom-up grouping of a sequence of similar actions on the same object into a single abstract action. For example, such a recognizer is invoked in our example application when the user moves the same time constraint indicator back and forth several times in a row.

Example Application

Our example application concerns solving problems such as the following:

> You are a Boston-based sales representative planning your trip home from San Francisco. On the way, you would like to meet a customer in Dallas who is only available afternoons between 1 and 4 p.m. You would like to avoid overnight flights and fly on American Airlines.

Scheduling a more complicated trip, e.g., one involving four or five cities, typically takes a user about 15 minutes and entails about 150 user or agent actions.

Interface Agents

In addition to illustrating the general framework for agent-user collaboration shown in Figure 1, our example agent is more specifically an *interface agent* (see Figure 3). The collaborative interface agent paradigm 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.

In the collaborative interface agent paradigm, the agent can both communicate with and observe the actions of the user. One of the agent's main responsibilities is to maintain the history and context of the collaboration.

The agent can also interact directly with the shared application program. The agent queries the state of the application using the application's programming interface (API). The agent modifies the application state using the same graphical interface as the user, so that the user can observe the agent's actions.

Our concept of an interface agent is similar to Maes's (1994), although she uses the term "collaborative" to refer to sharing information between multiple software agents, rather than collaboration between agents and people.

Figure 4 shows how the architecture of Figure 3 is realized on a user's display. The large window labelled "Application" is 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.

The smaller overlapping windows in the upper-right and lower-left corners of the screen in Figure 4 are the agent's and user's *home windows*, through which they communicate with each other. These windows are moveable and expand and shrink as their use changes. In a typical session, application actions by the user and agent are interleaved with communication actions between them. The agent manipulates objects in the application window using the hand icon currently shown at rest in its home window.

For additional details on some of the implementation issues related adding an interface agent to graphical user interfaces, see (Rich & Sidner 1996a).

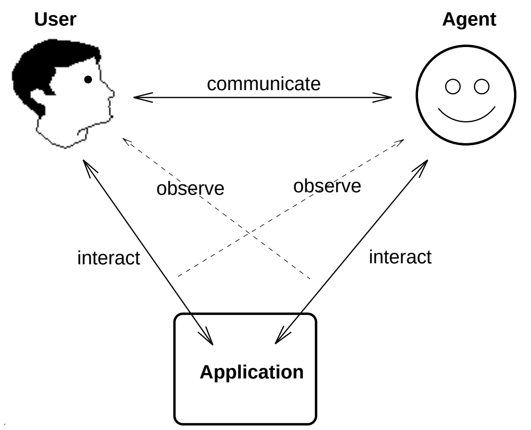


Figure 3: Collaborative interface agent paradigm.

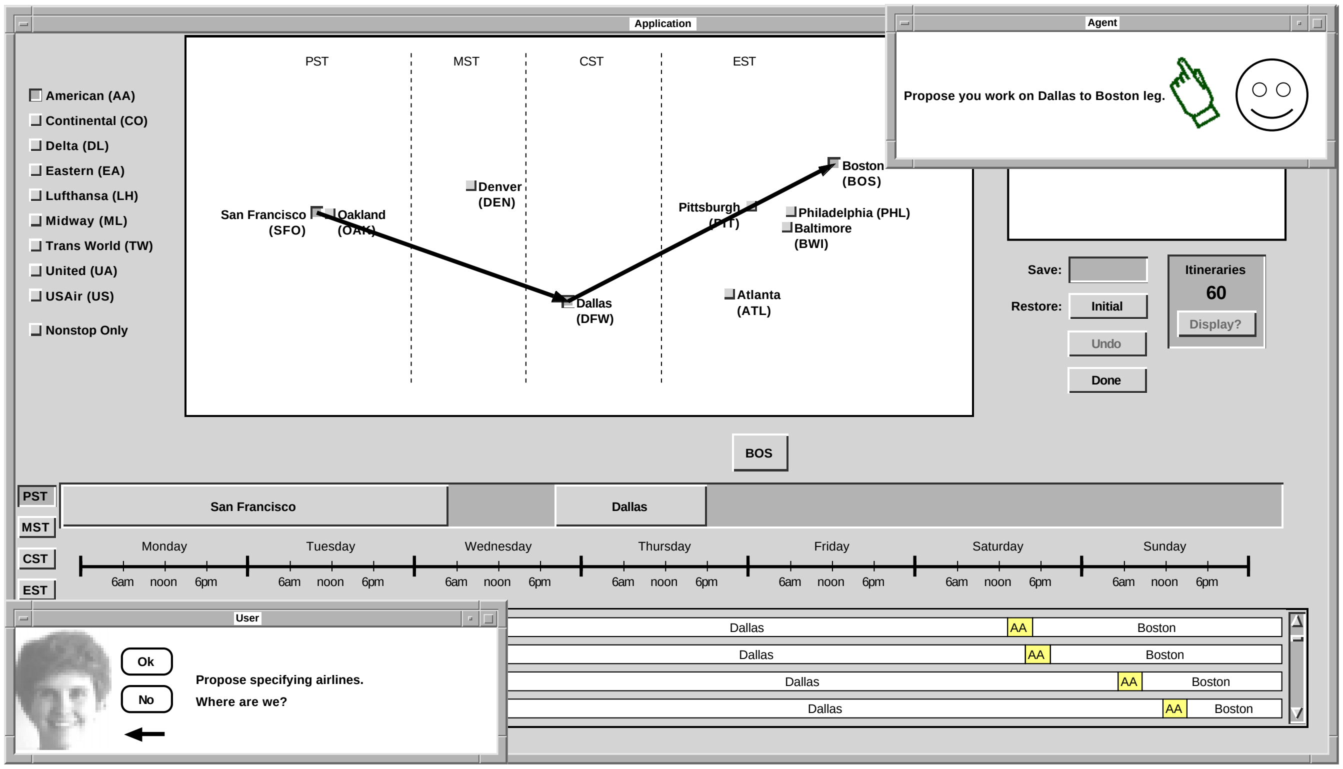
Sample Session

The user could just start working by herself on scheduling the San Francisco-Dallas-Boston trip above by directly manipulating objects in the application window. Instead, she chooses to communicate with the agent to initiate a collaboration. Clicking the arrow at the bottom of her home window causes the window to expand showing her current communication menu:

Propose scheduling a trip.
Where are we?

There is only one possible collaboration to propose here, since this example system was built with only one toplevel goal in mind. A typical real application would have a range of high-level goals. The agent indicates its acceptance of the user's proposal by displaying "Ok" in its home window. Note that at this point the agent has only its generic knowledge of the typical tasks involved in scheduling a trip and recipes for performing them. It does does not know anything about the user's particular problem or preferences.

The user now clicks in order on three cities on the map. The agent recognizes these three actions as forming a segment whose purpose is to identify one of the parameters of the current goal, i.e., the route of the trip. If the user requests a display of the segmented interaction history at this point (by choosing "Where are we?" from the communication menu), the following would be displayed in the agent's home window:



Propose you work on Dallas to Boston leg.

Propose specifying airlines.
Where are we?

Dallas	AA	Boston
Dallas	AA	Boston
Dallas	AA	Boston
Dallas	AA	Boston

Figure 4: Example application screen.

(Jointly scheduling a trip on the route)
USER: "Propose scheduling a trip."
AGENT: "Ok."
(User identifying route of scheduling a trip as San Francisco to Dallas to Boston)
USER: Add San Francisco to the route.
USER: Add Dallas to the route, allowing 97 itineraries.
USER: Add Boston to the route, allowing 100+ itineraries.

Notice that the interaction history above is hierarchically structured. The purpose of each segment is shown in parentheses at the start of the segment. The elements of a segment are either primitive actions or subsegments. Communication actions are indicated by quotation marks. In general, the purpose of a subsegment contributes to the purpose of the parent segment.

The user now clicks "Ok" in her home window to signal the end of the current segment. After a brief pause to see if the user is going to do something else, the agent takes the initiative and asks:

How should a trip on the route be scheduled?

This question is an example of intelligent assistance in which the agent helps the user focus on what needs to be decided next in order to push the task forward. The user is free either to answer the agent's question or to ignore it and proceed on her own. If she chooses to answer, the following option will be presented in her communication menu:

Propose scheduling a trip on the route via ___ .
working forward
working backward

The user chooses the working forward recipe and then asks "What should be done next?", to which the agent replies:

Propose you work on San Francisco to Dallas leg.

Here again the agent uses the current context to assist the user, in this case to suggest the next step in the current recipe. Working on a leg entails manipulating the "interval bars" in the horizontal slider area below the map (see Figure 4) to specify latest arrival and earliest departure times at each city.

We skip ahead now to the end of the sample session, where the interaction history is as shown in Figure 5. Notice the varying level of detail in this history: the

(Jointly scheduling a trip on the route via working forward)
(Done user identifying route of scheduling a trip as San Francisco to Dallas to Boston, allowing 100+ itineraries)
(Done user proposing a trip on the route be scheduled via working forward)
(Done user working on San Francisco to Dallas leg, allowing 70 itineraries)
(Done user working on Dallas to Boston leg, allowing 55 itineraries)
(Done jointly specifying airlines, allowing 10 itineraries)
USER: Add American specification, allowing no itineraries.
(Done agent adding United specification)
AGENT: "Propose I add United specification."
USER: "Ok."
AGENT: Add United specification, allowing 10 itineraries.
(Done user displaying itineraries)

Figure 5: History at end of sample session.

selected segment in which the agent and user jointly specify airlines is shown in full, while the internal elements of the other segments are suppressed. The user can interactively control the level of detail in interaction histories by single and double clicking on segments similarly to the way hierarchial file structures are typically inspected.

The airline specification segment in Figure 5 is expanded in full to show an example of the agent performing an application act, rather than just communicating with the user. The user began this segment by requiring all flights use American airlines, which resulted in no possible itineraries. Whenever a constraint (such as an airline specification) is entered or changed, the application program automatically recomputes the number of possible itineraries and displays this number in the box labelled "Itineraries." In the segmented interaction history, the number of possible itineraries after each segment or action is indicated if it is different from the number before.

Noticing that the user had over-constrained the trip, the agent proposed adding United airlines which, with the user's approval, it did. The agent did not propose this particular airline at random—it used the application's API to find an airline that would in fact increase the number of possible itineraries.

Related Work

Cohen (1992) and Jacob (1995), among others, have explored discourse-related extensions to direct manipulation interfaces that make previous context directly available. However, most work on applying human discourse principles to human-computer interaction, e.g., (Lambert & Carberry 1991, Yanklovich 1994), has assumed that natural language understanding will be applied to the user's utterances. Terveen (1991) has explored providing intelligent assistance through collaborative graphical manipulation without explicitly invoking the agent paradigm.

The two systems we know of that are overall closest in spirit to our own are Stein et al.'s MERIT (1995) and Ahn et al.'s DenK (1994). MERIT uses a different version of 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.

Conclusion

We have demonstrated the feasibility of an application-independent toolkit for applying human collaborative discourse principles to autonomous software agents. We hope the sample scenario above suggests that this approach can lead to agents that are natural and easy to collaborate with.

Our future plans include:

- improving the flexibility and robustness of the discourse processing algorithms, especially as related to incompleteness of the agent's recipe library,
- supporting negotiation between the user and agent,
- a pilot user study to compare using the example application with and without the interface agent, and
- using Collagen to build agents that operate remotely in space and time (e.g., on the Internet), which will require more discussion between the agent and user about past and future actions.

References

Ahn et al, R. 1994-5. The DenK-architecture: A fundamental approach to user-interfaces. *AI Review* 8:431–445.

Brodie, M.; Mylopoulos, J.; and Schmidt, J., eds. 1982. *On Conceptual Modelling.* NY, NY: Springer-Verlag.

Cohen, P. 1992. The role of natural language in a multimodal interface. *UIST'92*, pp. 143–149.

Grosz, B. J., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence.* To appear.

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

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

Jacob, R. J. K. 1995. Natural dialogue in modes other than natural language. In Beun, R.-J.; Baker, M.; and Reiner, M., eds., *Dialogue and Instruction.* Berlin: Springer-Verlag. 289–301.

Kautz, H. 1990. A circumscriptive theory of plan recognition. In Cohen, P. R.; Morgan, J. L.; and Pollack, M. E., eds., *Intentions and Communication.* Cambridge, MA: MIT Press. chapter 6, 105–133.

Lambert, L., and Carberry, S. 1991. A tripartite plan-based model of dialogue. In *Proc. 29th Ann. Meeting ACL.*

Lochbaum, K. E. 1994. Using collaborative plans to model the intentional structure of discourse. TR 25-94, Harvard Univ., Ctr. for Res. in Computing Tech. PhD thesis.

Lochbaum, K. E. 1995. The use of knowledge preconditions in language processing. *IJCAI'95*, pp. 1260–1266.

Maes, P. 1994. Agents that reduce work and information overload. *Comm. ACM* 37(17):30–40.

Rich, C., and Sidner, C. 1996a. Adding a collaborative agent to graphical user interfaces. *UIST'96.* To appear.

Rich, C., and Sidner, C. 1996b. Segmented interaction history in a collaborative interface agent. *3rd Int. Conf. on Intelligent User Interfaces.* To appear.

Rich, C. 1996. Window sharing with collaborative interface agents. *ACM SIGCHI Bulletin* 28(1):70–78.

Sidner, C. L. 1994. An artificial discourse language for collaborative negotiation. *AAAI'94*, pp. 814–819.

Stein, A., and Maier, E. 1995. Structuring collaborative information-seeking dialogues. *Knowledge-Based Systems* 8(2-3):82–93.

Terveen, G.; Wroblewski, D.; and Tighe, S. 1991. Intelligent assistance through collaborative manipulation. *IJCAI'91*, pp. 9–14.

Yanklovich, N. 1994. Talking vs. taking: Speech access to remote computers. *CHI'94*, pp. 275–276.