

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

COLLAGEN: A Collaboration Manager for Software Interface Agents

Charles Rich, Candace L. Sidner

TR97-21a March 1998

Abstract

We have implemented an application-independent collaboration manager, called Collagen, based on the SharedPlan theory of discourse, and used it to build a software interface agent for a simple air travel application. The software agent provides intelligent, mixed initiative assistance without requiring natural language understanding. A key benefit of the collaboration manager is the automatic construction of an interaction history which is hierarchically structured according to the users' and agents' goals and intentions.

User Modeling and User-Adapted Interaction, Special Issue on Computational Models for Mixed Initiative Interaction

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., 1998
201 Broadway, Cambridge, Massachusetts 02139

COLLAGEN: A Collaboration Manager for Software Interface Agents

Charles Rich Candace L. Sidner*

TR-97-21a March 1998

Abstract

We have implemented an application-independent collaboration manager, called Collagen, based on the SharedPlan theory of discourse, and used it to build a software interface agent for a simple air travel application. The software agent provides intelligent, mixed initiative assistance without requiring natural language understanding. A key benefit of the collaboration manager is the automatic construction of an interaction history which is hierarchically structured according to the user's and agent's goals and intentions.

*To appear in User Modeling and User-Adapted Interaction,
Special Issue on Computational Models for Mixed Initiative Interaction.*

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, 1998
201 Broadway, Cambridge, Massachusetts 02139

*Lotus Development Corporation

Publication History:–

1. First printing, TR-97-21a, March 1998

COLLAGEN: A Collaboration Manager for Software Interface Agents

CHARLES RICH

MERL—A Mitsubishi Electric Research Laboratory, Cambridge, Massachusetts, USA

CANDACE L. SIDNER

Lotus Development Corporation, Cambridge, Massachusetts, USA

March, 1998

Abstract. We have implemented an application-independent collaboration manager, called Collagen, based on the SharedPlan theory of discourse, and used it to build a software interface agent for a simple air travel application. The software agent provides intelligent, mixed initiative assistance without requiring natural language understanding. A key benefit of the collaboration manager is the automatic construction of an interaction history which is hierarchically structured according to the user's and agent's goals and intentions.

Key words: agent, collaboration, mixed initiative, SharedPlan, discourse, segment, interaction history.

1. Introduction

Current mixed-initiative interactive systems can be difficult to use: the order in which actions must be performed by the user and the system 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 design as viewed over a single or short sequence of interactions, but rather that current systems lack support for the user's problem solving *process* as it unfolds over extended periods of time. The overall goal of this research is to develop a new paradigm for human-computer interaction which explicitly supports the user's problem-solving process based on current theories of collaborative discourse.

Our most fundamental underlying assumption is that a human-computer interface based on familiar human discourse rules and conventions will be easier for people to learn and use than one that is not. Although we cannot yet confirm this assumption by our own empirical studies, we are encouraged by an analogy with the direct-manipulation paradigm for graphical user interfaces, whose success we believe is due in large part to users' familiarity with the rules and conventions of object manipulation in everyday life.

Much is known about the structure of human collaboration and discourse. In this work, we rely specifically on the SharedPlan work of Grosz and Sidner (1986; 1990), Grosz and Kraus (1996), and Lochbaum (1994; 1995; 1998). This work provides us with a well-specified computational theory that has been empirically validated across a range of human tasks. Section 3 describes the basic algorithms we use from SharedPlan theory. Section 7.1 discusses the relationship of this theory to the issue of global initiative in conversation.

A second important methodological choice has been to support the problem-solving level of human-computer interaction via the “software agent” paradigm. Specifically, we take the approach of adding a collaborative software agent to an existing graphical user interface. 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.

Finally, at the engineering level, our approach has been to develop an application-independent *collaboration manager* for software agents, called Collagen* (for *Collaborative agent*). A collaboration manager is a software component that mediates the interaction between a software interface agent and a user. It is similar to what is often called a “discourse manager,” except that it keeps track of not only the linguistic and attentional state of a discourse, but also the collaborative intentions of the participants. However, it is less than a fully automated planning system, because it does not by itself decide what the agent should do or say next (though it may provide some candidates); it primarily provides a representation for recording the decisions that the agent has made and communicated. Collagen also introduces a new graphical interface device, called the *segmented interaction history*, for displaying and manipulating the state of a collaborative problem-solving session.

To summarize the organization of the remainder of this paper: After further general discussion of our approach, we present a complete example session (Section 2) with an air travel planning application and agent we have built to demonstrate the functionality of Collagen. Following the example session, we review the relevant discourse theory and algorithms (Section 3), in preparation for a discussion of the application-independent architecture of Collagen (Section 4) and, more specifically, the problem-solving state transformations enabled by the segmented interaction history (Section 5). Section 6 focusses on how application-specific task knowledge is represented in Collagen. Section 7 reviews the mechanisms in Collagen and in our example agent which support mixed initiative. Finally, there is a comparison with other work and conclusions.

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

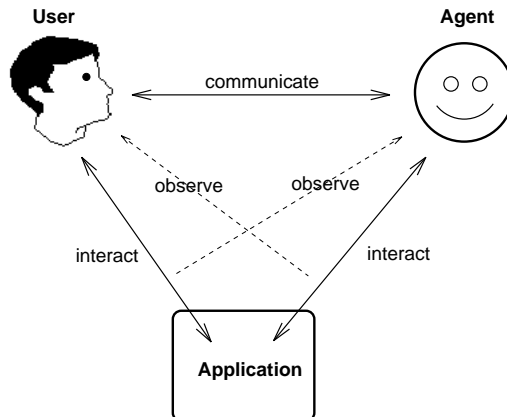


Figure 1. Collaborative interface agent paradigm.

1.1. COLLABORATIVE INTERFACE AGENTS

Our version of the software agent paradigm (Maes, 1994), which we term a *collaborative interface agent*, is illustrated in Figure 1. This 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.

Notice that the software agent is able to both communicate with and observe the actions of the user and vice versa. A crucial part of successful collaboration is knowing when a particular action has been performed. In the collaborative interface agent paradigm, this can occur two ways: either by a reporting communication (“I have done x ”) or by direct observation. Currently in Collagen, we treat both of these cases equivalently, since we are assuming a kind of close collaboration in which both participants know and intend that all their actions are observed. However, both the underlying theory and architecture of Collagen equally well apply to a situation in which some actions cannot be observed, but only reported, such as if the agent is performing actions at a remote network site.

Another symmetrical aspect of the collaborative interface paradigm in Figure 1 is that both the user and the agent can interact with the application program. There are a number of design alternatives regarding how the agent’s interaction with the application program is implemented (see Section 2.2). Typically, the agent queries the application state using the application’s programming interface (API). The agent may modify the application state either via the API or via the graphical interface.

Although, in the long run, communication between users and interface agents will very likely be in spoken natural language, we have decided for both practical and methodological reasons *not* to include natural language

understanding in our current system. 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 methodological 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 the communication is in.

1.2. MIXED INITIATIVE

The mixed-initiative capabilities of a particular agent, such as the air travel agent described in the next section, arise from the interplay of two sources: the application-independent algorithms and data structures in Collagen and application-specific code and libraries “inside” the agent. In terms of Figure 1, one can think of Collagen as a component of the agent. Alternatively (looking ahead to Figure 8), one can think of the collaboration manager as standing between the agent and the user.* In either case, Collagen supports mixed-initiative by interpreting discourse acts and maintaining a model of the achieved and expected tasks and goals of the user and agent, thereby eliminating the need to re-invent these facilities every time a new agent is built. As we will see in the next section, given Collagen as a base, an agent with a surprising richness of mixed initiative interaction can be implemented with the addition of very little application-specific programming.

2. An Example Application and Agent

In this section, we first describe the air travel application program we implemented to serve as a test bed for the development of Collagen. We then discuss some of the issues involved in adding an interface agent to this application, culminating with a detailed walkthrough of an implemented example session.

2.1. THE AIR TRAVEL APPLICATION

We wanted our test 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 application state is graphically visible and modifiable.

Figure 2 shows the graphical user interface to the air travel planning system we implemented in Common Lisp using the Garnet graphics package (Meyers et al., 1990). (For the discussion in this subsection, please ignore the

* This view more obviously generalizes to multiple software agents and users, which is a direction that we may, but have not yet, pursued.

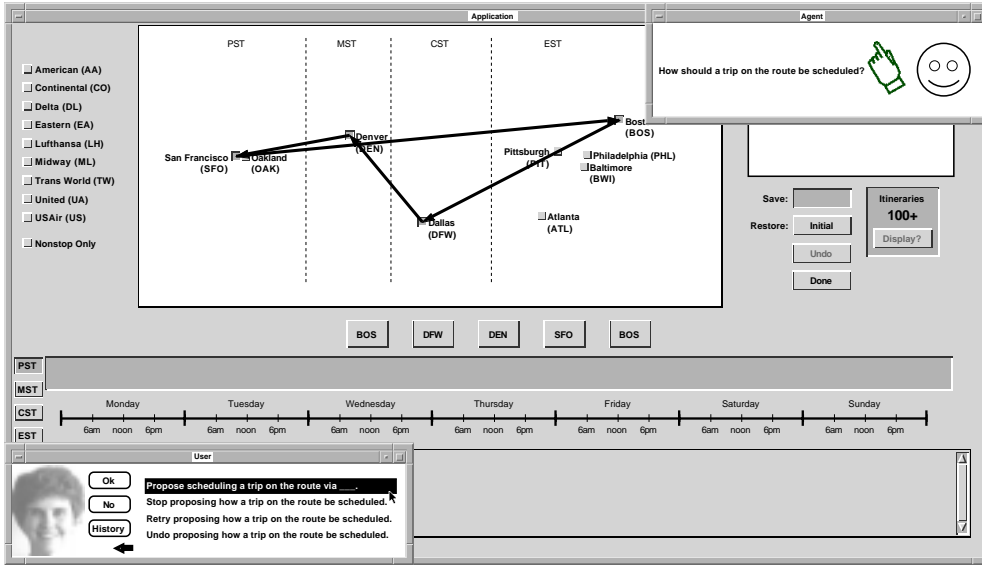


Figure 2. Test application screen.

overlapping windows in the upper-right and lower-left corners.) The application provides a direct-manipulation interface to an airline schedule data base 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, because 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 to attend your son's piano recital.

In order to gain some initial intuitions about the problem-solving process using this application, we asked seven visitors and staff members at MERL 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 application. 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, and final destination for the trip. Next,

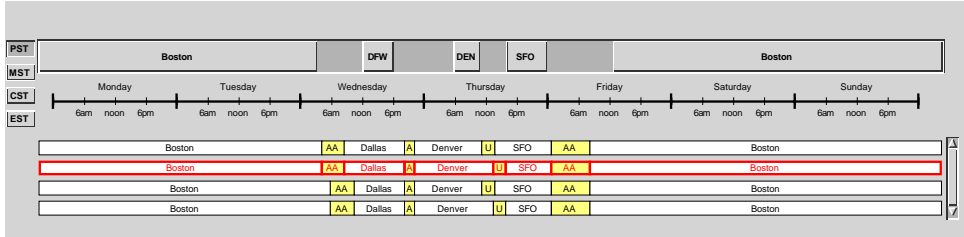


Figure 3. Interval bars and displayed itineraries in test application interface.

users typically manipulate some of the small rectangles labelled with city names. These city “interval” bars 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 (see Figure 3). 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 flight schedule data base 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 3).

In general, users find that displaying more than about ten 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 application program does not allow display when there are zero (over-constrained) or more than 100 itineraries (under-constrained).

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 a typical Undo button and a “snapshot” facility (via the Save button and Restore menu), these were not very convenient to use. For example, the snapshot facility requires users to interrupt their work flow to choose a name for and explicitly save the current state in anticipation of possibly needing to return to it later.

One of our general observations 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 this application.

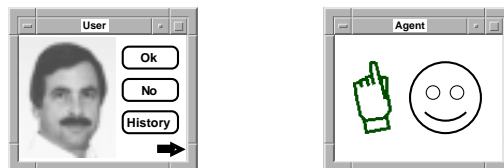
Another important property of these scenarios is that the initial problem statement is only partially formalizable within the system.* Taking the example problem statement above, notice that the application program does not provide a representation to specify in advance (even if you could) just how much 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.

2.2. ASYNCHRONOUS WINDOW SHARING

The first step in adding a collaborative interface agent to the air travel application is to establish the basic communication, observation, and interaction channels required by the paradigm as shown in Figure 1. This was achieved using a window-sharing layer implemented in the X Window System and described in detail elsewhere (Rich, 1996).

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.



To support asynchronous mixed-initiative interaction, the agent and its home window are serviced by a separate process from the application and user home window. 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 process

* A closely related property is that there is usually more than one "right" answer.

may run on a different machine than the user/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 was realized with the air travel application:

- 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 (see Section 6.1) is transmitted to the input buffer of the agent process.
- The user interacts with the application in the usual way, modifying the application state with her cursor and “querying” it with her eyes.
- The agent observes the user’s actions by virtue of a generic layer in the application that mirrors semantic actions into the input buffer of the agent process.
- The agent modifies the application state with its cursor (see discussion of “unGUI” module in (Rich, 1996)) 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 final two points above allow some design alternatives, which may be explored with other target applications. In particular, it may sometimes be difficult to support agent manipulation of the shared graphical interface by generating mouse-level events. A weaker alternative to the complete agent/user symmetry above would be for the agent to modify the application state through the API and just the use its cursor to point to the location where the display changed.

2.3. EXAMPLE SESSION

Figure 4 shows the segmented interaction history of a complete session between a user and our collaborative agent solving the problem posed in Section 2.1 above. This is only one of many possible histories with different structures, depending on what the user chooses to do at each point in time. Also, as we will see in Section 5, this history is an explicit object in the interaction, whose structure can be referred to by both the agent and the user.

The contents of the Figure 4, including indentation, is automatically generated by Collagen (in a separate history window) as a printout of data structures built up during the session by the algorithms described in the next section. Event numbers have been added at the left for exposition.

Indentation in the history indicates the nesting of collaborative goals. The phrase in parentheses at the start of each level of indentation indicates the

```

(Jointly scheduling a trip on the route via working backward, allowing 4 itineraries)
1  USER: "Propose scheduling a trip."
2  AGENT: "Ok."
(Done user identifying route of scheduling a trip as Boston to Dallas to Denver
  to San Francisco to Boston, allowing 100+ itineraries)
3  USER: Add Boston to the route.
4  USER: Add Dallas to the route, allowing 97 itineraries.
5  USER: Add Denver to the route, allowing 100+ itineraries.
6  USER: Add San Francisco to the route, allowing 100+ itineraries.
7  USER: Add Boston to the route, allowing 100+ itineraries.
8  USER: "Ok."
(Done user proposing a trip on the route be scheduled via working backward)
9  AGENT: "How should a trip on the route be scheduled?"
10 USER: "Propose scheduling a trip on the route via working backward."
(Done user working on San Francisco to Boston leg, allowing 70 itineraries)
11 AGENT: "Propose you work on San Francisco to Boston leg."
12 USER: Add Boston final interval with arrival Sun 11:30p.
13 USER: Change Boston final interval to arrival Fri noon.
14 USER: Add San Francisco stopover with arrival Wed 1:55a departure Wed 9:55a,
    allowing 70 itineraries.
15 USER: Change San Francisco stopover to arrival Thu 6:00p departure Fri 1:57a.
16 USER: "Ok."
(Done user working on Denver to San Francisco leg, allowing 61 itineraries)
17 AGENT: "Propose you work on Denver to San Francisco leg."
18 USER: Add Denver stopover with arrival Thu 8:21a departure Thu 3:19p.
...
(Done jointly specifying airlines, allowing 4 itineraries)
36 USER: Add American specification, allowing no itineraries.
(Done agent adding United specification, allowing 4 itineraries)
37 AGENT: "Propose I add United specification."
38 USER: "Ok."
39 AGENT: Add United specification, allowing 4 itineraries.
(User displaying and selecting itineraries)
40 USER: Display 4 itineraries.
41 USER: Select itinerary Wed AA/1099 Wed AA/313 Thu UA/343 Fri AA/928.

```

Figure 4. Segmented interaction history for example session.

goal (purpose) being achieved by that segment. A more formal specification of the segmented interaction history representation will be provided in Section 3 after the underlying discourse theory terminology has been formally defined.

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.

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."

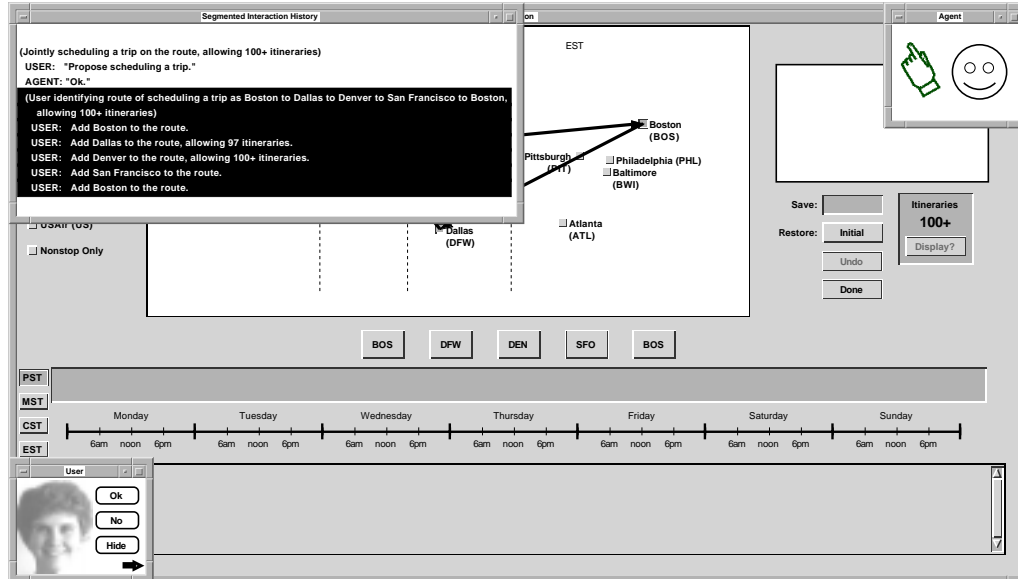



Figure 5. Pop-up window for segmented interaction history.

In the segmented interaction history, the number of possible itineraries after each segment or event is indicated if it is different from the number before.

Given the problem statement in Section 2.1 above, the user could just start working on scheduling this trip herself by pointing and clicking on the application window. Instead, she chooses to communicate with the agent to initiate a collaboration (event 1). As we will see in the ensuing conversation, the agent not only responds to the user's initiative here, but will also take the initiative itself at appropriate times in the joint activities.

In order to communicate with the agent, the user clicks on the arrow at the bottom of her home window, causing the window to expand to show her the current communication choice(s):

Propose scheduling a trip. 

There is only one possible collaboration to propose here, since this test application was built with only one toplevel goal in mind. A 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 (event 2). This simple utterance by the agent not only communicates acceptance of the proposed action to the user; it also reflects the creation within Collagen of an initial model of the purpose and expected elements of the collaboration.

Note that at this point, however, the agent has only generic knowledge of the typical tasks involved in scheduling a trip and recipes (general methods) for performing them. It does not know anything about the user's particular problem or preferences.

The user now clicks in order on cities on the map: Boston, Dallas, Denver, San Francisco, Boston. The agent recognizes these 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.

The segmented interaction history is computed incrementally. Its most basic function is to orient the user. For example, if the user left her computer in the middle of working on this problem and returned after a few hours (or days), the history would help her reestablish where in the problem solving process she was. In particular, if the user requested a display of the history at this point (by pressing the “History” button in her home window), the pop-up window shown in the upper left corner of Figure 5 would appear with the current segment highlighted.

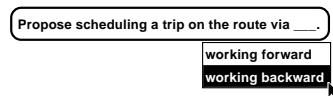
The user clicks on the “Ok” button (event 8) in her home window to signal the end of the current segment. The agent now takes the initiative and in its home window asks (event 9):

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 current task forward. Unlike conventional pop-up help windows, however, the user at this point can delay or ignore the agent’s question simply by continuing to click on the application window.

Figure 2 shows the moment in the interaction history after the agent’s question above when the user is about to answer by choosing a response from the communication menu in her home window. This response is event 10 in the segmented interaction history. Notice that the user’s first menu choice has a blank at the end where the name of the recipe is supposed to go. (See Section 5 for an explanation of the other choices.)

Once a communication menu choice is made, the user may be presented with another menu, such as the one below, to choose from the allowable values for any unspecified parameters of the communication action:



Our agent currently knows only two generic toplevel recipes for scheduling a trip on a given route, gleaned from our informal observations of test application users and study of transcripts of people talking on the phone to real travel agents (Kowtko and Price, 1989). These two recipes are: *working forward* on the legs of the trip starting at the originating city and *working backward* starting at the final destination.

The user chooses working backward (event 10), presumably because of the hard constraint on attending the piano recital after the trip, after which the agent says:

Propose you work on San Francisco to Boston leg.

Here again the agent uses the current context to assist the user, in this case to propose the next subtask. Working on a leg entails manipulating the “interval bars” in the horizontal slider area below the map to specify latest arrival and earliest departure times at a city (see Figure 3).

Notice that the agent’s response here results from a simple application-independent strategy, namely to propose the next (in this case, the first) executable step in the current recipe. The same strategy underlies the agent’s proposal in event 17 after the user finishes working on the San Francisco to Boston leg. As we will see in the architecture discussion in Section 4, this application-independent strategy is embedded in the collaboration manager, even though the recipes themselves are application-specific.

We skip ahead now to event 36. As mentioned earlier, we observed that one of the main ways that users of the test application get stuck is by over-constraining their itinerary, causing the itinerary count to become zero. Here the user has required all flights to use American airlines, which resulted in no possible itineraries.

The agent’s response (event 37) is to suggest reducing constraints by adding United to the set of allowable airlines. The agent did not propose this particular constraint change at random—it used its ability to access the test application’s full API to search the flight data base for an airline that would in fact increase the number of itineraries. This is a good example of where the capabilities of one participant in collaboration are usefully different from those of another. With the user’s permission (event 38), the agent uses its hand cursor to click on the United airlines button (event 39).

The interaction ends with the user displaying the four possible itineraries in the scrolling window (see Figure 3) and selecting one, which causes detailed information, such as flight numbers, to be printed in the message window above the Itineraries count in the application interface.

2.4. MIXED INITIATIVE IN THE EXAMPLE SESSION

The mixed-initiative behavior in the example session above arises from the interplay of two sources: the application-independent discourse model embodied in Collagen, and application-specific knowledge “inside” the air travel agent. Application-specific knowledge can be further decomposed into two forms: (i) a library of recipes that specify the typical steps and constraints for achieving certain goals and (ii) arbitrary pattern-action rules.

For example, the agent’s contributions in events 9, 11, 17 and 39 arise from the application of the following simple and generic discourse principles

to the collaborative discourse state at each step (these principles and others used in Collagen are described in more detail in the next section):

- A recipe needs to be identified for the current goal.
- Identification of a recipe may be achieved by asking the other participant.
- A goal or action may be performed when all of its parameters are known and all of its predecessors (in the current recipe) have been achieved.
- A goal or action may be performed by any participant who is capable, unless a specific participant has been specified.

In particular, the agent’s question in event 9 arises from the application of the first and second principles above to the state in which no recipe has yet been identified for the goal of scheduling a trip. The agent’s proposals in events 11 and 17 arise from the application of the third and fourth principles above to the steps of the working-backward recipe. (Only the user is capable of working on these goals, because only the user knows her travel preferences). The agent’s manipulation action in event 39 also arises from the application of the third and fourth principles above, together with an additional collaboration-specific rule which requires the agent to seek explicit agreement from the user (event 37) before performing any manipulation actions.

The content of the agent’s proposal in event 37 (i.e., the choice of United airlines) arises from an application-specific pattern-action rule for overconstrained situations, as discussed in the preceding section. Collagen plays an important role even in the case of these arbitrary pattern-action rules, since the pattern (situation description) part of such rules typically depends not only on the application state, but also on the current discourse state.

3. Collaborative Discourse Theory and Algorithms

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.

This section provides a brief overview of the collaborative discourse theory and algorithms on which Collagen is based. Since the goal of this work is to use well-established human discourse principles, readers are referred to the referenced literature for more details.

Collaborative discourse in Grosz and Sidner’s framework (Grosz and Sidner, 1986; Grosz and Sidner, 1990) is understood in terms of three interrelated kinds of discourse structure:

- intentional structure, which is formalized as partial *SharedPlans*.

- linguistic structure, which includes the hierarchical grouping of actions into *segments*.
- attentional structure, which is captured by a *focus stack* of segments.

We summarize the key features of each of these structures below, followed by a concrete example of Collagen’s discourse state representation. Finally, we describe the discourse interpretation and generation algorithms, which are the heart of Collagen’s discourse processing.

3.1. SHAREDPLANS

Grosz and Sidner’s theory predicts that, for successful collaboration, the participants need to have mutual beliefs* about the goals and actions to be performed and the capabilities, intentions, and commitments of the participants. The formal representation (Grosz and Kraus, 1996) 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 data base 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 data base);
- 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).

Several important features of collaboration should be noted here.

First, due to partial knowledge of the shared environment and each other, participants do not usually begin a collaboration with all of the conditions above “in place.” They typically start with only a *partial* SharedPlan. An important purpose of the communication between participants is to determine (possibly with the help of individual information gathering) 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.

* 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.

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 (Bratman et al., 1988).) Collagen therefore does not currently provide a generic framework for execution. Another way of saying this is that we provide a generic framework only for *recording* the order in which planning and execution occur, not for *deciding* how to interleave them.

3.2. DISCOURSE SEGMENTS AND FOCUS STACK

The concept of discourse segments is at the very foundation of discourse theory. Analysis of discourses from a range of human interactions has resulted in general agreement that discourse has a natural hierarchical structure. The elements of this hierarchy are called *segments*. A segment is a contiguous sequence of communicative actions 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.

The existence of segments can be seen in everything from pitch patterns in spoken discourse to the way that pronouns are interpreted. Automatic segmentation (i.e., the segmented interaction history) has therefore been our first milestone in applying discourse principles to human-computer interaction.

A simple example of segments in a task-oriented human discourse is shown in Figure 6, which is adapted from (Grosz [Deutsch], 1974). In this discourse, participant A is instructing participant B how to repair an air compressor. Notice that this analysis of discourse structure includes not only the participants' utterances, but also their actions (e.g., B removes belt). This is appropriate in a context, such as collaborative interface agents, where all actions on the shared artifact are known and intended to be mutually observable.

The toplevel segment and three embedded segments in Figure 6 are indicated by the brackets and indentation shown (further subsegments are elided). In Grosz and Sidner's theory, the segment structure of such 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 SharedPlan conditions described above.

For example, the purpose of the toplevel segment in Figure 6 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

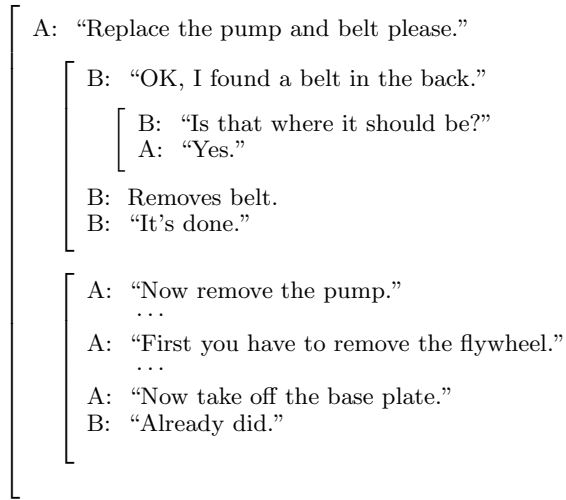


Figure 6. Segments in a task-oriented human discourse.

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.

The shifting focus of attention in a discourse is captured by a *focus stack* of discourse segments. In the natural flow of a collaborative discourse, new segments and subsegments are created, pushed onto the focus stack, completed, and then popped off the stack as the SharedPlan unfolds in the conversation. Sometimes participants also interrupt each other, abandon the current SharedPlan even though it is not complete, or return to earlier segments.

Thus, as we will see more concretely in the discussion of Figure 7 below, the attentional (focus stack) and intentional (SharedPlan) aspects of discourse structure theory are connected through the discourse segment purpose: each segment on the stack is associated with a SharedPlan for its purpose (Lochbaum, 1994; 1998).

3.3. DISCOURSE STATE REPRESENTATION IN COLLAGEN

The *discourse state* in Collagen is a concrete representation of the three kinds of discourse structure described above. Figure 7 shows an example discourse state.

The lower part of Figure 7 shows a *plan tree*, which is an approximate representation of a partial SharedPlan. Plan trees are composed of alternating act and recipe nodes as shown. Both acts and recipes have *bindings*, shown as

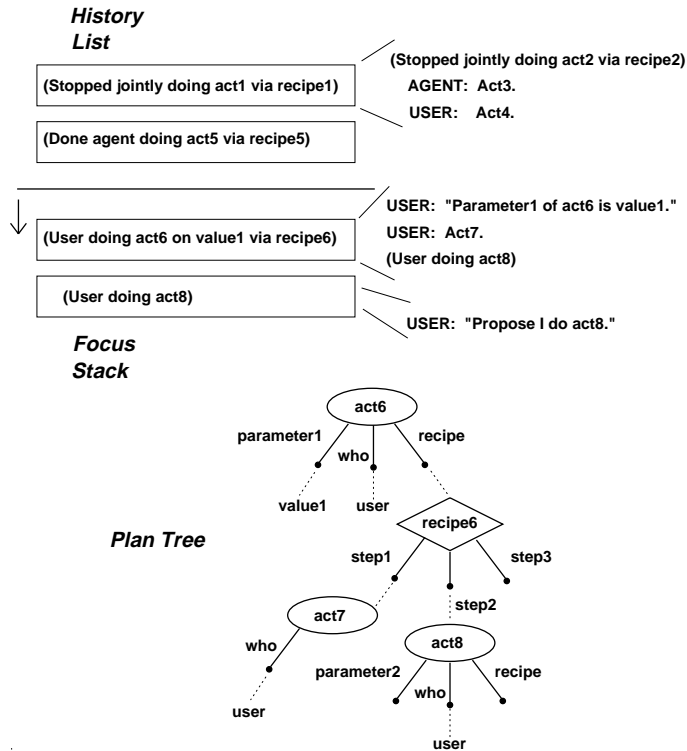


Figure 7. Internal discourse state representation.

labelled stubs in the figure, with constraints between them specified in their recipe library definitions. An act node has a binding for each of its parameters, who performs it and, if it is non-primitive, a recipe node. A recipe node has a binding for each step in the recipe. To support the nonmonotonic changes in discourse state required for negotiation and history-based transformations (Section 5), bindings and the propagation of logical information in the plan tree are implemented using a truth-maintenance system.

For example, in Figure 7, act6's sole parameter has been bound to value1 and act6's recipe has been bound to recipe6. If a history-based transformation "undoes" act7 and act8, then act6's recipe binding will be retracted. Similarly, act6's parameter binding will be retracted if the first communication act in its segment is undone.

The upper part of Figure 7 shows the *focus stack* (illustrated as growing downward) and the *history list*, which contains toplevel segments that have been popped off the focus stack. When a segment is popped off the focus stack, it is added to the history list if and only if it has no parent segment. In the figure, there are two segments on the focus stack and two segments

in the history list. The elements of one of the segments on the stack and one of the segments in the history list are shown expanded to the right.

Segments on the stack are called *open*, because they may still have acts added to them. Segments that have been popped off the stack are called *closed*. All the segments in the history list and their subsegments are closed. Segments on the stack may have closed subsegments.

Usually, the root of the plan tree (e.g., act6 in Figure 7) is the purpose of the base segment of the focus stack, and each subsegment (e.g., act8) corresponds to a subtree, recursively. The exception is when there is an interruption, i.e., a segment which does not contribute to its parent, in which case we have a disconnected plan tree for that segment. Plan trees remain associated with segments even after they are popped off the stack.

The two key discourse processing algorithms used in Collagen are *discourse interpretation*, which is a reimplementaion of Lochbaum's (1994; 1998) rgraph augmentation algorithm, and *discourse generation*, which is essentially the inverse of interpretation.

3.4. DISCOURSE INTERPRETATION

The main job of discourse interpretation in Collagen is to consider how the current direct communication or observed manipulation action 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.* 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 a 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 current segment (and the partial SharedPlan representation is appropriately updated). Furthermore, if this act completes the achievement of the current discourse segment purpose, the focus stack is popped.

If none of the above cases holds, discourse interpretation concludes that the current action starts an *interruption*, i.e., a segment that does not contribute to its parent. A new segment is pushed onto the focus stack with the current act as its first element. The purpose of this new segment may

* The last three cases are instances of a larger class of explanations that Lochbaum (1995) calls “knowledge preconditions.”

or may not be known, depending on the specific content of the initiating action.

The occurrence of interruptions may be due to actual interruptive material in the ongoing discourse or due to an incomplete recipe which does not include the current act even though it ought to. We take the view that, in general, the agent's knowledge will never be complete and it therefore must deal gracefully with unexpected events.

Another phenomenon which manifests itself as interruptions in the current discourse interpretation algorithm is when the user and/or agent are pursuing two (or more) goals in parallel, e.g., arbitrarily interleaving steps from both recipes. In this situation, some higher level representation of the parallel goals would be preferable to the alternating structure of pushes (interruptions) and pops (stop transformations, see Section 5.1) currently required. This is an area for future work.

It is tempting to think of discourse interpretation as the plan recognition problem, which is known to be exponential in the worst case (Kautz, 1990). However, this misses a key property of normal human discourse, namely that speakers work hard to make sure that their conversational partners can understand their intentions without a large cognitive search. Notice that only search performed by the discourse interpretation algorithm above is through the steps of the current recipe or all known recipes for the current segment's purpose (and this is *not* done recursively). We think it will be reasonable to expect users to communicate enough so that the agent can follow what is going on without having to do general plan recognition.

3.5. DISCOURSE GENERATION

The discourse generation algorithm is, as mentioned above, essentially the inverse of interpretation. It looks at the current focus stack and associated SharedPlan and produces a prioritized *agenda* of (possibly partially specified) actions which would contribute (according to the five cases above) 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 Collagen, the agenda contains communication and manipulation actions by either the user or agent, which would advance the current problem-solving process.

The main reason we believe that the menu approach to user communication demonstrated in Section 2 is workable is because the discourse generation algorithm typically produces only a relatively small number of communication choices, all of which are relevant in the current discourse context.

4. The Collagen Architecture

This section focuses on the main technical contribution of this work, which is to embed the theory-grounded algorithms and data structures described in Section 3 into a practical architecture (Figure 8) for building collaborative interface agents, such as our air travel example. Figure 8 is essentially an expansion of Figure 1 in which the Collagen discourse manager is made explicit as the mediator of all communication between the agent and the user.

All of the internal data flow in Figure 8 takes place using Collagen’s artificial discourse language (see Section 6.1). Whenever there is a need for the user to see information, such as in display of the user communication menu or the segmented interaction history, these internal representations are given an English gloss by simple string substitution in templates defined in the recipe library.

We refer to Collagen as a *collaboration manager*, because it provides a standard mechanism for maintaining the flow and coherence of agent-user collaboration. In addition to saving implementation effort, using a collaboration manager provides consistency across applications, and to the extent it is based on good principles, leads to applications that are easier to learn and use.

Even when using Collagen, however, a developer still must provide considerable application-specific information. After discussing the generic architecture of Collagen below, we will focus in Section 6 on how this application-specific information is provided via the recipe library and the artificial discourse language.

4.1. THE AGENT

Note that the agent itself is a “black box” in Figure 8. We are not trying to provide tools 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—whatever is appropriate for the application. What Collagen does provides is a generic framework for recording the decisions made and communicated by the agent (and the user), but not for *making* them. We believe this is a good software engineering modularity.

As can be seen in Figure 8, Collagen also provides some important new resources (inputs) for a developer to use in implementing the decision-making part of an interface agent: the discourse state, the agenda, and the recipe library.

The default agent implementation that “comes with” Collagen always simply chooses to perform the highest priority action in the current agenda for which the actor is either unspecified or itself. Our example agent was constructed by extending this default implementation only a page of

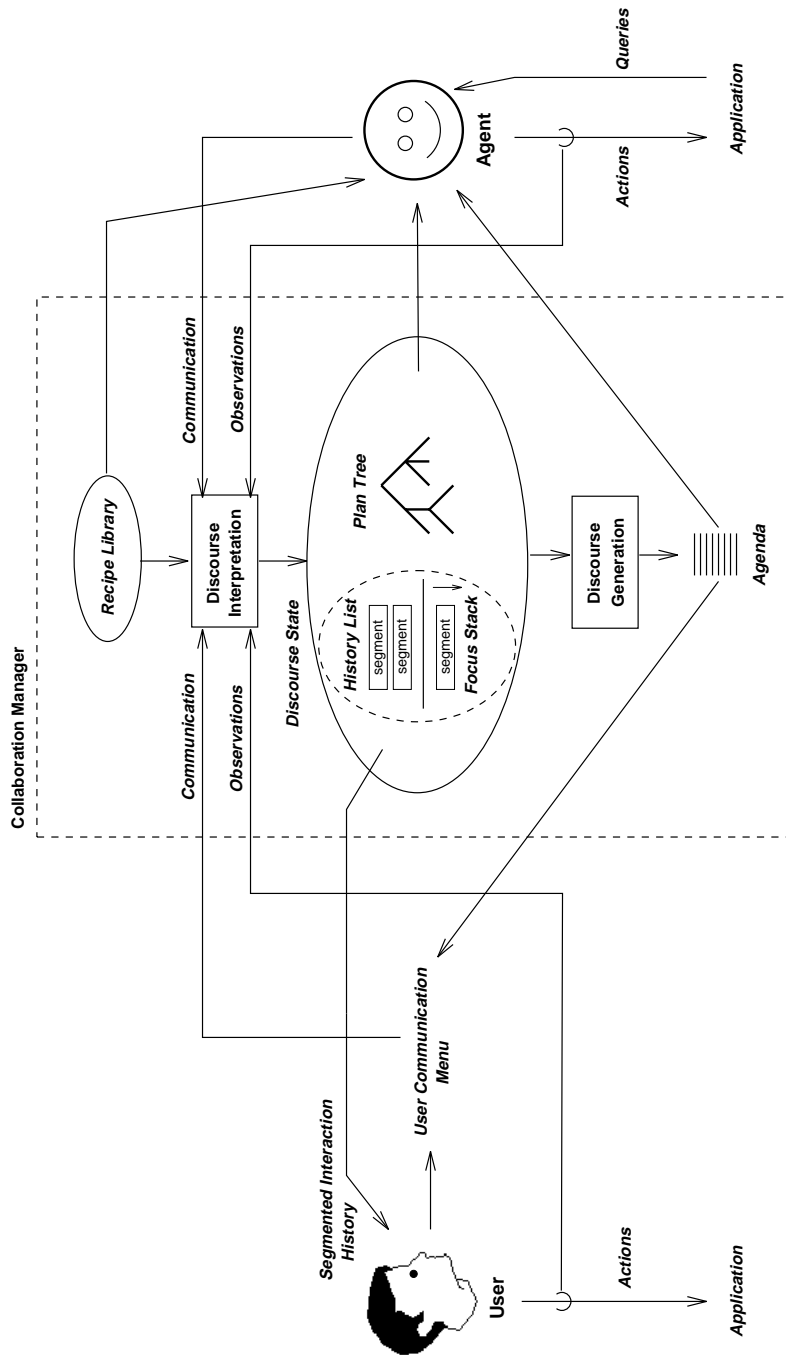


Figure 8. Collagen architecture.

application-specific logic, which sometimes proposed other actions based on querying the discourse and application states. For example, this application-specific code was triggered at event 37 in Figure 4, where the agent proposes adding United airlines. All of the agent’s other communications in the example session were the result of the default application-independent agent implementation.

4.2. THE BASIC EXECUTION CYCLE

The best way to understand the basic execution cycle of the architecture in Figure 8 is to start with the arrival of a communication or observation event (from either the agent or the user) at the discourse interpretation module at the top center of the diagram. The interpretation module updates the discourse state as described in Section 3.4 above, which then causes a new agenda of expected communication and manipulation acts (by either the user or agent) to be computed by the discourse generation module. As mentioned above, the agent may decide to select an entry in this new agenda for execution.

A subset of the agenda is also presented to the user whenever the user opens the communication menu in his home window. Specifically, the user communication menu is constructed by selecting all the communication actions in the agenda for which the actor is either unspecified or itself. 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.

If the user selects one of the communication menu entries, it becomes input to the discourse interpretation module, thus closing an execution cycle.

4.3. SEGMENTED INTERACTION HISTORY

In addition to the user communication menu, the second form of human-readable output produced by the Collagen discourse manager is the *segmented interaction history*, which appears in a pop-up window (see Figure 5) whenever the user presses the “History” button in her home window. The most basic function of the segmented interaction history is to provide the user with a structured guide to her problem solving process. It also serves as a menu for history-based transformations, which are discussed in the next section.

The segmented interaction history is produced from the current discourse state by the following three steps:

```

(Stopped jointly doing act1 via recipe1)
(Done agent doing act5 via recipe5)
(User doing act6 on value1 via recipe6)
  USER: "Parameter1 of act6 is value1."
  USER: Act7.
  (User doing act8)
  USER: "Propose I do act8."
  (Expect step3)

```

Figure 9. Segmented interaction history generated from discourse state in Figure 7.

1. List the purpose of each toplevel segment on the history list started with the oldest.
2. Recursively, starting with the toplevel open segment (the base segment of the focus stack), list the purpose of each closed subsegment followed by the purpose and elements of each open subsegment, indenting at each recursion.
3. For each open subsegment, starting with the most deeply nested, list the unexecuted recipe steps, if any, in the plan tree for that segment's purpose, outdenting at each level.*

An example of the result of applying these steps to the discourse state in Figure 7 is shown in Figure 9 with the current segment selected.

Notice above and in Figure 4 that the purpose of an open segment is glossed with a present participle, such as "doing." Closed segments are glossed starting with "done" or "stopped." Remaining unexecuted steps of a recipe are glossed starting with "expect."

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.

Users can also exploit the structure of the interaction history to control the level of detail that is presented. The presentation of a segment can alternate between just its purpose or all of its elements by single and double clicking similar to the way that hierarchical file structures are expanded and contracted in graphical file inspectors. For example, Figure 10a shows the same discourse state as above in which the oldest toplevel segment has been expanded to two levels of detail, while all the other segments have been contracted to just their purpose.

* Because it contains both historical and future information, this display might more accurately be called the interaction "context." However, we have kept the name "history" because it is more suggestive.

5. History-Based Transformations

Making the interaction history an explicit, manipulable object, and the fact that it is structured according to the user's intentions, opens the possibility for powerful transformations on the state of the problem solving process. In this section, we describe three basic categories of such transformations, which we call *stopping*, *returning*, and *replay*. The framework in which to understand these transformations is in terms of their different effects on the application state and the discourse state.

The details of the application state representation depend, of course, on the application. For the purpose of this discussion, we assume that the application provides some method for reestablishing any earlier state, neglecting the important engineering tradeoffs between copying the entire state of an application at various "checkpoints" versus keeping enough information to reconstruct intermediate states by undoing or replaying actions. (If checkpointing is expensive, segment boundaries suggest good places at which to do so.)

In all of the transformations below, the elements of closed segments are never modified. A copy of the focus stack and the plan tree are stored at the start and end of each segment. (Because the elements of a segment are acts and subsegments, the start of a segment does not always correspond to the end of another segment.)

The basic unit to which history-based transformations are applied is the segment. Requests to apply transformations applicable to the current segment (the top of the stack) always appear at the end of the user's communication menu, after the entries derived from the current discourse manager agenda. For example, at the moment in the opening scenario shown in Figure 2, the purpose of the current segment is to propose (the recipe for) how the trip will be scheduled. Therefore, as seen in the figure, the following three lines appear at the bottom of the user communication menu:

Stop proposing how a trip on the route be scheduled.
Retry proposing how a trip on the route be scheduled.
Undo proposing how a trip on the route be scheduled.

To apply transformations to other segments, the user pops up the interaction history window and then selects the desired segment by clicking on it. Applicable transformation requests for the selected segment are then automatically added to the communication menu.

5.1. STOPPING

The simplest history-based transformation is to pop the current segment off the focus stack without changing the application state. Furthermore, if the purpose of the popped segment contributes to its parent, the appropriate

unbindings are also performed in the plan tree. The stop transformation is applicable only to open segments.

The user may employ this transformation to let the agent know that, even though the current goal has not been achieved, she is no longer working towards it. It may also be useful when the agent has misunderstood what the current goal is. Stopping is a component of some of the more complicated transformations described below.

5.2. RETURNING

Returns are a category of transformation in which both the application and discourse states are reset to an earlier point in the problem solving process. There are three forms of return, which we call *retry*, *revisit*, and *undo*. In all three forms, the application state is reset to the state at the start (retry and undo) or end (revisit) of the target segment.

5.2.1. *Retry and Revisit*

Intuitively, retry is the transformation to use when you want to return to working on an earlier goal—achieved or not—and try achieving it a different way. Retry is applicable to any segment.

Revisit is the transformation to use when you want to pick up where you left off working on an earlier goal, especially one that was stopped. Revisit is applicable only to closed segments, since all open segments are currently being worked on.

To illustrate retry and revisit, Figure 10a shows the history corresponding to the abstract discourse in Figure 7, with the segment to be returned to selected. Figures 10b and 10c show the interaction histories after a retry or revisit transformation has been applied. Notice that in both cases, there are two segments on the stack after the return.* Notice also that in a revisit transformation the recipe is preserved, whereas in a retry the recipe becomes unbound.

In general, resetting the discourse state for a retry or revisit involves an appropriate stop followed by resetting the stack and plan tree to their states at either the start (retry) or end (revisit) of the selected segment. If the segment being returned to (e.g., act2 in Figure 10) is, or its parent is, on the history list, then the appropriate segment to stop is the segment at the base of the stack (e.g., act6), thereby emptying the stack. Otherwise, the appropriate segment to stop is the open sibling segment of the segment being returned to, if any.

* Act1, the purpose of the parent of the segment being returned to, is glossed as “returning to” rather than “retrying” or “revisiting,” because, in general, we could be returning to the middle of it.

(a) Before return:

(Stopped jointly doing act1 via recipe1)

(Stopped jointly doing act2 via recipe2)
 AGENT: Act3.
 USER: Act4.

(Done agent doing act5 via recipe5)
 (User doing act6 on value1 via recipe6)

(b) Retry (return to start of) segment:

(Stopped jointly doing act1 via recipe1)
 (Done agent doing act5 via recipe5)
 (Stopped user doing act6 on value1 via recipe6)
 (Returning to jointly doing act1 via recipe1)
 (Retrying jointly doing act2)
 USER: "Retry jointly doing act2."

(c) Revisit (return to end of) segment:

(Stopped jointly doing act1 via recipe1)
 (Done agent doing act5 via recipe5)
 (Stopped user doing act6 on value1 via recipe6)
 (Returning to jointly doing act1 via recipe1)
 (Revisiting jointly doing act2 via recipe2)
 USER: "Revisit jointly doing act2 via recipe2."
 AGENT: Act3.
 USER: Act4.

Figure 10. Examples of returning.

5.2.2. *Undo*

Undo is the familiar transformation in which you want to pretend that you never even started working on a goal. Undo is applicable only to open segments, or if the stack is empty, the most recent segment in the history list or any of its terminating subsegments. For example, undoing act6 in the initial state of Figures 7 and 10a would yield an empty stack and the following history:

(Stopped jointly doing act1 via recipe1)
 (Done agent doing act5 via recipe5)
 (Undone user doing act6 on value1 via recipe6)

Resetting the discourse state to undo an open segment involves the same steps as stopping that segment. The only difference is that with undo the application state is also reset. Undoing the last (or terminating) segment on the history list (when the stack is empty) requires only unbinding that segment's purpose from its parent in the plan tree.

5.3. REPLAY

Replay is a transformation which allows the user to reuse earlier work in a slightly different, i.e., the current, context. The basic idea is that all of the

(Jointly scheduling a trip on the route via working forward, allowing 26 itineraries)
 (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) USER: Add Dallas stopover with arrival ... departure ... USER: Change Dallas stopover to arrival ... departure ... USER: Add Boston arrival ... USER: Change Boston to arrival ...

(Done jointly specifying airlines, allowing 10 itineraries)
 (Done user displaying itineraries)

(Retried user identifying route of scheduling a trip as Oakland to Dallas to Boston, allowing 100+ itineraries)
 USER: "Retry user identifying route of scheduling a trip."
 USER: Add Oakland to the route.
 USER: Add Dallas to the route, allowing 87 itineraries.
 USER: Add Boston to the route, allowing 100+ itineraries.

(Done user working on Oakland to Dallas leg, allowing 93 itineraries)
 (Replayed working on Dallas to Boston leg, allowing 8 itineraries)
 USER: "Replay user working on Dallas to Boston leg."
 AGENT: Add Dallas stopover with arrival ... departure ...
 AGENT: Change Dallas stopover to arrival ... departure ...
 AGENT: Add Boston arrival ...
 AGENT: Change Boston to arrival ...

(Done user displaying itineraries)
 (Revisiting jointly specifying airlines, allowing 26 itineraries)
 USER: "Revisit jointly specifying airlines."
 USER: Add American specification, allowing no itineraries.
 (Done agent adding United specification, allowing 10 itineraries)
 AGENT: "Propose I add United specification."
 USER: "Ok."
 AGENT: Add United specification, allowing 10 itineraries.
 USER: Add USAir specification, allowing 26 itineraries.

Figure 11. Transformations in test application.

application acts in the selected segment are put together into one (possibly hierarchical) “recipe,” which is then executed by the agent in the current context.

When executing such a replay recipe, it is important for the agent to be prepared for the possibility that some of the acts, e.g., adding an airline that has already been specified, may not be valid in the current context. Depending on the specific details of the agent’s interface to the application, such errors may need to be handled by application-specific code in the agent, or may be taken care of by the application’s existing API or graphical interface.

Figure 11 is example of how replay can be used, together with returns, in the test application. Notice that the scenario in this figure is a shorter trip (only two legs) than the scenario in Section 2. The levels of expansion of the history have been set to show only the details of interest. The segment that

is going to be replayed is shown selected and underlining has been added to highlight the three transformation segments.

To motivate this example, suppose that after displaying itineraries (see the blank line in the middle of Figure 11), the user got the idea of trying to leave from the nearby Oakland airport instead of San Francisco. In order to pursue this alternative, she returned to (retried) the first subsegment in the history, this time entering the route Oakland-Dallas-Boston on the map in the application window. Notice that the application state at the end of this retried segment did not include any city arrival/departure constraints.

Next, the user constrained her departure time from Oakland (“Done user working on Oakland to Dallas leg” in the history). Then, instead of manually (re-)entering the arrival/departure constraints for Dallas and Boston, she requested replay of the selected segment.

After displaying and reviewing the possible itineraries starting in Oakland, however, the user decided to return to working on the San Francisco route after all. In particular, at the end of Figure 11, she is revisiting the earlier airline specification segment (fifth subsegment down from the top) in order to see what happens if she adds USAir to the specified airlines.

6. 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 data base or “domain modelling” in artificial intelligence (Brodie et al., 1982). It also overlaps with modern specification practices in software engineering, although the goals and recipes in a collaborative discourse model 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 our methodology. 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,” so to speak—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 need not rely on the task model alone for its decision making, 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 to the recipe library.

6.1. 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).

Below are examples of how some of the communications in our example scenario 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.

$PFA(37, agent, SHOULD(add-airline(t, agent, ua)), user)$
 AGENT: "Propose I add United specification."

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

$PFA(1, user, SHOULD(schedule(t, who, route)), agent)$
 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(9, agent,
    SHOULD(PFA( $t_1$ , user,
                RECIPE(schedule( $t_2$ , who, route(bos, dfw, den, sfo, bos)),
                        recipe),
                agent)),
    user)
AGENT: "How should a trip on the route be scheduled?"

```

Notice that the glossing algorithm has some limited ability to introduce definite references, such as “the route” above, based on the focus stack and some application-specific heuristics.

6.2. 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 test 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 test 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 test application when the user moves the same interval bar back and forth several times in a row.

7. Local and Global Initiative in Conversation

Most work on initiative in conversation, e.g., (Clark and Schaeffer, 1989; Walker and Whittaker, 1990; Traum and Hinkelman, 1992), has focused on problems that are *local* to a discourse segment. Phenomena in this realm include:

- turn taking and conversational control (who gets to speak next),
- interruptions, and
- grounding (how the current speaker indicates that she has heard and understood the content of the previous speaker’s turn).

Other work, e.g., (Traum and Hinkelman, 1992; Green, 1994; Guinn, 1996; Allen et al., 1996), has looked at initiative more globally in terms of “having something to say.” This work tends to focus on the conversants’ problem-solving level and on choosing appropriate speech acts or discourse plan operators.

In our view, an approach to the so-called “mixed initiative” issue in interactive systems must address both levels of phenomena: the global level of having something to say that is relevant to what has recently been said or done, and the local level, concerning when and how a participant gets the opportunity to speak.

7.1. GLOBAL INITIATIVE

The collaborative discourse theory upon which Collagen is based provides strong support for dealing with the global constraints on initiative. The maintenance of a discourse state representation and the agenda which is computed from it by the discourse manager provides a Collagen-based agent with an explicit choice of relevant things to say at most points in a conversation. Given this architecture, there is no need for a separate collection of discourse plan operators about “conversational moves” that compel the agent to answer questions or perform actions requested of it (Chu-Carroll and Brown, 1998). The agent’s cooperative behavior results directly from the overall model of collaboration.

As discussed in Section 4.1, the Collagen architecture leaves it up to the agent to decide between answering a question, performing an interface action or choosing some other behavior because that is appropriately a function of application-specific planning rather than discourse processing.

Recent research, e.g., (Guinn, 1994; Chu-Carroll and Carberry, 1994; Chu-Carroll and Carberry, 1995), has also begun to consider negotiation as part of global initiative. By negotiation, we mean the ability to resolve differences in beliefs that are relevant to some shared goal. Negotiation is fundamental to collaboration because collaborators often have differing points of view about the goals and recipes they undertake as well as the state of

the world at any point in time. Collagen’s artificial language, in particular the full version described in (Sidner, 1994), is a start toward pursuing this idea. However there is much more work to be done before Collagen can incorporate a general negotiation facility.

7.2. LOCAL INITIATIVE

Because current theories of local initiative in conversation do not yet provide general-enough algorithms for turn taking, control and grounding, Collagen does not provide any support in these areas. We have, however, experimented in our test agent for air travel planning with several ad hoc mechanisms for local initiative, which we describe below.

One of the most basic local initiative concerns we needed to address in our agent implementation was how the user relinquishes control to the agent. For example, this needs to happen when the user decides she does not want to contribute any further to the current SharedPlan and instead would like to see what the agent can contribute. To support this specific behavior, we designed the agent to interpret an “ok” said by the user (other than in answer to a direct yes-no question) as an signal of relinquishing control.*

A second local initiative mechanism we built into the agent was a way to get the user’s attention when the agent does not have control. Since it was difficult and awkward to physically grab the user’s cursor in the middle of use, we gave the agent the ability to wave its cursor “hand” when it has something important to contribute to the conversation.** The resulting behavior is very humanlike and affecting. However, some observers of our interface have urged us to just have the agent “barge into” the user’s interaction with the application interface.

Finally, we agree with Walker and Whittaker’s (1990) observation that many discourse initiative phenomena that appear to be local are in fact dependent upon global constraints. More careful attention to global models may result in theories that better explain both levels of mixed initiative.

8. Comparison with Other 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.

Most conventional work on user interface concentrates on optimizing the appearance and functionality of a single interaction or a short sequence of interactions. In contrast, our work is about supporting a user’s problem

* We also experimented with an alternate signal of having the user return her cursor to the user home window for a period of two or three seconds, but found this was awkward.

** We chose not to use computer keyboards sounds, such as bells, because they introduced new medium for which we had no general framework.

solving process by relating current actions to the global context and history of the interaction.

Our concept of a collaborative interface agent is closest to the work of Maes (1994), although she uses the term “collaborative” to refer to the sharing of information between multiple software agents, rather than collaboration between the agent and the user. Cohen et al. (1994) has also developed interface agents without collaborative discourse modelling. Terveen (1991) has explored providing intelligent assistance through collaborative graphical manipulation without explicit invoking the agent paradigm.

At a more abstract level, recent work on mixed-initiative systems by Guinn (1998) and Cohen et al. (1998) treat collaboration as an extension of single agent problem solving in which some designated participant has the “task initiative” for each mutual goal. Cohen et al., for example, conclude that task initiative depends on who is proposing a goal. In contrast, our work is based on an underlying formalization (Grosz and Kraus, 1996) in which collaboration is fundamentally a property of groups. Although this distinction is not highlighted in our current scenarios, it will become important in future extensions.

Cohen (1992) and Jacob (1995), among others, have explored discourse-related extensions to direct manipulation that incorporate anaphora and make previous context directly available. However, most work on applying human discourse principles to human-computer interaction, e.g., (Lambert and Carberry, 1991; Yanklovich, 1994), have assumed that natural language understanding will be applied to the user’s utterances

In Moore et al.’s work (Lemaire and Moore, 1994; Moore and Swartout, 1990), which focuses on explanation dialogues, users are presented with a full textual history of their interaction with the system, from which they may select any phrase as the context for a further query. Unlike our approach, Moore’s history display has no explicit structure other than the alternation of user and system utterances. Internally, however, Moore’s work does use a deep representation of the user’s and system’s goals.

The basic idea underlying Collagen’s user communication menu, namely replacing natural language understanding by natural language generation based on the expectations of context, has also been used by Fischer (1994) for cooperative information retrieval and by Mittal and Moore (1995) for clarification subdialogues.

The three systems we know of that are overall closest in spirit to our own are Stein and Maier’s MERIT (1995), subsequent work on MIRACLE by Stein, Gulla and Thiel (1998), and Ahn et al.’s DenK (1995). MERIT and MIRACLE use a different discourse theory, which is compiled into a less flexible and less extensible finite-state machine representation. Neither of these systems deal with actions that directly manipulate a graphical inter-

face. DenK has the goal of providing a discourse-based agent, but has not yet modelled collaboration.

9. Conclusions

In summary, applying the software agent paradigm and collaborative discourse theory to human-computer interaction in graphical user interfaces has posed a number of challenges, including:

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

We believe the current work has made a strong start toward these goals and provides a new conceptual platform upon which we and others can now build higher.

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 and handling parallel interleaved goals,
- 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.

A major effort which has occupied much of the past year has been to reimplement the Collagen in Java. This effort is not yet complete at the time of this writing. The Java implementation should greatly facilitate experimentation by other interested researchers, which we invite. Please visit our project home page at <http://www.merl.com/projects/collagen> for up-to-date information.

References

- Ahn et al., R.: 1995, 'The DenK-architecture: A Fundamental Approach to User-Interfaces'. *Artificial Intelligence Review* **8**, 431–445.
- Allen et al., J. F.: 1996, 'A Robust System for Natural Spoken Dialogue'. In: *Proc. 34th Annual Meeting of the ACL*. pp. 62–70.
- Bratman, M. E., D. J. Israel, and M. E. Pollack: 1988, 'Plans and Resource-Bounded Practical Reasoning'. *Computational Intelligence* **4**(4), 349–355.
- Brodie, M., J. Mylopoulos, and J. Schmidt (eds.): 1982, *On Conceptual Modelling*. New York, NY: Springer-Verlag.

- Chu-Carroll, J. and M. Brown: 1998, 'An Evidential Model for Tracking Initiative in Collaborative Dialogue Interactions'. *User Modeling and User-Adapted Interaction*. In this issue.
- Chu-Carroll, J. and S. Carberry: 1994, 'A Plan-Based Model for Response Generation in Collaborative Task-Oriented Dialogues'. In: *Proc. 12th National Conf. on Artificial Intelligence*. Seattle, WA, pp. 799–805.
- Chu-Carroll, J. and S. Carberry: 1995, 'Response Generation in Collaborative Negotiation'. In: *Proc. 33rd Annual Meeting of the ACL*. Cambridge, MA, pp. 136–143.
- Clark, H. H. and E. F. Schaeffer: 1989, 'Contributing to Discourse'. *Cognitive Science* **13**(2), 259–294.
- Cohen, P.: 1992, 'The Role of Natural Language in a Multimodal Interface'. In: *Proc. 5th ACM Symp. on User Interface Software and Technology*. Monterey, CA, pp. 143–149.
- Cohen et al., P.: 1994, 'An Open Agent Architecture'. In: O. Etzioni (ed.): *Software Agents, Papers from the 1994 Spring Symposium, SS-94-03*. Menlo Park, CA: AAAI Press, pp. 1–8.
- Cohen et al., R.: 1998, 'What is Initiative?'. *User Modeling and User-Adapted Interaction*. In this issue.
- Fischer, M., E. Maier, and A. Stein: 1994, 'Generating Cooperative System Responses in Information Retrieval Dialogues'. In: *Proc. 7th Int. Workshop Natural Language Generation*. Kennebunkport, ME, pp. 207–216.
- Green, N. L.: 1994, 'A Computational Model for Generating and Interpreting Indirect Answers'. Ph.D. thesis, Univ. of Delaware, Dept. of Computer and Info. Sci.
- Grosz, B. J. and S. Kraus: 1996, 'Collaborative Plans for Complex Group Action'. *Artificial Intelligence* **86**(2), 269–357.
- Grosz, B. J. and C. L. Sidner: 1986, 'Attention, Intentions, and the Structure of Discourse'. *Computational Linguistics* **12**(3), 175–204.
- Grosz, B. J. and C. L. Sidner: 1990, 'Plans for Discourse'. In: P. R. Cohen, J. L. Morgan, and M. E. Pollack (eds.): *Intentions and Communication*. Cambridge, MA: MIT Press, Chapt. 20, pp. 417–444.
- Grosz [Deutsch], B. J.: 1974, 'The Structure of Task-Oriented Dialogs'. In: *IEEE Symp. on Speech Recognition: Contributed Papers*. Pittsburgh, PA, pp. 250–253.
- Guinn, C. I.: 1994, 'Meta-Dialogue Behaviors: Improving the Efficiency of Human-Machine Dialogue—A Computational Model of Variable Initiative and Negotiation in Collaborative Problem-Solving, Communication and Miscommunication'. Ph.D. thesis, Duke University.
- Guinn, C. I.: 1996, 'Mechanisms for Mixed-Initiative Human-Computer Collaborative Discourse'. In: *Proc. 34th Annual Meeting of the ACL*. pp. 278–285.
- Guinn, C. I.: 1998, 'Principles of Mixed-Initiative Human-Computer Collaborative Discourse'. *User Modeling and User-Adapted Interaction*. In this issue.
- Jacob, R. J. K.: 1995, 'Natural Dialogue in Modes other than Natural Language'. In: R.-J. Beun, M. Baker, and M. Reiner (eds.): *Dialogue and Instruction*. Berlin: Springer-Verlag, pp. 289–301.
- Kautz, H.: 1990, 'A Circumscriptive Theory of Plan Recognition'. In: P. R. Cohen, J. L. Morgan, and M. E. Pollack (eds.): *Intentions and Communication*. Cambridge, MA: MIT Press, Chapt. 6, pp. 105–133.
- Kowtko, J. C. and P. Price: 1989, 'Data Collection and Analysis in the Air Travel Planning Domain'. In: *DARPA Workshop Proc.* Cape Cod, MA.
- Lambert, L. and S. Carberry: 1991, 'A Tripartite Plan-Based Model of Dialogue'. In: *Proc. 29th Annual Meeting of the ACL*. Berkeley, CA.
- Lemaire, B. and J. Moore: 1994, 'An Improved Interface for Tutorial Dialogues: Browsing a Visual Dialogue History'. In: *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems*. Boston, MA, pp. 16–22.
- Lochbaum, K. E.: 1994, 'Using Collaborative Plans to Model the Intentional Structure of Discourse'. Technical Report 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'. In: *Proc. 14th Int. Joint Conf. Artificial Intelligence*. Montreal, Canada, pp. 1260–1266.
- Lochbaum, K. E.: 1998, 'A Collaborative Planning Model of Intentional Structure'. *Computational Linguistics*. Forthcoming.
- Maes, P.: 1994, 'Agents that Reduce Work and Information Overload'. *Comm. ACM* **37**(17), 30–40. Special Issue on Intelligent Agents.
- Meyers et al., B.: 1990, 'Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces'. *IEEE Computer* **23**(11), 71–85.
- Mittal, V. and J. Moore: 1995, 'Dynamic Generation of Follow-up Question Menus: Facilitating Interactive Natural Language Dialogues'. In: *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems*. Denver, CO, pp. 90–97.
- Moore, J. and W. Swartout: 1990, 'Pointing: A Way Toward Explanation Dialogue'. In: *Proc. 8th National Conf. on Artificial Intelligence*. Menlo Park, CA, pp. 457–464.
- Rich, C.: 1996, 'Window Sharing with Collaborative Interface Agents'. *ACM SIGCHI Bulletin* **28**(1), 70–78.
- Rich, C. and C. Sidner: 1996, 'Adding a Collaborative Agent to Graphical User Interfaces'. In: *Proc. 9th ACM Symp. on User Interface Software and Technology*. Seattle, WA, pp. 21–30.
- Rich, C. and C. Sidner: 1997a, 'Collagen: When Agents Collaborate with People'. In: *Proc. 1st Int. Conf. on Autonomous Agents*. Marina del Rey, CA, pp. 284–291.
- Rich, C. and C. Sidner: 1997b, 'Segmented Interaction History in a Collaborative Interface Agent'. In: *Proc. Int. Conf. on Intelligent User Interfaces*. Orlando, FL, pp. 23–30.
- Sidner, C. L.: 1994, 'An Artificial Discourse Language for Collaborative Negotiation'. In: *Proc. 12th National Conf. on Artificial Intelligence*. Seattle, WA, pp. 814–819.
- Stein, A. and E. Maier: 1995, 'Structuring Collaborative Information-Seeking Dialogues'. *Knowledge-Based Systems* **8**(2-3), 82–93.
- Stein, A., E. Maier, and U. Thiel: 1998, 'User-Tailored Planning of Mixed Initiative Information Seeking Dialogues'. *User Modeling and User-Adapted Interaction*. In this issue.
- Terveen, G., D. Wroblewski, and S. Tighe: 1991, 'Intelligent Assistance through Collaborative Manipulation'. In: *Proc. 12th Int. Joint Conf. Artificial Intelligence*. Sydney, Australia, pp. 9–14.
- Traum, D. R. and E. A. Hinkelman: 1992, 'Conversation Acts in Task-Oriented Spoken Dialogue'. *Computational Intelligence* **8**(3), 575–599.
- Walker, M. A. and S. Whittaker: 1990, 'Mixed Initiative in Dialogue: An Investigation into Discourse Segmentation'. In: *Proc. 28th Annual Meeting of the ACL*. pp. 70–79.
- Yanklovich, N.: 1994, 'Talking vs. Taking: Speech Access to Remote Computers'. In: *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems*. Boston, MA, pp. 275–276.