Segmented Interaction History in a Collaborative Agent

Charles Rich, Candace L. Sidner

TR96-14   June 1996

Abstract

We have developed an application-independent toolkit, called Collagen, based on the SharedPlan theory of collaborative discourse, in which interaction histories are hierarchically structured according to a userś goals and intentions. We have used Collagen to implement an example collaborative interface agent with discourse processing, but not natural language understanding. In this paper, we concentrate on how a segmented interaction history supports user orientation, intelligent assistance, and transformations, such as returning to earlier points in the problem solving process and replaying segments in a new context. Superseded by TR97-21.

Third International Conference on Intelligent User Interfaces, Orlando, FL, January, 1997, pp. 23-30.

Segmented Interaction History in a Collaborative Interface Agent

Charles Rich   Candace L. Sidner*

TR-96-14   June 1996

Abstract

We have developed an application-independent toolkit, called Collagen, based on the SharedPlan theory of collaborative discourse, in which interaction histories are hierarchically structured according to a user's goals and intentions. We have used Collagen to implement an example collaborative interface agent with discourse processing, but not natural language understanding. In this paper, we concentrate on how a segmented interaction history supports user orientation, intelligent assistance, and transformations, such as returning to earlier points in the problem solving process and replaying segments in a new context.

To appear in Third Int. Conf. on Intelligent User Interfaces, Orlando, FL, January 1997.

*Lotus Development Corporation

Publication History:–

1. First printing, TR-96-14, June 1996

Segmented Interaction History in a Collaborative Interface Agent

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

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

ABSTRACT
We have developed an application-independent toolkit, called Collagen, based on the SharedPlan theory of collaborative discourse, in which interaction histories are hierarchically structured according to a user's goals and intentions. We have used Collagen to implement an example collaborative interface agent with discourse processing, but not natural language understanding. In this paper, we concentrate on how a segmented interaction history supports user orientation, intelligent assistance, and transformations, such as returning to earlier points in the problem solving process and replaying segments in a new context.

Keywords
Interaction history, discourse, segment, collaboration, interface agent, undo, replay.

INTRODUCTION
One of the key features that should distinguish "intelligent" user interfaces from conventional ones is better support for the intentional level of human-computer interaction, especially as it unfolds over time. 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 solving process by relating current actions to the global context and history of the interaction.

Our approach to achieving this goal has been to draw upon what is known about human collaborative discourse, specifically from the SharedPlan work of Grosz, Sidner, Kraus, and Lochbaum [3, 4, 5, 8, 9]. This work has provided us with the basic theory and algorithms necessary build a representation of interaction history in which problem-solving goals and intentions are explicit. One of our underlying assumptions is that a human-computer interface based on human discourse rules and conventions will be easier for people to learn and use than one that is not.

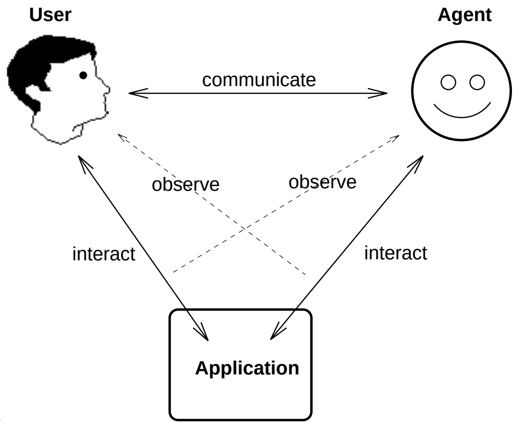
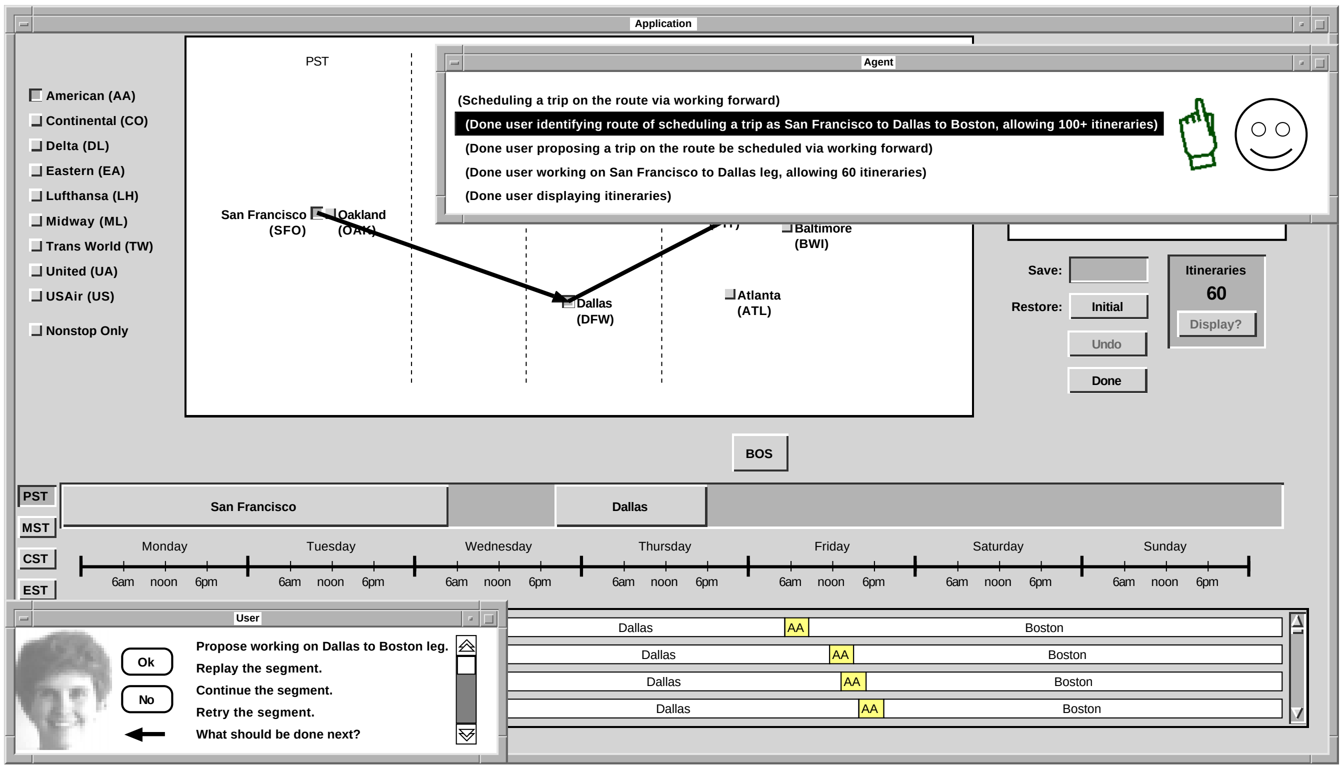


Figure 1: Collaborative interface agent paradigm.

We have implemented a application-independent toolkit for this purpose, called Collagen [14], and used it to build an complete example system [13] demonstrating how the information needed to *construct* a segmented interaction history can be acquired without unduly disrupting the flow of work. This paper concentrates on how the segmented history is *used* to support user orientation, intelligent assistance, and history-based transformations.

In the following sections, after further discussion of our general approach, we present a sample session with our demonstration system. We then describe the Collagen's application-independent discourse processing framework, following which we discuss history-based transformations, such as returning to earlier points in the problem solving process and replaying segments in a new context. All of the examples in this paper are from the demonstration system, which has been implemented in Common Lisp and runs in real time (a maximum of a few seconds per interaction).

COLLABORATIVE INTERFACE AGENT
Our current demonstration system uses the paradigm of a collaborative interface agent (see Figure 1). However, as will be clear below, the basic structures for representing and using the interaction history are not limited to this paradigm. Our concept of an interface agent is similar to the work of Maes [10], except she uses the term



(Scheduling a trip on the route via working forward)
(Done user identifying route of scheduling a trip as San Francisco to Dallas to Boston, allowing 100+ itineraries)
(Done user proposing a trip on the route be scheduled via working forward)
(Done user working on San Francisco to Dallas leg, allowing 60 itineraries)
(Done user displaying itineraries)

Propose working on Dallas to Boston leg.
Replay the segment.
Continue the segment.
Retry the segment.
What should be done next?

Figure 2: Application with overlapping user and agent home windows.

"collaborative" to refer to the sharing of information between multiple software agents.

The collaborative interface agent paradigm mimics the relationships that hold when two humans collaborate on a task involving a shared artifact, such as two mechanics working on a car engine together or two computer users working on a spreadsheet together. This implies, as shown in Figure 1, that the software agent is able to communicate with and observe the actions of the user. One of the agent's main responsibilities is to maintain the history and context of the collaboration, based partly on communication from the user about her intentions and partly on observation of the user's actions.

The agent can also interact directly with the shared application program. The agent queries the state of the application using the application's programming interface (API). The agent could also modify the state of the application through the API. However, the interface agent paradigm is more complete when the agent modifies the application state using the same graphical interface as the user.

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 it is in.

As the internal semantic (i.e., content) representation of the communication between the user and agent, we employ an artificial language developed by Sidner [15]. Simple string templates are used to translate messages from this internal representation into English sentences for the user to read.

DEMONSTRATION SYSTEM

Figure 2 shows how the architecture of Figure 1 is realized on a user's display. The large window labelled "Application" is a direct-manipulation interface to an airline schedule database and a simple constraint checker. By pressing buttons, moving sliders, and so on, the user can specify and modify the geographical, temporal, and other constraints on a planned trip. The user can also retrieve and display possible itineraries satisfying the given constraints. A typical session to schedule a complicated trip, e.g., one involving four or five cities, lasts about 15 minutes and entails about 150 actions (mouse clicks) on the application window.

The smaller overlapping windows in the upper-right and lower-left corners of the screen in Figure 2 are the agent's and user's *home windows*, through which they communicate with each other. These windows are moveable and expand and shrink as their use changes. In a typical session, application actions by the user and agent are interleaved with communication actions between them. The agent manipulates objects in the application window by pointing and clicking with the hand icon shown in its home window.

For additional discussion of issues related to adding a collaborative agent to graphical user interfaces, see [13].

SAMPLE SESSION

The sample session in this section provides:

- a further introduction to the application domain,
- an example of interactively and incrementally building a segmented interaction history, and
- examples of using the segmented history to support user orientation and intelligent assistance.

This session is only one of many possible variations in the order of actions and division of labor between the user and agent. Notice that we will use the terms "history" and "context" somewhat interchangeably below. The technical distinction between these needs to wait until the underlying discourse processing algorithms and data structures are described in the next section.

The session begins with the user being given the following problem statement:

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

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. Clicking on the arrow at the bottom of her home window causes the window to expand showing her current communication choices:

Propose scheduling a trip.
What have we been doing?

There is only one possible collaboration to propose here, since the demonstration system 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. Note that at this point the agent has only generic knowledge of the typical tasks involved in scheduling a trip and recipes (general methods) for performing them. It does does not know anything about the user's particular problem or preferences.

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

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

Notice that the interaction history above is hierarchically structured. Each *segment* in the hierarchy has a *purpose*, which is described in parentheses at the start of the segment. The elements of a segment are either primitive (communication or application) actions or subsegments. Communication actions are indicated by quotation marks. In general, the purpose of each subsegment contributes to its parent segment's purpose.

The most basic function of the segmented interaction history is to orient the user. For example, if the user left her computer in the middle of working on a problem and returned after a few hours (or days), the history would help her reestablish where in the problem solving process she was.

The user now clicks on "Ok" in her home window to signal the end of the current segment. The agent now takes the initiative and asks:

How should a trip on the route be scheduled?

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

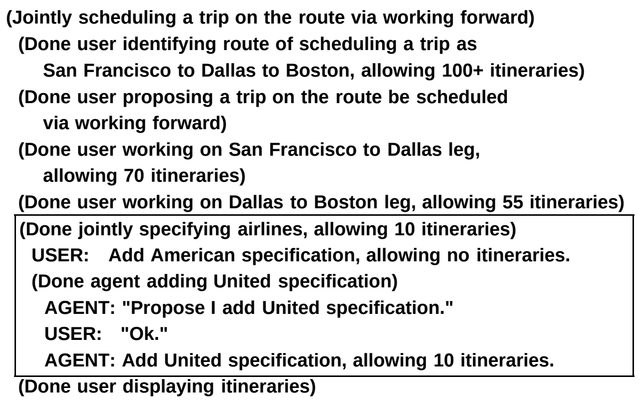
```
Propose scheduling a trip on the route via ___ .
                                  working forward
                                  working backward
```

The agent knows about two recipes for scheduling a trip: 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 forward, after which the agent says:

Propose you work on San Francisco to Dallas leg.

Here again the agent uses the current context to assist the user, in this case to suggest the next subtask. Working on a leg entails manipulating the "interval bars" in the horizontal slider area below the map (see Figure 2) to specify latest arrival and earliest departure times at a city.

We skip ahead now to the end of the sample session, where the interaction history is as shown in Figure 3. Notice the varying level of detail in this history: the selected segment, in which the agent and user jointly specified airlines, is shown in full, while the internal elements of the other segments are suppressed. The user



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

Figure 3: History at end of sample session.

can interactively control the level of detail in interaction histories by single and double clicking on segments similarly to the way hierarchial file structures are typically inspected.

The airline specification segment in Figure 3 is expanded in full to show an example of the agent performing an application act, in addition to just communicating with the user. The user began this segment by requiring all flights use American airlines, which resulted in no possible itineraries. Whenever a constraint (such as an airline specification) is entered or changed, the application program automatically recomputes the number of possible itineraries and displays this number in the box labelled "Itineraries." In the segmented interaction history, the number of possible itineraries after each segment or action is indicated if it is different from the number before.

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

Given the interaction above, it is tempting to think of what the agent is doing as plan recognition, which is known to be exponential in the worst case [6]. 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. As we will see in the next section, the only search performed by the agent's discourse processing 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.

DISCOURSE PROCESSING IN COLLAGEN

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

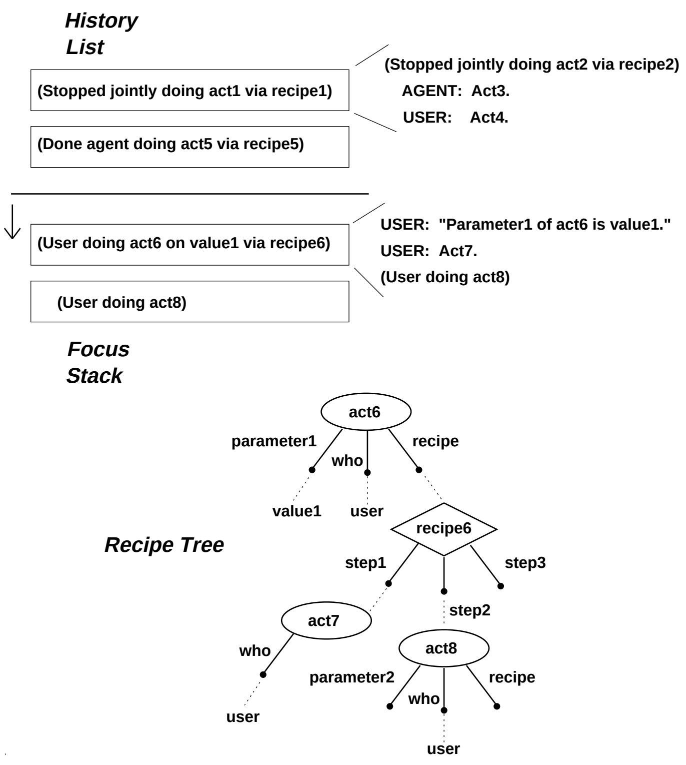


Figure 4: Internal discourse state representation.

Discourse processing in Grosz and Sidner's SharedPlan framework [4, 5] has the three interrelated components, represented internally as shown in Figure 4:

- The linguistic structure of a discourse includes the hierarchical grouping of acts into *segments*.
- The attentional state of a discourse captures the shifting focus of attention of the participants. We represent attentional context as a *focus stack*[1] of discourse segments (shown in the figure as growing downward) plus a *history list*, which contains toplevel segments that have been popped off the stack.
- The intentional structure of a discourse, corresponding to the current partial status of the participants' SharedPlans, is represented as a *recipe tree*, which is a reimplementation of Lochbaum's rgraph [8].

The main job of discourse processing is to explain how the current communication or application act contributes 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 the recipe library),
- identifies the recipe to be used to achieve the current purpose,

[1]The focus stack also constrains the use of definite references, such as "the route."

- 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.

The last three cases above are instances of a larger class of explanations that Lochbaum [9] calls "knowledge preconditions."

It is important to note that these discourse processing algorithms and data structure are application-independent. The only application-specific information used is the *recipe library*, which contains an abstract specification of each act type and its parameters, plus recipes for implementing the non-primitive acts. The recipe library for the current demonstration contains 15 act types and 8 recipes. It is probably about half the size it needs to be to reasonably cover the application domain.

Segments

In the example discourse state in Figure 4, there are two segments on the focus stack and two segments in the history list. The elements of two of these segments are shown expanded to the right. When a segment is popped off the stack, it is added to the history list if and only if it has no parent segment.

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.

A *segmented interaction history* is thus a printout of the history list and the focus stack, in which each segment is described by an English gloss of its purpose. Open segments are glossed with a present participle, such as "doing;" closed segments are glossed starting with "done" or "stopped." For example, Figure 5a is a printout of the history list and stack in Figure 4.

Recipe Tree

The recipe tree is composed of alternating act and recipe nodes, as illustrated in Figure 4. Both acts and recipes have *bindings*, shown as 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, bindings and the propagation of logical information in the recipe tree are implemented using a truth-maintenance system.

For example, in Figure 4, 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.

In general, the root of the recipe tree is the purpose of the base segment of the focus stack, and each subsegment 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 recipe tree for that segment. Recipe trees remain associated with segments even after they are popped off the stack.

The recipe tree is not (currently) presented graphically to the user. However, it underlies much of the intelligent assistance that the agent provides.

There are many other issues having to do with interactively and incrementally building segmented interaction histories which are not addressed in this section, such as dealing with unexpected user actions and allowing negotiation about proposals. We have tentative solutions in some of these areas; others are the topic of current research. In the remainder of this paper, however, we will concentrate on how the history is used once it is built.

INTELLIGENT ASSISTANCE

One of the most basic types of intelligent assistance is suggesting what to do next. The agent provides this assistance essentially by running the discourse interpretation algorithm described above "backwards" to produce an *agenda* of (possibly partially specified) expected acts, i.e., acts which would cause the binding of the current discourse purpose's unbound parameters, etc.

Priorities are assigned to acts in the agenda based on some simple application-independent heuristics. For example, the highest priority is assigned to steps of a recipe, all of whose parameters have been bound and all of whose predecessors in the recipe have been executed. The agent often proposes the highest priority act from this agenda.

For example, the agent's question near the beginning of the sample session, "How should a trip on the route be scheduled?", is the gloss of a statement in Sidner's artificial language proposing that the user perform the communication act of choosing a recipe for the current purpose. Later in the session, when the agent says, "Propose you work on San Francisco to Dallas leg," it is proposing that the user perform the first step in the current recipe.

In addition to this application-independent algorithm, the discourse state representation and the agenda of expected acts generated from it are also useful resources for writing application-specific agent code. For example, in the air travel domain, knowing whether the user is working forward or backward affects the order in which constraints are removed by the agent in an over-constrained situation.

Finally, the context-dependent communication menu in the user's home window is itself a kind of intelligent assistance derived from the discourse agenda. Except for a few special entries, such as "What have we been doing?" and the transformation requests described in the next section, the choices in the user communication menu are simply the subset of the discourse agenda which are communication acts that may be performed by the user.

HISTORY-BASED TRANSFORMATIONS

Making the interaction history an explicit, manipulable object, and the fact that it is structured according to the user's intentions, presents 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 application state representation depend, of course, on the application. For the purpose of this discussion, we assume 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.)

The discourse state representation was described in detail in the preceding section. Of the three components of discourse state, only the focus stack and recipe tree are changed by the transformations below. The elements of closed segments are never modified. A copy of the stack and recipe 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, e.g,

Stop user working on Dallas to Boston leg.
Undo user working on Dallas to Boston leg.

To apply transformations to other segments, the user requests display of the interaction history ("What have we been doing?") and then selects the appropriate segment in the history display. Applicable transformation requests for the selected segment are then automatically added to the communication menu. The segmented interaction history serves a kind of orientation function here for finding the argument to a history-based transformation.

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 recipe 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.

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.

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 5a shows the history corresponding to Figure 4, with the segment to be

(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)
  USER:  "Parameter1 of act6 is value1."
  USER:  Act7.
  (User doing act8)
```

(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 5: Examples of returning.

returned to selected. Figures 5b and 5c 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.[2] 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 recipe 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 5) 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.

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 4 and 5a 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 recipe tree.

Replay
Replay is a transformation which allows you to reuse earlier work in a slightly different, i.e., the current, context. The basic idea is that all of the 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 6 is example of how replay can be used, together with returns, in a realistic scenario from the example

```
(Jointly scheduling a trip on the route via working forward)
 (Done user identifying route of scheduling a trip as
    San Francisco to Dallas to Boston, allowing 100+ itineraries)
 (Done user proposing a trip on the route be scheduled
    via working forward)
 (Done user working on San Francisco to Dallas leg,
    allowing 70 itineraries)
 (Done user working on Dallas to Boston leg, allowing 55 itineraries)
   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)
   USER:   "Revisit jointly specifying airlines."
   USER:   Add American specification, allowing no itineraries.
   (Done agent adding United specification)
     AGENT: "Propose I add United specification."
     USER:   "Ok."
     AGENT: Add United specification, allowing 10 itineraries.
   USER:   Add USAir specification, allowing 26 itineraries.
```

Figure 6: Transformations in example application.

application domain continuing on from the end of Figure 3. Recall that this is a textual history of interactive events, such as mouse clicks and display updates, in the application and home windows. Parts of the history have been expanded to show 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.

After displaying itineraries at the end of the sample session (see gap in Figure 6), 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.

[2] 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.

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 6, 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.

RELATED WORK

Cohen et al. [1] first demonstrated the use of an explicit, structured interaction history to allow users to locate and return to earlier contexts. The structure of Cohen's histories, however, only captured the relationship between questions and follow-up questions in a query interface. By selecting a node in the question tree, users could follow up on any earlier question, not just the most recent one. Our history representation, by comparison, captures the relationship between both actions and communications (including questions) and the goals to which they contribute, making possible much more powerful intelligent assistance and transformations.

In Moore et al.'s work [7, 12], 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 [2] for cooperative information retrieval and by Mittal and Moore [11] for clarification subdialogues.

Finally, in terms of overall goal, the most closely related work is Stein et al.'s MERIT system [16], which also strives to systematically apply human discourse theory (albeit a different one) to human-computer interaction. However, rather than explicitly representing the linguistic, attentional, and intentional structures underlying collaboration as in Collagen, MERIT relies on interaction scripts. Even though they are nondeterministic, these scripts lack the flexibility and richness of structure which Collagen provides and which supports the functions of a collaborative agent. For example, although the hierarchical dialogue history in the MERIT's CORINNA interface looks similar to a segmented interaction history, there is no representation in MERIT of the purpose of each segment or how it contributes to its parent.

CONCLUSION

We have demonstrated an application-independent approach to representing and using a segmented interaction history to support user orientation, intelligent assistance, and history-based transformations. This work is part of a broader agenda to apply principles of human collaboration and discourse to improve human-computer interaction.

Our future plans include:

- improving the flexibility and robustness of Collagen's discourse processing algorithms, especially as related to incompleteness of the agent's recipe library,
- a pilot user study to compare using the example application with and without the interface agent,
- supporting negotiation between the user and agent,
- and extending the interaction history representation to include higher-level, application-independent structures, such "making a comparison," with associated history-based transformations.

REFERENCES

1. P. Cohen et al. Synergistic use of direct manipulation and natural language. *CHI'89*, pp. 227–233.
2. M. Fischer, E. Maier, and A. Stein. Generating cooperative system responses in information retrieval dialogues. *INLGW'94*, pp. 207–216.
3. B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 1996.
4. B. J. Grosz and C. L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.
5. B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions and Communication*, chapter 20, pages 417–444. MIT Press, Cambridge, MA, 1990.
6. H. Kautz. A circumscriptive theory of plan recognition. In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions and Communication*, chapter 6, pages 105–133. MIT Press, Cambridge, MA, 1990.
7. B. Lemaire and J. Moore. An improved interface for tutorial dialogues: Browsing a visual dialogue history. *CHI'94*, pp. 16–22.
8. K. E. Lochbaum. Using collaborative plans to model the intentional structure of discourse. TR 25-94, Harvard U., Ctr. for Res. in Computing Tech., 1994. PhD thesis.
9. K. E. Lochbaum. The use of knowledge preconditions in language processing. *IJCAI'95*, pp. 1260–1266.
10. P. Maes. Agents that reduce work and information overload. *Comm. ACM*, 37(17):30–40, July 1994.
11. V. Mittal and J. Moore. Dynamic generation of follow-up question menus: Facilitating interactive natural language dialogues. *CHI'95*, pp. 90–97.
12. J. Moore and W. Swartout. Pointing: A way toward explanation dialogue. *AAAI'90*, pp. 457–464.
13. C. Rich and C. Sidner. Adding a collaborative agent to graphical user interfaces. *UIST'96*. To appear.
14. C. Rich and C. Sidner. Collagen: When agents collaborate with people. *Agents'97*. To appear.
15. C. L. Sidner. An artificial discourse language for collaborative negotiation. *AAAI'94*, pp. 814-819.
16. A. Stein and E. Maier. Structuring collaborative information-seeking dialogues. *Knowledge-Based Systems*, 8(2-3):82–93, April 1995.