A SCHEMA BASED APPROACH TO DIALOG CONTROL

Paul C. Constantinides, Scott Hansma, Chris Tchou, Alexander I. Rudnicky

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA. 15213
{pcc,hansma,ctchou,air}@cs.cmu.edu

ABSTRACT

Frame-based approaches to spoken language interaction work well for limited tasks such as information access, given that the goal of the interaction is to construct a correct query then execute it. More complex tasks, however, can benefit from more active system participation. We describe two mechanisms that provide this, a modified stack that allows the system to track multiple topics, and form-specific schema that allow the system to deal with tasks that involve completion of multiple forms. Domain-dependent schema specify system behavior and are executed by a domain-independent engine. We describe implementations for a personal calendar system and for an air travel planning system.

1. INTRODUCTION

The success of frame-based information access systems, such as for the ATIS domain (e.g., Ward & Issar, 1994) and others (e.g, Goddeau et al., 1996), leads to the question of whether such an approach could be adapted for domains that may require more sophisticated dialog management. Ideally one should be able to preserve the discourse processing capabilities that can be realized within the frame paradigm, such as the use of current frame state to guide system initiative. This could then be augmented with additional data and control structures that allow a system to manage extended interactions, without altering the fundamental architecture of the system. A related issue centers on the separation of domain-specific computation and domain-independent discourse processes and to the degree to which these two can be successfully separated and domain-independent components reused for different applications, an issue of general interest (e.g, Aust et al., 1995).

This paper reports on ongoing work on dialog management in two application domains, personal calendar scheduling and air travel planning. In contrast to information access, both domains are characterized by the potential for extended interaction, though for different reasons. Scheduling can be viewed as a problem-solving task that focuses on the manipulation of a calendar data object. It can be effectively supported by providing the user with a small set of operators that allow the generation and validation of solutions. The task is doable on this basis because solutions need only respond to local constraints (i.e., is this time slot already occupied?) though the user may in fact need to engage in substantial search (our tasks presuppose a voice-only interface, without access to a graphical display). The air travel planning tasks differs in that the goal of working with the system is to create an extended, internally consistent, data structure, an itinerary. The size of the constrained data structure imposes additional requirements on the system's interactions with the user, for example keeping explicit track of the progress of the task and understanding necessary relationships between separate components of the itinerary (e.g., noting violations).

All systems described in this paper incorporate the Sphinx continuous speaker-independent recognition system (e.g., Ravishankar, 1996) as well as the Phoenix spoken language parser (e.g., Ward & Issar, 1994).

2. THE SCHEDULER DOMAIN

The Scheduler system supports a variety of activities related to accessing and modifying a personal information database. High level tasks are characterized as goals and broadly include creating, moving, or deleting appointments. The user performs a particular task (goal) through a sequence of interactions where each subsequent interaction turn reflects the address of a sub-goal. In this way, the discourse related to a particular goal consists of a sequence of sub-goal interactions. Whereas the user generally initiates goals, the system initiates dependent sub-goals to resolve the goal with the user. The Scheduler dialog control system selects an appropriate sub-goal to complete the goal's frame, where frames are simply tightly bound slots ("who will you be meeting with?"), or to resolve a discourse level operation ("should I make this appointment?"). Sub-goal selection is based on the context state in order to make progress towards the desired goal.

The dialog manager also has the ability to initiate goals during an interaction; specifically, this is the case when a sequence of turns results in a goal frame that is inconsistent with the state of the database. The system detects this, and informs the user of the inconsistency, giving the user the opportunity to choose between modifying the goal frame in context, or initiating a new goal to modify the database. An example of this is as follows:

U: Make an appointment with Eric at 3 p.m.
C: You already have an appointment with Kevin at 3.
U: Could you move that appointment to 4 p.m.?

The dialog control structure allows for only one goal to be in the foreground at a time. It does, though, provide a mechanism for sub-dialogs that could involve other goals. On completion of a goal, focus is returned to the last pending goal. This approach for conversational modeling is therefore stack-based.

This basic model serves as the backbone for the Scheduler dialog control system. Additional mechanisms, which focus on the limitations of speech based systems and of human capability in conversation, extend the basic model and broaden robustness of the system under real use. For example, we discovered that users found it difficult to manage dialog that involved more than two levels of depth on the stack. As with the example above, the user might push a Move goal on top of a Create goal in arranging a schedule. A user would not, however, proceed to stack another Move on top of a pending move; this would not represent typical use for most human users.

Limiting the depth of the context stack and assigning priority levels to goals are the primary mechanisms for achieving this end. Stack depth limitation seeks to ease the potential for burden on the user; while the system can handle pending context stacks of arbitrary depth, a user cannot. We have examined several approaches for implementing this including decay of goal initiatives, and forcing functions. The latter strategy restricts the user by rejecting new initiatives when the stack is at a maximum depth, whereas the former strategy discards goal frames if they have not been addressed for a number of successful turns (or if the frame becomes inconsistent with the state of the database). Goal priority attempts to leverage the tendency of a user to focus on completion of a particular task before moving on to another. The implicit ordering between goals allows the system to score the confidence of a user's request and ask for clarification if there is ambiguity.

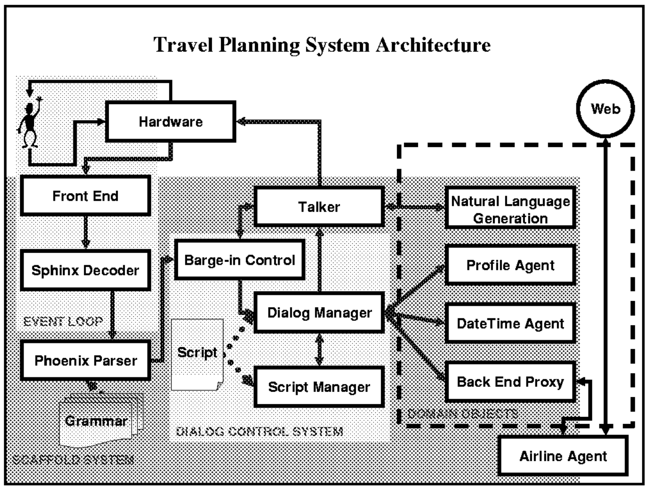

Figure 1: The system architecture

3. TRAVEL PLANNING DOMAIN

Travel planning differs from personal information management in that the planning task is characterized by a more complex structure, commensurate with the complexity of the itinerary that is created during a session. This provides both a benefit, there is a natural task structure that the system and the user can use to guide completion, and a difficulty, since the system needs to maintain suitable representations of this structure and be able to compute over them. From our perspective it is a useful task for studying complex dialog behavior.

In working with the travel-planning task we also took the opportunity to focus explicitly on the problem of creating a domain-independent architecture for the dialog component. This was achieved by separating dialog management into a computational engine and a declarative specification of its behavior (which was domain-specific). Further domain-specific computation was isolated into Domain Agents with well-defined interfaces covering their area of expertise (such as airline schedules, date and time interpretation). *Figure 1* shows the current system architecture.

3.1 Travel Planner 1

The dialog component of the Travel Planner makes use of two key data structures: a *product* and a *script*. Conceptually, the mutual goal for a session is to interactively create a product. In the travel domain this is an itinerary and is composed of a collection of individual forms corresponding to major product elements, such as the specification for a travel leg or for a hotel reservation. Scripts are schemas that describe a strategy for completing a goal by specifying the overall sequence of activity that will result in the creation of a finished product. The script is multi-level; the top level corresponds to the major elements of an itinerary (e.g., a leg, a car reservation); following the convention developed for the Scheduler, these are referred to as *goals*. Nested within each goal are *sub-goals*, corresponding to individual data elements, or *targets* (e.g., a date, a destination). Associated with each sub-goal element are additional specifications for system behavior, for example, which parser slots should be examined to fill the current target, which domain agent to invoke, and what information to include in a query or statement to the user. In addition to specifying how data might be acquired (from the user or from a domain agent), the goal statement includes descriptions of meta-interactions with the user, for example concerning the selection of a single flight from a set of such retrieved from the airline schedule domain agent.

The system for managing these script-based dialogs is implemented as two discrete components, a Script Manager (SM) and a Dialog Manager (DM). The Script Manager controls conversation flow through the script using a stack to model the conversation. In our initial implementation we separated the script into two components (the *script* proper in the SM and a corresponding *map* in the DM).

The Script Manager determines the current topic of conversation; this topical context is stored and referenced as a position within the script. The overall structure of the script defines the (ordered) set of topics that need to be addressed in order to complete the task. It's important to remember that at the schema level, order is not fixed but can be overridden by a topic shift initiated by the user. Within a topic schema order is controlled by *policy* which can be either *fixed* or *free*. System 1 implemented a fixed policy. Each script element, or schema, specifies a goal and its associated sub-goals, which become active when that schema is accessed. This script information is shared with the Dialog Manager, but script control is decentralized from dialog action and the main program. The Dialog Manager and Script Manager exchange the necessary information to carry out a dialog. The Dialog Manager passes an abstracted instance of the context to the script manager, which the Script Manager examines and returns the next necessary sub-goal.

The Script Manager manages the script position, or the topical context, using a stack. The top of the stack points to the script position that is the current topic of conversation. Other entries, below this on the stack, represent pending schema from previous discussion. These goals are pending, and are resolved when they ascend to the top of the stack.

Other things being equal, the current topical context proceeds through the script, addressing the goals and sub-goals necessary to complete the product. User initiatives can specify new goals and result in a topical context push that suspends the current schema to address the schema specified by the user ("Let's talk about the hotel in Denver."). This is also useful in cases where the user interrupts the current dialog, such as in a request for help. System initiatives may also lead to a context push operation, creating a sub-dialog for resolving inconsistent targets (constraint violations) or more generally, undertaking clarification dialog with the user.

Once a schema is completed, the Script Manager examines the context stack. The Script Manager will resume any previously suspended topic still on the stack before accepting a new schema from the script as the current focus of conversation.

Progress through a schema is accomplished by sub-goal completion. Reasoning over an abstracted representation of the context allows the Script Manager to direct conversation flow. The responsibility of acting on a sub-goal, though, lies within the Dialog Manager. With storage of the data context, and the ability to access other system modules, the Dialog Manager mediates between these various other parts of the system (such as language generation, parser, or other domain objects) to perform system activity and to manage and construct the context.

3.2 Travel Planner 2

The Travel Planner 1 framework allowed us to implement a system that could successfully help a user complete a single-city itinerary (i.e., to and from one destination). It also suggested that the representation we chose for dialog was sufficiently powerful for the class of systems that we were proposing to implement. The specification language we developed, however, would be more accurately characterized as an assembly-level pseudo-code than as an actual programming language for dialog, therefore a goal for the second implementation was to create a higher-level language that would be compiled into the dialog pseudo-code. The resulting language allowed us to rapidly develop and modify more complex versions of the Travel Planning task.

This high-level language, or schema language, merged the two types of specification discussed previously, unifying and extending both of the previous control components (script and map). Whereas the previous implemented specification method required a detailed understanding of the system operation, the new schema language was closer to a higher-level language using conventional primitive operators. Many of the previously used operators were expanded to reflect both additional functionality and refinement of the first implementation. The access structures to domain objects remained primarily unchanged; these allow the system designer to easily invoke domain object methods. The context structure was also unchanged, allowing any schema to access and modify the global context. Flow control was further abstracted, giving the designer the ability to more easily program explicit dialog strategies. Interaction with the parser was also improved, allowing a schema to examine in more detail the structure of the parse. These improvements to the system facilitated expansion of the System 1 travel planning schema implementation. After porting the script/map implementation to the new schema version, it was expanded. For example, we easily added more detailed schema related to time, hotels, and car rental, as well as expanded the script to handle multiple legged trips.

The System 2 schema language is programmed by a system designer and compiled into the script and map files for use at run time. The schema file is structured into two separate portions. At the top of the file, the designer declares the context and intermediate variables that the schemas will use. Variables are typed, and are specified as targets if they are part of the product. The other section in the file contains the schemas used to control dialog. Each schema constitutes a section which can contain assignment operators, calls to object methods (including back-end and language generation), if clauses, and invocations of other schema. The code here defines the dialog system behavior.

Further modifications to the new system schema were aimed towards adjusting the interaction policy. We observed that the first system's challenge/response form of interaction was limiting to most users. Specifically, users tended to express their requirements in a way that did not directly correspond with the system's expectations. To adjust for this form of interaction, we made a distinction specifying system response strategies as following either strict or free policy. The terms strict and free expresses the type of input that the system will accept with respect to expectation for response. A strict policy disallows the user from specifying information other than what the system expects (based on the current task, goal, or sub-goal); a free policy enables the user to express their constraints in any order they see fit. A free policy enhances the system's conversational ability, as natural conversation is not traditionally constrained. Rigid policy, although constraining, allows for other techniques that increase system understanding accuracy to be introduced. Generating an expectation for admissible new input allows the speech decoder to use language models constrained or biased towards this expectation. Additionally, it could allow the parser to better disambiguate the most valid parse using expectation in metrics for parse scoring. These methods can buffer otherwise fragile interactions that arise when noisy input leads to unreliable interpretations.

The System 2 version underwent noticeable usability improvement over the System 1 version. Expanded task coverage allowed a greater degree of freedom of types of itineraries that could be created; policy expansion permitted more flexible and rich interactions. The new framework allowed us to easily update and revise the dialog strategy yielding fast turnaround time for system improvement.

Although first set of changes yielded good overall improvements, there are still lessons within this system which we believe are worth exploring. Specifically, we are interested in augmenting the specification language. Sub-dialog and sub-process structures are not explicitly modeled in the current system, but rather implicitly arise under the schema framework.

This approach does not give the system designer a simple mechanism for implementing these classes of interaction. General representation of an extended set of discourse phenomena through primitives in the schema language could ease system design and enrich interaction.

The context management facility of the Dialog Manager has been sufficient to support our current implementations. It does not, however, support other context operations that could be of use. Specifically, state rollback is currently unsupported. Definition and propagation of constraints between context data is also unsupported. This facility is for flagging and acting upon user inputs that are inconsistent with the world or context state, in addition to providing "common sense" defaults for new schema.

The current system only allows for a single schema to be in focus at one time. As schemas are goals, the user cannot coordinate the completion of multiple goals through concurrent topic discussion. This may be valuable in cases where user constraints are related to dependencies between different goals.

We also believe that providing dynamic control over policy would be of benefit. This would allow a strict or free policy to be selected contingent on the characteristics of a session, such as input quality, user skill level, etc.

A broader problem, which affects dialog systems in general, is consistency checking between knowledge structures. A detailed description of this is outside the scope of this paper, but its role in system improvement is worth mentioning. The components in a dialog system rely on knowledge bases that are often – in implementation at least – independent. This poses a problem when these information sources are updated inconsistently with other knowledge sources, or when new information is not transferred to all other relevant sources. System bugs that arise from these inconsistencies are often the most difficult to trace. Automatic methods for conferring modifications across dependent knowledge sources would reduce or eliminate this problem.

4. SYSTEM EVALUATION

The system has been in operation since the spring of this year and has been available for experimental use by individuals beyond the development group (the existence of the system was publicized within the University, though not beyond it). To provide a preliminary evaluation, we analyzed calls recorded over a 12 day period. A total of 57 calls were recorded, from 26 different individuals.

The results from system use indicate that the majority of callers were able to complete meaningful dialogs. Specifically:

- 38 (or 2/3 of the calls) calls were successful and produced itineraries consistent with the constraints provided by the user, who remained within the domain as defined in the system.
- 9 calls were successful, resulting in consistent itineraries where the user had difficulty getting some information they requested; the user asks for information or service that was not understood by the system (i.e., not in the domain as implemented).
- 10 calls were unsuccessful due to back-end problems (2, schedule information was obtained over the Web), poor recognition (3, e.g., a child's voice), giving up for reasons seemingly unrelated to system performance (3), or user demands that exceeded system capabilities (1, demands which even a real travel agent could not satisfy)

Thus, our system overall achieved 82% task completion. Although most people booked two-leg flights, session reservations varied from one-leg to four-leg flights. The average completion time for two-leg flights was about 4 minutes and 10 seconds, as compared to about 3 minutes and 55 seconds for similar human to human sessions for two flight reservations. Although the times are comparable, it would be inaccurate, at this stage of development, to claim that the automatic system provides a level of service comparable to that of a human agent. With a view to this, we are exploring more diagnostic metrics for characterizing performance.

5. ACKNOWLEDGEMENTS

The work described in this paper would not have been possible without the help provided by various members of the Carnegie Mellon Speech Group.

This research was sponsored in part by the Department of the Navy, Naval Research Laboratory under Grant No. N00014-93-1-2005. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

6. REFERENCES

1. Ward, W. and Issar. S. Recent improvements in the CMU spoken language understanding system. In Proceedings of the ARPA Human Language Technology Workshop, March 1994, 213-216.
2. Goddeau, D., Meng, H., Polifroni, J., Seneff, S. & Busayapongchai, S. A form-based dialogue manager for spoken language applications. Proceedings of ICSLP, 1996, 701-704.
3. Aust, H., Oerder, M. Seide, F. and Steinbiss, V. The Philips automatic train timetable information system. Speech Communication 17, 1995, 249-262.
4. Ravishankar, M., Efficient Algorithms for Speech Recognition. Ph.D Thesis, Carnegie Mellon University, May 1996, Tech Report. CMU-CS-96-143.