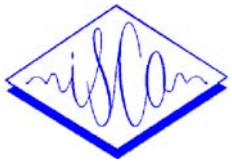

6th European Conference on
Speech Communication and Technology
(EUROSPEECH'99)
Budapest, Hungary, September 5-9, 1999

CREATING NATURAL DIALOGS IN THE CARNEGIE MELLON COMMUNICATOR SYSTEM

Rudnicky, A.I., Thayer, E., Constantinides, P., Tchou, C., Shern, R., Lenzo, K., Xu W., Oh, A.
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213 USA
{air,eht,pcc,tchou,shern,lenzo,xw,aliceo}@cs.cmu.edu
http://www.speech.cs.cmu.edu

ABSTRACT

The Carnegie Mellon Communicator system helps users create complex travel itineraries through a conversational interface. Itineraries consist of (multi-leg) flights, hotel and car reservations and are built from actual travel information for North America, obtained from the Web. The system manages dialog using a schema-based approach. Schemas correspond to major units of task information (such as a flight leg) and define conversational topics, or foci of interaction, meaningful to the user.

Keywords: Dialog, spoken language, systems

10.21437/Eurospeech.1999-344

1. INTRODUCTION

The Carnegie Mellon Communicator [3] helps a user create a travel itinerary consisting of air transportation, hotel reservations and car rentals. Domain information is obtained from open sources on the Web. Information is cached locally in the system and is regularly updated, or is accessed live during the course of a session.

The salient characteristics of the domain under study are the richness of the domain (in that many different interrelated pieces of knowledge need to be specified by the user) and the need for negotiation (in that a solution, the itinerary, requires a search over different possible solutions and the coordination of different constraints). Like others, (e.g., [1]) we see the process as one of cooperative problem solving. However we believe that the role of the computer is to support problem solving activity on the part of the user, with the user having primary initiative in the process, in the sense of setting goals and judging the acceptability of solutions. The role of the system is to provide information for decisions, propose solutions and to highlight potential constraint violations.

The travel-planning domain is interesting because it is difficult to approach in terms of existing techniques for managing spoken-language interaction. For example, the sequence of interactions needed to complete the task is not easily reduced to a fixed sequence of steps and alternatives (e.g., [8]); even if it were, users would likely find such a system difficult to use. For similar reasons, simple form-based approaches (e.g., [6]) are difficult to adapt to this domain. For example, the structure of the "form" is not always predictable in advance. Moreover, a significant portion of the interactions in this domain deals with the consequences of constraints imposed by the user's goals (which may change or evolve over the course of a session) and the possibilities afforded by the domain (which similarly may change over the course of a session).

2. TASK-BASED DIALOG MANAGEMENT

The successful completion of a task in this domain requires that, at the end of a session, the two agents (user and system) involved agree on a particular result (i.e., an acceptable itinerary). It also requires that, between them, they have some understanding of how to go about completing the task. This in turn requires that two types of knowledge be captured in systems that successfully supports this task: a representation for the domain-specific information that needs to be created and a representation that captures the structure of the activity needed to complete the task. It has been suggested that properly devised data structures that reflect the constraints imposed by the domain can be used to interpret user inputs and guide dialog (for instance, [4] [10]). We believe this is true, but only to a degree; it is also desirable to capture the structure of the activity that takes place (e.g. [7]) and thereby capture whatever expertise may be involved. This expertise reflects what humans know about performing tasks; it is both domain-specific and more general, representing "conversational skills" that may be transferable between domains.

In our approach, we represent the knowledge needed to conduct a dialog in terms of two principal data structures: A *product* that holds the result of the interaction and a set of *schema* that detail how elements of the product can be interacted about. The combination of product and schema generates necessary dialog-level behavior. Additional resources are needed to create a complete system. These include the components of a spoken language understanding system (a decoder and parser) as well as a set of domain agents. Domain agents are intended to handle all domain-specific information access and interpretation, with the goal of excluding such computation from the dialog management component. The dialog manager does contain domain-specific information, as described in schema. The set of schema developed for a task, collectively referred to as the script, is specified declaratively.

The itinerary is represented as a hierarchical data structure that is constructed interactively and is populated, over the course of a session, with information that (jointly) satisfies the constraints imposed by the user (who has a particular trip in mind) and the system (which has access to information about the domain, e.g., schedules and domain-specific constraints). The itinerary could be described as being composed of a dynamically constructed hierarchical form, with information implicit in the structure but also contained at different nodes in the tree. The tree structure allows the inheritance of information, allowing child nodes to naturally inherit information (e.g., the locale for a particular subset of arrangements) that had previously been elicited from the user. A given node otherwise acts as a conventional form, aggregating items of information pertinent to a particular stage of an itinerary (e.g., flight number, arrive and depart times for a particular leg).

Creating an itinerary consists of two activities: the composition of an itinerary structure and the population of this structure with trip-specific information. Both components are governed by schema. Constructor schema are used to build and edit the product structure. Thus, when a user says "I'd like to fly to New York tomorrow", the constructor will add subtrees corresponding to the outbound and inbound legs of the trip to the itinerary. Subsequent interactions can edit the tree ("and then I need to go to Montreal" would add another subtree, while "I'll just stay in New York" might prune the return leg subtree). Both the system and the user have access to constructors. For example, if the depart and return dates turn out to be the same, the system might simply edit out the subtree corresponding to the hotel arrangement.

The product tree defines a set of required conversational foci for a session, as well as their natural sequence, derived from a depth-first traversal of the itinerary tree. Foci correspond to nodes in the tree and each focus has an associated schema that details a conversational strategy. Schema detail information such as what domain-specific transforms to apply to a user input, language-generation specifications corresponding to data items as well as references to generalized conversational strategies (for example, subdialogs for navigating a solution set).

Topic focus is, by default, governed by an ordered agenda that specifies the list of topics that need to be covered for a particular itinerary. The agenda has a default ordering which is generated through a depth-first traversal of the current itinerary tree. Both the user and the system have control over the ordering and can thereby cause the focus to shift, rearranging the existing agenda ("let's talk about the first leg again") or introduce new (sub) topics ("I know of five airports in Hawaii. Would you like to fly to Honolulu?").

The correspondence between a user input and a topic is determined through a set of receptor nets associated with each schema. These specify the concepts that a user might speak in relation to the topic at hand. (Nets refer to concepts defined in the Phoenix semantic grammar used in the system.) The system selects a topic for focus by consulting the (ordered) agenda and activating the first schema that is able to accept the net(s) found in the user's utterance. In the normal course of events this will be the topic at the top of the current agenda. Control can pass to another schema under several circumstances. Trivially, when the schema for a given topic has completed (i.e., all necessary slots have been filled) that schema is popped from the agenda and the next schema is exposed. Alternately, if the top-most schema cannot receive the current input, subsequent schema on the stack are tested. A set of default schema, that always sort to the bottom of the agenda, catch inputs that do not correspond to topics on the agenda. These include schema that handle topic-shift requests, requests for help and one that provides a limited undo capability. Uninterpretable input (which would have been coded as "garble" at an earlier stage of processing) is absorbed by the bottom-most item of the agenda.

The agenda structure used in this system is a generalization of the stack structure developed for the Scheduler system (described in [3]). The former implementation allowed us to manage sub-dialogs and default behaviors. The agenda structure adds the ability to manage the multi-step creation of a complex product, using multiple schema.

3. SYSTEM ARCHTECTURE

3.1 Design

The system is completely modular and is implemented using DCOM and socket communication. The current implementation is being ported to the DARPA Hub architecture to explore issues of architecture. Individual modules communicate through message passing or through method invocation (blocking and non-blocking, depending on the nature of the service being sought). Messages are structured as frames that specify performatives as well as attribute-value pairs. The system is speech-only and is meant to be accessed through a telephone. It can in principle control a multi-modal interface but is not currently configured to do so.

3.2 Attention and decoding

Since it is telephone-based, the Communicator allows free input and does not enforce a fixed-turn protocol (where only one participant at a time can speak). To handle user barge-in we use a combination of hardware echo-cancellation and software speech detection, a modification of an algorithm implemented by [8] and based on energy levels and heuristic rules.

The Sphinx-II decoder is used in a real-time configuration, with acoustic models trained on a combination of band-passed ATIS, Broadcast News and task-specific data collected live using a system prototype. The decoder generates a confidence score on a word-by-word basis [2], which is used to determine whether the utter-

ance should be rejected. At the time of writing, the nominal vocabulary size is 2283. A class-based trigram language model is used, with 20 classes containing 1059 tokens, derived from a 58k-word corpus of transcribed speech, and described in [5].

3.3 Understanding

We use the Phoenix semantic parser [9], with a grammar derived from the CMU ATIS system, but substantially augmented based on transcriptions collected using the prototype system. We found that the travel-planning domain, although overlapping substantially with ATIS, contained significant amounts of new language both in terms of additional variants and in terms of new concepts.

Understanding also includes a post-parsing stage that performs a number of additional actions. In particular, post-parsing attempts to chose between multiple parses (if such occurred) and to judge the goodness of the understanding (if such is below threshold, the utterance is marked for rejection). The post-parse stage is also used to compute values for certain parts of a parse (for instance, a time expression which would be referred to the appropriate domain agent). These are then passed on as part of the parse tree and are available for subsequent processing. Specifically, this module examines features of the utterance parse, characterizing it with respect to its fragmentation and coverage over the utterance. The goal of the module is to derive information leading to an assessment of the parse quality, allowing the system to attempt a recovery rather than blindly acting on the information. A simple example of when this module is useful is in the case of contradictory specifications or user edits: "yes, uh, no", or "I'd like to go to Chicago, uh, Boston".

3.4 Dialog Manager

The Communicator Dialog Manager isolates all dialog-relevant computation in a single module; it is also designed to be a domain-independent engine whose behavior is specified through a task-dependent script (together with Domain Agents). Our goals were to isolate dialog processes from other aspects of the system and to provide a locus for specifying the structure of activity (i.e., domain and conversational skills). Domain-specific processing is meant to be encapsulated in Domain Agents (such as Date and Time, Flight Schedules, etc.) Based on analyses of human-human travel planning [5], we implemented scripts that would capture the conventional activities associated with the performance of a complex task. We refer to these distillations as *schema*, which correspond to specific sub-tasks in the travel planning activity. Collectively these form a *script* that provides the system with structure for interacting with the user on a particular topic. It is important to understand that the script does not provide a fixed order in which a task might be performed. This order is generated from the agenda.

In conjunction with the script, the system uses a *target* data structure to hold the product of the interaction. Individual schema are keyed to portions of the target structure and relate user inputs to individual targets.

We believe the schema structure can be used to capture both a default strategy that the system should follow for achieving different subgoals and the detail of how to interact with the user in specific situations, for example, in navigating a solution set or in carrying out confirmation sub-dialogs. Sub-dialogs are in fact discrete entities, corresponding to schema and can be invoked recursively using the stack mechanism.

In [3] we described a mechanism for setting interaction policy. Two policies were defined, a rigid policy that required the user to respond to queries in a fixed order (sometimes described as a 'system-initiative" dialog) and a free policy that allowed the user to volunteer information relevant to the current topic (described as a "mixed-initiative" dialog). Switching between policies can be made contingent on the occurrence of garbled inputs (see above) and could be used to modify processing strategies, say at the decoder or parser levels. Such interaction policies are easily implemented as different schema, however it has been our experience that users generally dislike the rigid policy, consequently we use the free policy.

The goal for the script is to allow the designer to specify task and discourse-level knowledge for a system. The scripting language promotes this goal by providing abstractions for common operations (such as the extraction of meaning from a parse tree) and providing facilities for interacting with domain agents. In principle no operations to do with domain information should take place within the dialog manager.

3.5 Domain Agents

All domain-specific computation is "outsourced" by the Dialog Manager (which however captures the structure of the task itself). Currently, the following agents are used in the system: a travel information agent, a date and time interpretation agent, and a user model agent. The air travel information agent has additional internal structure that allows it to service requests asynchronously. An initial request for flight information is serviced using a cached schedule; subsequent requests for that leg use live retrievals. The architecture allows for the inclusion of arbitrary additional agents, so long as these provide interfaces that subscribe to the existing message passing protocol. Whether a domain agent is added to the system depends on whether a distinct sub-domain of knowledge can be demarcated within the larger application. Domain Agents can contact each other for services and do not need to communicate through a central point (such as the Dialog Manager).

3.6 Language generation

Language generation is template-driven; a desired output is specified by a performative along with a set of attrib-

utes that specify information potentially to be spoken. Actual utterances are constructed using phrase and sentence-based templates. Information on what performative to express and with which attributes is specified in the dialog schema. The generation component incorporates a history mechanism that allows generation to modify the final form of the spoken utterance by filtering attributes that appear in the output. We have used this mechanism to control processes such as the generation of implicit confirmation and managing the use of old and new information on a topic-by-topic basis.

3.7 Supervisor mode

The Communicator also incorporates a human supervisor component. Under certain defined circumstances, the system will decide that the user is having difficulties that are beyond its current capabilities. It will then request help from a human (i.e., a member of the development team) who can mediate between the user and the system. Under normal circumstances, the supervisor system passively monitors an ongoing interaction. System diagnostics use rule based reasoning to automatically determine if the interaction has gone awry, at which point, the system confers responsibility for the interaction to the human supervisor who can assume the role of either the human or the system. Introducing human intervention in cases of system failure allows us to gain a better understanding of failure modes and allows us to characterize reasonable recovery strategies (as devised by a human). Such strategies can be later automated and incorporated into the system.

4. CONCLUSION

We believe that the dialog management approach we have described in this paper will scale with the complexity of the underlying domain and the tasks that people expect to be able to perform.

As complexity increases, there is a corresponding need to automate the acquisition of knowledge. Apart from domain knowledge, knowledge needs to be dynamically acquired at different levels of the system, to track the evolution of a domain and of system users. The problem, if not the exact solutions, of knowledge acquisition is understood at some levels of modeling, such as acoustic, lexical and language. We are interested in extending the automatic acquisition of knowledge to grammar, dialog and generation levels of the system and in better understanding the integration of domain knowledge and dialog knowledge.

5. ACKNOWLEDGEMENTS

We would like to thank Maxine Eskenazi, Wayne Ward, Ravi Mosur, Rita Singh and Kristie Seymour for discussions as well as their help in various phases of this work.

This research was sponsored by the Space and Naval Warfare Systems Center, San Diego, under Grant No. N66001-99-1-8905. The content of the information in this publication does not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred.

6. REFERENCES

[1] Allen, J., Miller, B., Ringger, E., and Sikorski, T., A Robust System for Natural Spoken Dialogue, *Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics (ACL'96)*, June 1996. pp. 62-70.

[2] Bansal, D. and Ravishankar, M. New features for confidence annotation. In *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP)*, December 1998, Sydney, Australia

[3] Constantinides, P., Hansma, S. and Rudnicky, A.I. A script-based approach to dialog management. *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP)*, 1998.

[4] Denecke, M. and Waibel, A. Dialogue strategies guiding users to their communicative goals. *Proceedings of Eurospeech97*, September 1997, Rhodes, Greece

[5] Eskenazi, M., Rudnicky, A., Gregory, K., Constantinides, P., Brennan, R., Bennett, T. and Allen, J. Data collection and processing in the Carnegie Mellon Communicator. *(these proceedings).*

[6] J. Polifroni, S. Seneff, J. Glass, and T.J. Hazen. Evaluation Methodology for a Telephone-based Conversational System (postscript). *Proc. First International Conference on language Resources and Evaluation*, pp. 42-50, Granada, Spain, May 1998.

[7] Rich C. and Sidner, C. COLLAGEN: A Collaboration Manager for Software Interface Agents *User Modeling and User-Adapted Interaction*, Vol. 8, No. 3/4, 1998, pp. 315-350

[8] Rudnicky, A.I., Reed, S. and Thayer, E.H. SpeechWear: A mobile speech system , *Proceedings of the 4th International Conference on Spoken Language Processing (ICSLP)*, 1996.

[9] 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.

[10] Wright, J., Gorin, A. and Abella, A. Spoken language understanding within dialogs using a graphical model of task structure. In *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP)*, December 1998, Sydney, Australia