A Multiple-Application Conversational Agent

Steven Ross
IBM Research
1 Rogers St
Cambridge, Massachusetts
617 693 5746
Steven_Ross@us.ibm.com

Elizabeth Brownholtz
IBM Research
1 Rogers St
Cambridge, Massachusetts
617 693 4148
Beth_Brownholtz@us.ibm.com

Robert Armes
IBM Research
1 Rogers St
Cambridge, Massachusetts
617 693 5446
Robert_Armes@us.ibm.com

ABSTRACT
In this paper, we describe the rationale behind and architecture of a conversational agent capable of speech enabling multiple applications

Categories and Subject Descriptors
H5.2 [**INFORMATION INTERFACES AND PRESENTATION**]: User Interfaces - *Voice I/O, Natural Language*

General Terms: Algorithms, Design, Human Factors

Keywords: Speech processing, natural language, conversational agent, multimedia interfaces.

1. INTRODUCTION
Speech recognition research has been ongoing for more than 50 years, but it is only within the last decade that large-vocabulary, continuous-speech recognition capability has become available on personal computers. Our team, the Lotus Speech Initiative, was formed to investigate how we could leverage the state of the art in speech recognition and synthesis technology to enhance interaction between people and computers.

Rather than focusing on using speech as an alternative to keyboard, mouse, and screen, our project viewed speech as an additional channel of communication between the user and the computer. We saw speech interaction as providing a means to "talk around" windows, enabling users to communicate with background processes without changing their focus of attention. This suggested a command and control system that would seamlessly blend speech interaction with multiple applications, blurring the boundaries between them and creating a virtual speech environment to which individual applications would provide added capabilities.

Our team developed a prototype that we call The Lotus Conversational Interface (LCI). This single application serves as a focal point for all speech activity on the desktop and provides speech services to all participating applications. LCI has been used to speech-enable applications that provide email, instant messaging, expertise location, stock quotes, alarm clock, and calculator services. Our goals for LCI went beyond multiple application support, however. Our aim was to develop a system that provided a productivity advantage. We wanted it to be natural, easy, and fun to use, and to avoid annoying the user.

IUI'04, Jan. 13–16, 2004, Madeira, Funchal, Portugal.
ACM 1-58113-815-6/04/0001.

To accomplish these goals, we formulated a set of voice user interface principles that we strove to implement in our system. We also wanted a design that would minimize the work required to speech-enable an application. Ideally, applications would be speech-enabled without modification to the application itself, and certainly without modification to the core speech application. The combination of these requirements led to an interesting and innovative software architecture.

2. VOICE USER INTERFACE PRINCIPLES
The voice user interface principles that we developed are intended to make the conversational agent behave as a faithful servant: helpful yet respectful. We modeled our agent after the prototypical English butler, standing politely at the ready. The principles are divided into three broad categories according to the goal that they promote.

The first set of principles is intended to encourage natural interaction and politeness in the interface. The second set of principles promotes dependability and trustworthiness in the interface. These principles are motivated by the imperfect nature of speech recognition today. Users need assurance that they have been heard correctly and that the correct operation is being performed, or that the answer being delivered is the answer to the question they asked. The third set of principles supports consistency and transparency in the user's mental model of the agent.

3. ARCHITECTURE
Our approach to handling multiple applications was to associate one or more grammars with each application. Each grammar defines a set of utterances that will be accepted by the speech recognition engine. Statistically-based approaches [2] to specifying acceptable utterances were considered but were rejected because of the difficulty of collecting and annotating large numbers of utterances, and the problems inherent in combining separate statistically-based applications. In LCI, the grammars are simultaneously active in the speech engine, and are kept in a prioritized most-recently-used list for determining which application will handle a particular utterance. Since the grammars for different applications may overlap, and the speech engine that we use, IBM ViaVoice [4], may recognize a particular utterance as being associated with any of the grammars that accept it, we test recognized utterances against the grammars in priority order to determine the target application for the utterance. The list of grammars is reordered when an application is interacted with, regardless of whether it is via speech, mouse, or keyboard.

Each speech-enabled application in LCI is defined by an ontology, syntax specification, lexicon, rule-base, and script. The ontology defines the objects and classes in the application domain, and

describes each object or class in terms of its attributes and their value classes. This specification provides a crude semantic model for the application domain, but it says nothing about how one would talk to the application. The syntax specification provides a high-level description of the forms of legal utterances in terms of their ontological constituents.

The LCI Syntax Manager creates a grammar suitable for the speech engine from the ontology, syntax specification, and lexicon (which provides synonyms and part-of-speech information for the words defined in the ontology.) This mechanism allows large, complete and consistent grammars to be generated from relatively compact specifications. Furthermore, since the syntax is specified in terms of its semantic constituents, the generated grammar can be automatically annotated with semantic information sufficient to construct a representation of the meaning of any accepted utterance. Annotating a grammar with semantic information is described in [1]. Our syntax templates allow this process to be automated.

A recognized utterance is translated to a frame-structured semantic representation using a recursive substitution translation mechanism. A second level of semantic analysis distills the frame representation to a set of attribute-object-value triple propositions.

Having determined what an utterance means, the system must now decide what to do about it. The propositions are fed into a simple goal-directed, rule-based system, which reasons about how to process the utterance. Rules in the rule-base consist of some number of conditions and some number of actions, any of which can be either a proposition or a script call. Some rules function as a sort of machine-readable documentation for script functions, effectively saying that under a set of specified conditions, a call to a particular script function will achieve a particular result. The conditions specified in a rule can refer to the semantic content of the request itself or to other salient aspects of the environment, so the same request can result in different outcomes under different circumstances, when appropriate. Other rules function in a supporting role, providing the means to disambiguate names or resolve references. The reasoning component uses these rules in a goal-directed manner to attempt to satisfy the overarching goal of processing the utterance. Conversation emerges from this process, rather than being pre-scripted, as the reasoning facility eventually discovers that it cannot resolve an ambiguity or derive missing information that it needs, and a question to the user is generated. Language is generated using language generation rules and syntax templates in a reverse of the process used for understanding an utterance, translating from propositional form back to English text. The net result of the reasoning process is that the proper script calls are made to induce the application to take the actions that will accomplish the user's command, or retrieve the information that will answer the user's question.

The use of a rule-based approach to controlling dialog is described in [7]. In our system, the rules describe how to take action or derive needed information, and the dialog emerges as a consequence of rule interpretation. LCI's dialog manager moderates the flow of incoming and outgoing utterances. It ensures that the system does not interrupt when the user is speaking, requests permission to speak when a response is unexpected or delayed, and keeps track of commands that require further information from the user.

The high level architecture of LCI is similar to that of the DARPA Communicator [6], consisting of a central messaging hub through which the various modules communicate. A set of external interface modules isolate the system from speech engine, script engine, and operating system dependencies.

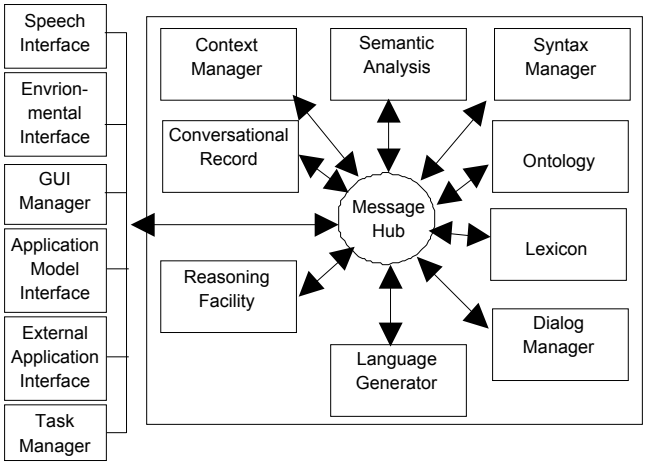

Figure 1. LCI Top Level Architecture.

4. LESSONS LEARNED FROM USE

To date, LCI has been used to speech-enable seven different applications. Our high-level specification languages allowed rapid speech enabling of new applications. For example, we speech-enabled a simulated automobile in the course of just a few days. Providing voice access to climate control, telephone, radio, windows, doors, and navigation functions was accomplished with less than one hundred twenty five syntax templates and two hundred rules.

Many people have used LCI in formal observed sessions and informally over sustained periods. The most striking lesson we've learned is how critically important recognition accuracy is in speech-based user interfaces. It is remarkable how quickly the productivity advantage of the system evaporates if a command is misrecognized and the user has to recover or repeat herself. Some users' annoyance levels rise rapidly if the system fails to recognize a legitimate command or incorrectly interprets background noise as a command. We implemented an optional push-to-talk feature that was particularly helpful in avoiding extraneous command recognitions, but the loss of the hands-free aspects of the interaction was too high a price to pay for some users. Clearly, the better the recognition accuracy, the more acceptable the voice interface would be. This was demonstrated over the course of this project, when we were fortunate to have speech engine upgrades available on several occasions. Each one was better than the last, and each one improved the experience of working with LCI.

Although less important than speech recognition accuracy, the quality of the speech synthesis used in the system is also very important. The speech synthesis system we used was fairly easy to understand, but the quality of the voice was grating enough that it was tolerable only for short interactions. Although users could have the computer read messages out loud, they rarely did so for this reason. Using concatenative speech synthesis techniques [3] may provide a more pleasant voice and consequently garner greater user acceptance.

For the most part, adherence to our voice user interface principles resulted in the quality of interaction that we were looking for, but practical experience dictated that some of our principles had to be modified. For example, we abandoned our plan to have the system stop talking when it heard a new command because when the system detected the audio of its own speech synthesis, it stopped talking. It was constantly interrupting itself. Echo cancellation technology in the speech engine would make "barge-in" detection practical. We also had to modify the principle governing when the system asked permission to speak. Our original plan was to require permission if the user issued a new command before the system responded to an earlier command, based on the assumption that the user had changed the subject and that the first response was no longer expected. In practice, however, users frequently issued a rapid series of commands, which led to an annoying number of polite interruption requests. Removing this requirement allowed the system to deliver out-of-sequence responses without permission if too much time had not elapsed, yet this did not seem to confuse our users.

5. DIRECTIONS FOR FUTURE WORK

A great deal of interesting work remains to be done in this area. Our work in grammar generation rapidly resulted in larger grammars than the ViaVoice grammar compiler could handle, so we ultimately had to scale back our syntax and modify the compiler itself in order to proceed. More attention could be directed to both of these areas.

Our language generation techniques, while adequate for our purposes, were rather simplistic and could be enhanced. It would be useful, for example, to be able to work backwards from a set of propositions to a grammatical utterance whose meaning was captured by those propositions, as described in [5]. That would allow paraphrasing of commands and language generation without resorting to special purpose rules and templates in some circumstances.

Several aspects of our original plan have not yet been implemented due to time and manpower constraints. One such feature is a "foundation domain model." We wanted to develop a common base definition of ontology, syntax templates, rules, and so on that would be inherited and extended by individual application definitions. For example, the basic structures of command and question processing, reference resolution, and name handling would be similar across many applications. Centralizing this functionality in the foundation would simplify application development and promote consistency across applications. Another aid to application development that we have not yet pursued is a set of tools to provide an integrated speech application development environment. Such tools would allow definition, editing, testing, and debugging of all aspects of the application definition, and would ease the development process further.

Extending the scope of our domain models could give LCI a wider perspective of how individual commands fit within the larger space of the tasks that the user is engaged in and the goals that the user is pursuing, as in [8].

While LCI was originally developed to operate on desktop computers, we see great potential for applying conversational agents in other areas as well. For example, the benefits of conversational interaction would be even greater in handheld or automotive environments where non-conversational interaction is inconvenient or even dangerous.

6. CONCLUSION

Conversational interaction with computers has been a staple of science fiction for years, and seems to be an inevitable development whose time has not quite yet arrived. With LCI, we have gotten a taste of what interaction with a conversational agent can be, and we remain enthusiastic that with continuing improvements in the quality of speech recognition and synthesis systems, this technology will eventually have its day.

7. ACKNOWLEDGMENTS

We would like to thank all of the members of the Lotus Speech Initiative team, particularly Julie Alweis, Jeff MacAllister, Kathy Howard, and Majie Zeller, and all the supporters of our work over the past several years within IBM. Special thanks to Whitney Grunwald, David Lubensky, and Irene Greif, without whom our vision would never have been realized.

8. REFERENCES

[1] Allen, James, Natural Language Processing, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA (1995)

[2] Della Pietra, Stephan, Epstein, Mark , Roukos, Salim, and Ward, Todd, Fertility Models for Statistical Natural Language Understanding. ACL 1997, pp. 168-173

[3] Donovan, R.E. and Eide, E.M., The IBM Trainable Speech Synthesis System, Proce. ICSLP'98, Sydney, Austraila (1998).

[4] IBM, Speech Manager Application Programming Interface Developers Guide, IBM ViaVoice SDK for Windows (August 1998)

[5] Jurafsky, Daniel, and Martin, James H., Speech and Natural Language Processing, Prenctice Hall, Upper Saddle River, New Jersey (2000)

[6] MITRE Corporation. 1999. DARPA Communicator Web Page. http://fofoca.mitre.org/

[7] O'Neill, Ian M. and McTear, Michael F., Object-Oriented Modelling of Spoken Language Dialog Systems, *Natural Language Engineering*, Best Practice in Spoken Language Dialogue System Engineering, Special Issue, Volume 6 Part 3 (October 2000)

[8] Rich, Charles and Sidner, Candace L., COLLAGEN: When Agents Collaborate With People, Int. Conf. on Autonomous Agents, Marina del Rey, CA, (Feb. 1997)