SPEECHBUILDER: Facilitating Spoken Dialogue System Development

James Glass and Eugene Weinstein

Spoken Language Systems Group
MIT Laboratory for Computer Science
Cambridge, Massachusetts 02139, USA
{glass, ecoder}@mit.edu

Abstract

In this paper we report our attempts to facilitate the creation of mixed-initiative spoken dialogue systems for both novice and experienced developers of human language technology. Our efforts have resulted in the creation of a utility called SPEECHBUILDER, which allows developers to specify linguistic information about their domains, and rapidly create spoken dialogue interfaces to them. SPEECHBUILDER has been used to create domains providing access to structured information contained in a relational database, as well as to provide human language interfaces to control or transaction-based applications.

1. Introduction

As anyone who has tried to create a mixed-initiative spoken dialogue system knows, building a system which interacts competently with users, while allowing them freedom in what they can say and when they can say it during a conversation, is a significant challenge. For this reason, many systems avoid this tactic, and instead take a more strategic approach which focuses on a directed dialogue. In fact, many researchers argue that conversational, mixed-initiative dialogue systems may not be worth pursuing, both for practical and philosophical reasons. Certainly, there are many technical difficulties to overcome, which include recognizing and understanding conversational speech, generating reasonable and natural responses, and managing a flexible dialogue strategy.

Over the past decade, the Spoken Language Systems Group at the MIT Laboratory for Computer Science has been actively developing the human language technologies necessary for creating such conversational human-machine interfaces. In recent years we have created several systems which have been publicly deployed on toll-free telephone numbers in North America, including systems providing access to information about weather forecasts, flight status, and flight schedules and prices [1, 2].

Although these applications have been successful, there are limited resources at MIT to develop a large number of new domains. To address this issue, we have recently set out to make it easier to rapidly prototype new mixed-initiative conversational systems. Unlike other portability efforts that we are aware of, which tend to employ directed-dialogue strategies (e.g., [3]), our goal is to enable the kinds of natural, mixed-initiative systems which are now created manually by a relatively small group of expert developers.

In this paper we describe our initial efforts in developing a utility called SPEECHBUILDER. The next section describes the architecture we have adopted. This is followed by a description of the human language technologies which are deployed in SPEECHBUILDER, and the knowledge representation it uses. We then describe the current state of development, followed by ongoing and future activities in this project.

2. System Architecture

The approach that we have adopted for developing the SPEECHBUILDER utility has been to leverage the basic technology which is deployed in our more sophisticated conversational systems. There are many reasons for doing this. First, in addition to developing a programmable client-server architecture for conversational systems [4], we have devoted considerable effort over the last decade to improving human language technology (HLT) in speech recognition, language understanding, language generation, discourse and dialogue, and most recently, speech synthesis. By employing these HLT components we minimize duplication of effort, and maximize our ability to adopt any technical advances which are made in any of these areas. Second, by using our most advanced HLT components, we widen the pool of potential developers to include both novices and experts, since the latter can use SPEECHBUILDER to rapidly prototype a new domain (a very useful feature) and subsequently modify it manually. Third, since we are not limiting any of the HLT capabilities in any way, we allow for the potential for SPEECHBUILDER-created systems to eventually scale up to the same level of sophistication as our most capable systems. Lastly, by focusing attention on portability as a major issue, we can potentially identify weaknesses in some of our existing HLT components. This can lead to better solutions, which will ultimately benefit all of our conversational systems.

We have made a conscious decision to have as simple an interface as possible for the user, while providing mechanisms to incorporate any needed complexities. For example, developers do not specify natural language grammars for their domain. Instead, they specify the basic semantic concepts (called *keys*), and provide examples of user utterances which trigger different system behaviors (called *actions*). The system then uses these inputs to configure the language understanding grammar automatically. The developer can optionally create additional hierarchy in the grammar by using bracketing to label portions of the example sentences as being subject to a particular structure. Table 1 contains example sentences and their corresponding key/action representations encoded as CGI parameters (which are used for one kind of SPEECHBUILDER configuration).

Names in alphabetical order. This research was supported by DARPA under contract N66001-99-1-8904 monitored through Naval Command, Control and Ocean Surveillance Center and under an industrial consortium supporting the MIT Oxygen Alliance.

To be presented at the 7th European Conference on Speech Communication and Technology, Aalborg, Denmark, September, 2001.

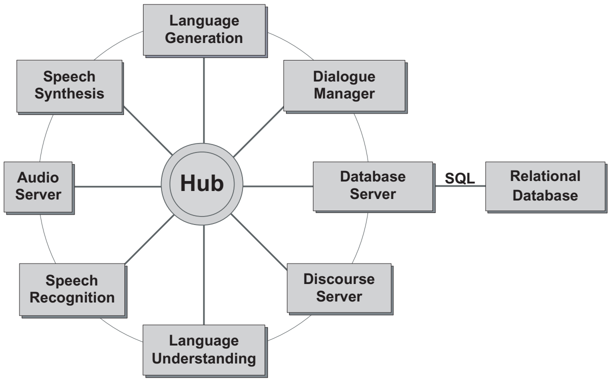

Figure 1: SpeechBuilder configuration for database access.

turn on the lights in the kitchen **action=set&frame=(object=lights,room=kitchen,value=on)**
will it be raining in Boston on Friday **action=verify&frame=(city=Boston,day=Friday,property=rain)**
show me the Chinese restaurants on Main Street **action=identify&frame=(object=(type=restaurant, cuisine=Chinese,on=(street=Main,ext=Street)))**
I want to fly from Boston to San Francisco arriving before ten a m **action=list&frame=(src=BOS,dest=SFO, arrival_time=(relative=before,time=(hour=10,xm=AM)))**

Table 1: Example sentences and their CGI representations.

2.1. SpeechBuilder Configurations

There are currently two ways in which a SpeechBuilder application can be configured. The first configuration can be used by a developer to create a speech-based interface to structured data. There is no programming required. As shown in Figure 1, this model makes use of the GALAXY architecture and all of the associated HLT components to access information stored in a relational database which is populated by the developer. This database table is used along with the semantic concepts and example utterances to automatically configure the speech recognition, language understanding, language generation (including both SQL and response generation), and discourse components (described in Section 3). A generic dialogue manager handles user interactions with the database. Armed with a table of structured data, an experienced developer can use SpeechBuilder to create a prototype system in a matter of minutes.

The second possible SpeechBuilder configuration provides the developer with total control over the application functionality, as well as the discourse, dialogue, and response generation capabilities [5]. In this model, the developer creates a program implementing domain-specific functionality and deploys it on a CGI-enabled web server. As shown in Figure 2, this configuration uses a subset of the GALAXY components. The semantic frame is converted to the CGI parameter representation shown in Table 1 by means of the language generation component and is sent to the developer CGI application using HTTP.

Since the developer CGI application is stateless, the SpeechBuilder server maintains a dialogue state variable which is exchanged with the CGI application on every turn. It is the responsibility of the application to decide what information, if any, to inherit from this state variable, and what information to retain for the next dialogue turn. Table 2 shows how the state information can be used to keep track of local discourse context.

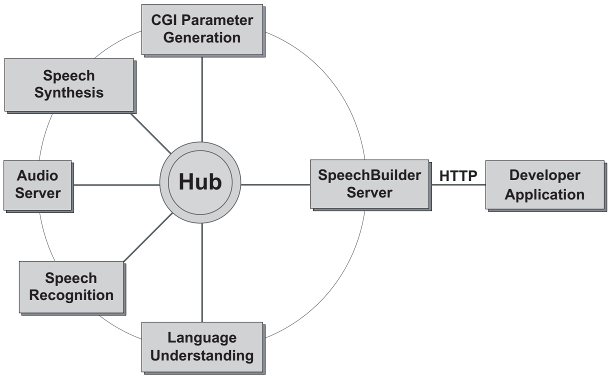

Figure 2: Alternative SpeechBuilder configuration.

what is the phone number of John Smith **action=identify&frame=(property=phone,name=John+Smith)**
what about his email address **action=identify&frame=(property=email)** **&history=(property=phone,name=John+Smith)**
what about Jane Doe **action=identify&frame=(name=Jane+Doe)** **&history=(property=email,name=John+Smith)**

Table 2: Example interaction using the dialogue state variable.

2.2. Web-Based Interface

SpeechBuilder employs a web-based interface, and is implemented via a number of Perl CGI scripts. The web interface allows the developer to manipulate all of the domain specifics, such as keys and actions, query responses, and vocabulary word pronunciations. These domain specifics are stored in an XML document, and the developer has the ability to edit this file manually without using the online interface. The web interface contains a facility for downloading and uploading the XML representation of a domain, and for uploading CSV (comma separated value) representations of data tables. The developer also uses the web interface to compile the domain (that is, to configure the GALAXY servers according to domain specifics), and to start and stop the runtime modules of the domain. Once a domain has been compiled, the developer can also examine the parse tree and semantic frame produced for each example sentence. The interface makes it very easy for a developer to continually modify, recompile, and redeploy a domain during the development cycle.

3. Human Language Technologies

As shown in Figure 1, SpeechBuilder makes use of all the major GALAXY components [4]. The programmable hub executes a program which was created specifically for the SpeechBuilder application, although it contains all of the functionality of the programs used by our main systems. A specific version of the hub program is configured for each developer.

The speech recognizer is configured to use generic telephone-based acoustic models, and is connected to the language understanding component via an N-best interface [6]. Since users may speak words which are not specified in the vocabulary, we have incorporated an out-of-vocabulary model [7]. Baseform pronunciations which do not occur in our large online dictionaries are generated by rule [8]. SpeechBuilder provides an editing facility for developers to modify pronunciations. The recognizer deploys a hierarchical n-gram grammar

Key	Examples
color	red, green blue
day	Monday, Tuesday, Wednesday
room	living room, dining room, kitchen

Table 3: Examples of concept keys.

derived from the language understanding grammar rules and the example sentences provided by the developer.

For language understanding, SpeechBuilder configures a grammar file as well as a file for converting full or partial parses into a meaning representation [9]. Language understanding is configured to back off to robust parsing (i.e., concept spotting) when no full parse is available. The discourse server is based on a new component which performs inheritance and masking after an internal electronic 'E-form' has been generated from the semantic frame [2].

The dialogue management server was modeled after the functionality of our main systems [2]. Since this component still tends to be extensively customized for every domain, we created a simple generic server which is intended to handle the range of situations which can arise in database query domains. We fully expect this component to increase in complexity as we consider a wider range of domains.

The language generation component is actually used for generating three different outputs, as it is in our main systems [10]. The first use of generation is for creating the 'E-form' representation used by the discourse and dialogue components. The second use is to generate an SQL query for use by the database server. The third use is to generate a response to the user which is vocalized using the speech synthesizer.

4. Knowledge Representation

Linguistic constraints are specified by a developer in terms of a set of concept keys and sentence-level actions via the web interface. Each is described in more detail in the following.

4.1. Concept Keys

Concept keys usually define classes of semantically equivalent words or word sequences. All the entries of a key class should play the same role in an utterance. Concept keys can be extracted from the database table or can be manually specified by the developer through the web-based interface. In order to appear in the semantic frame a concept *must* be a member of a concept key class. A regularization mechanism allows the developer to specify variations on the spoken form (e.g., "Philadelphia," vs. "Philly") that map to a standardized form (e.g., "PHL"). Table 3 contains examples of concept keys.

4.2. Sentence-level Actions

Actions define classes of functionally equivalent sentences, so that all the entries of an action class perform the same operation in the application. Actions are example queries that one might use in talking to the domain. Action labels determine the clause name of the semantic frame produced by the language understanding component. SpeechBuilder generalizes all example sentences containing particular concept key entries to accept all the entries of the same key class, and thus builds the natural language template. SpeechBuilder also tries to generalize the non-key words in the example sentences so that it can understand a wider variety of user queries than was provided by the developer. Table 4 contains example actions.

Action	Examples
identify	what is the forecast for Boston what will the temperature be on Tuesday I would like to know today's weather in Denver
set	turn the radio on in the kitchen please can you please turn off the dining room lights turn on the TV in the living room

Table 4: Examples of actions.

Put object==(the blue box) location==(on the table) **object=(color=blue,item=box),location=(item=table)**
Put object=(the blue box) location=(on the table) **object=(blue_box),location=(table)**
Put the box location==(on the table location==(in the kitchen)) **item=box,location=(relative=on,item=table,** **location=(relative=in,room=kitchen))**

Table 5: Examples of strict (==) and flattened (=) hierarchy.

The system created by SpeechBuilder can potentially understand a larger set of queries than are defined by the set of sentence examples, since the examples are converted into a hierarchical n-gram for the recognizer, and the understanding component backs off to concept spotting when a complete parse is not found. However, the system performs better if given a richer set of examples.

4.3. Hierarchical Concept Keys

SpeechBuilder allows the developer to build a structured grammar when this is desired. This is done by "bracketing" example sentences within the actions – using parentheses to enforce a structure on the way that each particular sentence is parsed. The semantic frame created by the natural language processor reflects the hierarchy specified by the bracketing.

To bracket a sentence, the developer encloses the substructure which they wish to separate in parentheses, preceded by a name for the substructure followed by either == or =, depending on whether the developer desires to use *strict* or *flattened* hierarchy. Strict hierarchy maintains the key/value structure of all concept keys present in the bracketed text. In contrast, flattened hierarchy collapses all internal key/values into a single concept value. Table 5 shows examples of bracketed example sentences which make use of the hierarchical concept keys.

4.4. Responses

In addition to configuring ways of asking about information, the developer must also specify how the system will present information to the user. When a database is provided by the developer, SpeechBuilder configures a generic set of replies for each appropriate database query which could be requested by a user. In addition, responses are generated to handle situations common to all database applications (e.g., no data matching the input constraints, multiple matches to a user query, too many matches, etc.). Each of these default responses can be modified by the developer, as desired, to customize the domain.

5. Current Status

SpeechBuilder has been accessible from within MIT and limited other locations for beta-testing since November 2000, and it has been used to create several different domains. The *LCSinfo* domain, for example, provides access to contact information for the approximately 500 faculty, staff, and students working at the MIT Laboratory for Computer Science (LCS) and Artificial Intelligence Laboratory (including phone numbers, email addresses, room locations, positions, and group affiliations) and is able to connect the caller to any of the people in the database using call bridging. Applications similar to *LCSinfo* domain are under development by novice developers elsewhere at MIT, as well as at the Space and Naval Warfare Systems Center in San Diego, CA. SPEECHBUILDER has also been used to create a simple appliance application which controls various physical items in an office (e.g. lights, curtains, projector, television, computer, etc.). This domain is now being used within LCS. SPEECHBUILDER has also been used by members of our group, as well as our industrial partners to create small mixed-initiative database access applications (e.g., schedule information, music queries, stock quotes).

In order to allow developers to test out their systems, we have set up a centralized telephone access line, which developers can use to access their deployed domains. In addition, we have provided a local-audio setup so that those developers with access to the GALAXY code distribution can run their domains on their own machines independently of any MIT hardware.

6. Ongoing and Future Activities

There are many ongoing and future activities which will improve the SPEECHBUILDER infrastructure. For example, in order to be able to handle more complicated schema for information access, we plan to introduce the ability to access multiple intersecting database tables from one domain. Currently, all database cells are treated as strings, but in the future we plan to implement functionality for handling Boolean and numerical values as well.

Although we have created an initial database query dialogue manager, we plan to continue to add features which will provide the developer more control for their particular application. A related area of future research will be to improve the current communication protocol for non-database applications.

In areas of speech recognition and understanding, we plan to incorporate our recent work in confidence scoring [11]. In the longer term we would like to introduce unsupervised training of acoustic and linguistic models. Currently, developers can only upload transcriptions, which are used as training data for these components.

The current configuration of the system defaults to using a commercial speech synthesizer. We would like to provide a facility to developers whereby they can configure a simple concatenative speech synthesizer [12].

The work we have done thus far has been restricted to American English. However, as we have ongoing research efforts in multilingual conversational systems (e.g., Japanese, Mandarin Chinese, and Spanish), we have begun to modify the SPEECHBUILDER interface so that multilingual systems can be configured as well.

The systems developed thus far have been prototypes that have not been rigorously evaluated. In order to demonstrate the usefulness of the currently deployed architecture, we plan to configure a weather information domain within SPEECHBUILDER, and evaluate at least the speech recognition and understanding components on our standard test sets [1]. By demonstrating at least compatible performance with our manually created systems, we hope to show the potential for creating robust, yet rapidly configurable, spoken dialogue systems.

7. Summary

This paper has provided an overview of our efforts to facilitate the creation of mixed-initiative spoken dialogue systems. If successful, we believe this research will benefit others by allowing people interested in spoken dialogue systems to rapidly configure applications for their particular interests. We have successfully deployed two distinct configurations of the SPEECHBUILDER utility which connect to applications located on a remote web-based server, or to a local database. Several prototype systems have been created with this utility, which are currently in active use. In the near future, we hope to extend the current framework so as to allow for the creation of more complex and powerful spoken dialogue systems.

8. Acknowledgements

Many people in the SLS group contributed to this work, including Issam Bazzi, Scott Cyphers, Ed Filisko, TJ Hazen, Lee Hetherington, Jef Pearlman, Joe Polifroni, Stephanie Seneff, Chao Wang and Jon Yi.

9. References

[1] V. Zue et al., "JUPITER: A telephone-based conversational interface for weather information," *IEEE Trans. Speech and Audio Proc.*, 8(1), 2000.

[2] S. Seneff and J. Polifroni, "Dialogue management in the MERCURY flight reservation system," *Proc. ANLP-NAACL 2000, Satellite Workshop*, Seattle, 2000.

[3] S. Sutton, et al., "Universal Speech Tools: The CSLU Toolkit," *Proc. ICSLP*, Sydney, 1998.

[4] S. Seneff, R. Lau, and J. Polifroni, "Organization, communication, and control in the GALAXY-II conversational system," *Proc. Eurospeech*, Budapest, 1999.

[5] J. Pearlman, "SLS-LITE: Enabling Spoken Language Systems Design for Non-Experts," M.Eng. Thesis, MIT, Cambridge, 2000.

[6] J. Glass, T. Hazen, and L. Hetherington, "Real-Time Telephone-Based Speech Recognition in the JUPITER Domain," *Proc. ICASSP*, Seattle, 1999.

[7] I. Bazzi and J. Glass, "Modeling out-of-vocabulary words for robust speech recognition," *Proc. ICSLP*, Beijing, 2000.

[8] A. Black, K. Lenzo and V. Pagel, "Issues in building general letter to sound rules," *ESCA Speech Synthesis Workshop*, Jenolan Caves, 1998.

[9] S.Seneff, "TINA: A natural language system for spoken language applications," *Computational Linguistics*, 18(1), 1992.

[10] L. Baptist and S. Seneff, "GENESIS II: A versatile system for language generation in conversational system applications," *Proc. ICSLP*, Beijing, 2000.

[11] T. Hazen, T. Burianek, J. Polifroni, and S. Seneff, "Recognition confidence scoring for use in speech understanding systems," *Proc. ISCA ASR Workshop*, Paris, 2000.

[12] J. Yi, J. Glass, and L. Hetherington, "A flexible, scalable finite-state transducer architecture for corpus-based concatenative speech synthesis," *Proc. ICSLP*, Beijing, 2000.