TOWARDS EVERY-CITIZEN'S SPEECH INTERFACE: AN APPLICATION GENERATOR FOR SPEECH INTERFACES TO DATABASES

Arthur R. Toth, Thomas K. Harris, James Sanders, Stefanie Shriver, Roni Rosenfeld

School of Computer Science
Carnegie Mellon University
usi-requests@cs.cmu.edu

ABSTRACT

One of the acknowledged impediments to the widespread use of speech interfaces is the *portability problem*, namely the considerable amount of labor, data and expertise needed to develop such interfaces in new domains. Under the Universal Speech Interface (USI) project, we have designed unified look-and-feel speech interfaces that employ semi-structured interaction and thus obviate the need for data collection. More importantly, the unified structure of USI-compliant interfaces makes possible the automatic generation of new interfaces from a terse, high-level specification. In this paper, we describe an application generator and accompanying toolkit that allow even non-programmers to generate and use fully functional speech interfaces to their chosen database in less than 15 minutes.

1. INTRODUCTION

One of the acknowledged impediments to the widespread use of speech interfaces is the *portability problem*, namely the considerable amount of labor, expertise and data needed to develop such interfaces in new domains. During the early and mid 1990s, in the aftermath of the ATIS project [1], it was often estimated that several person-years of effort would be needed to build ATIS-like systems. Furthermore, at least some of the people involved in such an effort would have to be seasoned speech researchers with considerable experience in building such systems. Additionally, tens of thousands of utterances would need to be collected in Wizard-of-Oz style experiments in order to create a sufficiently robust grammar.

Subsequently, much of the experience gained in ATIS and similar projects was generalized to new domains, with the result that experienced dialog system developers could build new systems in similarly limited domains with only several person-months of effort. More recently, with the widespread commercial use of directed-dialog systems, toolkits were created that further reduced development time to perhaps several person-weeks. Also, speech researchers were no longer needed to be part of the development effort, as it could be done entirely by speech engineers and speech interface designers.

This level of effort and expertise may be reasonable for commercial development of speech interfaces. However, in order to empower individuals to create their own speech interfaces, the barrier has to be lowered further. In this paper, we describe an application generator and accompanying toolkit that allow even non-programmers to generate fully functional speech interfaces to their chosen database in as little as 15 minutes. We hope that this further lowering of the barrier will result in a large number of speech interfaces being generated by many individuals for many purposes. These could include telephone-based interfaces to useful public data, specialized interfaces for the visually impaired, and even one-time, throw-away telephone-based interfaces to private information for people anticipating difficulties in data communication, such as during foreign travel.

2. THE UNIVERSAL SPEECH INTERFACE (USI) PROJECT

The Universal Speech Interface (USI) (a.k.a. "speech graffiti") project (see http://www.cs.cmu.edu/˜usi [2]) strives to create and test unified look-and-feel speech interfaces that employ semi-structured interaction[1]. A main advantage of this approach is user skill transference across applications: a user trained (for 5 minutes) in any USI-compliant application can be immediately productive in any other compliant application. In addition, the semi-structured interaction increases speech recognition accuracy and obviates the need for domain data collection. Finally, the unified structure of USI-compliant interfaces makes possible the automatic generation of new interfaces from a terse high-level specification, which is the subject of this paper.

Although the USI project seeks a unified design for all application types, our first designs were limited to information-access type applications, more specifically database access. We have gone through several iterations of this design, with the result that it is now relatively stable. We have since moved on to generalize the design to include gadget- and device-control applications, and are planning further generalization to other application types, (e.g. transaction systems and interactive guidance systems). However, these newer designs have not yet been tested and thus the application generator described here is limited to generating database-access applications only.

3. DESIGN CONSTRAINTS

Our goal was to provide a toolkit which allows non-expert devlopers to effortlessly create database-oriented USI-compliant speech interfaces. In solving this problem, we adhered to the following constraints:

- The resulting application should run on a single platform.
- The toolkit should help enforce USI design principles.
- The toolkit should leverage the use of existing software.

[1]For a more general discussion of the USI philosophy, see [3].

- The target developers should not be required to know much about programming or speech processing to create basic working interfaces to their applications.
- The generated application should be extensible by developers who do have knowledge of programming and/or speech processing.

4. USI APPLICATION ARCHITECTURE

The architecture of the generated database-oriented USI application is similar to that described in [2] but has been modified to meet the goal of running on a single platform. The modifications consist primarily of porting parts from Linux to Windows, or in some cases to the Cygwin environment running under Windows (http://www.cygwin.com). The design is still modular, with a text-based component at its core. The text-based component consists of the USI library with an API that enforces the interaction style, and the domain manager which handles application-specific information. The USI library interacts with the Phoenix parser [4]. In the case of a toolkit-generated application, the domain manager interacts with a PostgreSQL database (http://www.postgresql.org). However, the code is currently being generalized to work with Oracle and other ODBC-compliant database packages, and will soon be generalized to plain ASCII files and web tables.

Speech recognition is handled by Sphinx II [5] and speech output is handled by the Festival Speech Synthesizer [6]. A general-purpose diphone voice is included with Festival. Developers have the option of creating higher-quality limited-domain voices based on unit selection. Speech I/O is connected to the text-based component via a Perl script. (It is also possible to connect the text-based component to an existing Visual Basic script to provide additional telephony support, but this is more hardware-specific and is therefore not currently part of the toolkit.)

5. CREATING A NEW USI APPLICATION

A toolkit that creates a USI database application must handle the following tasks:

- Generate code for the domain manager which properly accesses the database.
- Generate a grammar file for the Phoenix Parser which both enforces USI interaction style and is consistent with the database content (e.g. column names, datatypes, named entities in open datatypes).
- Generate a language model and pronunciation dictionary for the Sphinx II speech recognition system which are consistent with the grammar.
- Properly cross-link these various knowledge sources so that multiple generated USI applications do not interfere with each other's operation.

6. BASIC TOOLKIT

In the most general scenario, a user can create a USI application from scratch by manually creating all the files mentioned in the previous section, checking for consistency across the various portions, and performing any necessary debugging. We have done that with a variety of database access applications, such as USI/MovieLine, which provides information about movies playing in the Pittsburgh area, and USI/ApartmentLine, which provides information about available apartments in the Pittsburgh area. We have subsequently separated out portions of the domain manager code that were similar across all these applications, and provided methods that can be called with data specific to a particular system. This allowed us to create the first version of our toolkit.

Although this reduced the amount of work necessary to create a new application, it still required knowledge of programming and many system details. To further simplify the task of generating new applications, we examined the remaining variability across different domains and databases. This turned out to arise from translating from USI-style user queries to SQL queries, and converting the results of the queries back from SQL to the USI style. These mappings varied across domains because they involved logic particular to the domain. We were able to distill this logic into a number of associations between USI concepts and database concepts. These associations, coupled with the database contents, were almost sufficient to specify a complete USI application. The remaining necessary items included a handful of application-level attributes and database attributes (the latter including column names, table names, and table linking information).

7. XML-DRIVEN APPLICATION GENERATION

The next challenge was to determine how to have users specify this necessary information without requiring much knowledge of programming or speech processing. We decided that having a user create an XML document would be a good solution for a number of reasons:

- XML is flexible enough to specify the structure of our system information.
- A variety of editors can be used to create the document.
- XML parsers which validate the document are freely available.

One possibility was to base our approach on VoiceXML (http://www.voicexml.org). We decided not to pursue this approach as it would have required adopting a different system architecture, and it would have been difficult to enforce our interface style in that architecture.

The other option involved creating a Document Type Definition (DTD) to specify the structure of XML documents used for generating USI applications, creating scripts to generate and compile necessary files based on the DTD, and executing and documenting a computing environment for doing this on a devloper's machine.

The items mentioned in the Basic Toolkit section fell into three main categories, which then became elements in our DTD: Application, Database, and Slot. An XML document according to our DTD consists of an Application element followed by a Database element and a variable number of Slot elements. The Application and Database attributes mentioned earlier map directly to XML attributes in our DTD. Most of the items associated with slots also map directly to XML attributes in our DTD, but we found it useful in a few cases to define further elements. These additional elements were associated with slots and belonged to one of two categories: names used to refer to slots and value types associated with particular slots. It was useful to have the names as elements because multiple names could be used with a single slot. Specifying the value types as a set of elements allowed us to add further structure to them, such as enumerations and parameterization. The final DTD was a mere 45 lines.

8. APPLICATION GENERATION

An XML document conforming to our DTD contains all the information needed for generating the application. Once developers create such documents, they can invoke the application generator (AppGen) code, providing as input the XML document as well as the database itself. The AppGen then generates and compiles all the necessary components, as depicted in Figure 1.

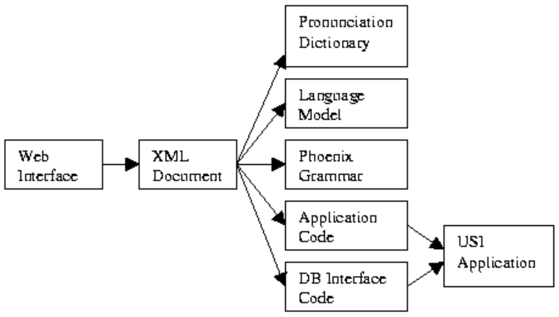


Fig. 1. Application generation process.

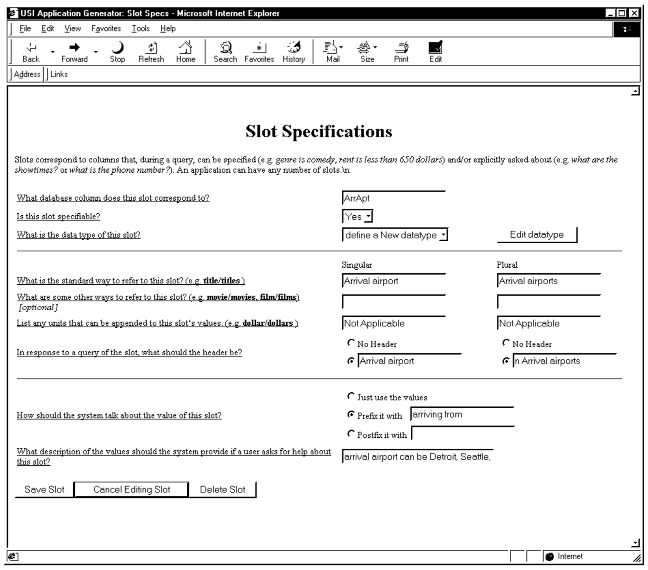
9. A WEB-BASED APPLICATION GENERATOR

After gaining some experience with the XML-based Application Generator, we concluded that the process could be simplified even further. The solution was a Web-based program that lets the developer fill out some forms, and then generates an XML document. The forms to be filled out include application-level information, slot-level information, and new datatype information. Whenever possible, a pulldown menu of choices is offered. The short prompts and questions labelling fields in the forms are hyperlinked to pop-up help windows. An example slot-level page is shown in Figure 2.

Much effort was made to reduce the number of form fields to a minimum, in order to simplify the interaction. As a result, some less commonly used options are not available via the Web interface, but can still be invoked by editing the resulting XML document.

The Web interface has the following advantages:

- It requires the least amount of expertise or computer savvy. Virtually anyone who uses computers knows how to fill forms and select from menus.
- The Web program can check for data consistency and completeness, issuing error messages or confirmation prompts where appropriate.
- If the server is run centrally by the USI development team, design changes, upgrades and bug fixes are immediately available to all users, wherever they may be.


USI Application Generator: Slot Specs - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Size Print Edit
Address Links
Edit datatype
Save Slot
Cancel Editing Slot
Delete Slot
Internet

Slot Specifications

Slots correspond to columns that, during a query, can be specified (e.g. *genre is comedy*, *rent is less than 650 dollars*) and/or explicitly asked about (e.g. *what are the showtimes?* or *what is the phone number?*). An application can have any number of slots.\n

What database column does this slot correspond to?	ArrApt	
Is this slot specifiable?	Yes	
What is the data type of this slot?	define a New datatype	
	Singular	Plural
What is the standard way to refer to this slot? (e.g. **title/titles**)	Arrival airport	Arrival airports
What are some other ways to refer to this slot? (e.g. **movie/movies, film/films**) *[optional]*		
List any units that can be appended to this slot's values. (e.g. **dollar/dollars**)	Not Applicable	Not Applicable
In response to a query of the slot, what should the header be?	No Header [✓ Arrival airport]	No Header [✓ n Arrival airports]
How should the system talk about the value of this slot?	Just use the values [✓ Prefix it with: arriving from] Postfix it with:	
What description of the values should the system provide if a user asks for help about this slot?	arrival airport can be Detroit, Seattle,	

Fig. 2. Example Application generator screen.

The Web Application Generator serves as both a repository for USI applications and as a tool to create new ones. Each user has the ability to either create an application from scratch or to adapt one from a previously created application. In fact, anything created using this generator can be shared freely among users, including whole applications, as well as slot types and even datatypes. This practice maximizes utility and minimizes the time it takes to create new applications.

The Application Generator is designed to minimize the number of questions the average user would need to fully specify a simple to moderately complex database application. Only 10 minutes was required by an experienced user to create an application to access a simple, 8-column database of flight information. In general, new users can expect to spend about an hour to create their first USI application using the Web based Generator; the amount of time decreases considerably with increased usage and levels off to about one to two minutes per slot.

10. SUMMARY AND FUTURE PLANS

We created a toolkit for streamlining the generation of USI-compliant speech interfaces for database access. The need for explicit programming was eliminated by the introduction of an XML-based repository of domain-specific and application-specific declarative knowledge. The process was in turn further simplified by using an interactive Web-based interface to automatically generate such XML documents. This allows even non-programmers to create fully functional speech interfaces in as little as 15 minutes.

We have tested our application generator and are now making it freely available (for further information, send mail to usi-requests@cs.cmu.edu). We plan to offer this tool to students and instructors for incorporation into course projects, at Carnegie Mellon and elsewhere. At ICSLP, we will report on all such experiences and related feedback.

Our ultimate goal for this project is to make *talking* to structured data as easy and as ubiquitous as *typing* at it. Our immediate plans include generalizing the backend interface to ODBC-compliant databases, including ASCII files, and soon thereafter to HTML-based tables. We hope that this latter step will facilitate the automatic “speechification” of Web information. We also plan to extend the application generator to other USI application types, such as gadget- and device-control, and interactive guidance systems.

11. ACKNOWLEDGEMENTS

The USI project is funded in part by the Pittsburgh Digital Greenhouse (http://www.digitalgreenhouse.com). This research was also 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.

12. REFERENCES

[1] Patti J. Price, “Evaluation of spoken language systems: the ATIS domain,” in *Proceedings of the DARPA Speech and Natural Language Workshop*, June 1990.

[2] Ronald Rosenfeld, Xiaojin Zhu, Stefanie Shriver, Arthur Toth, Kevin Lenzo, and Alan W Black, “Towards a universal speech interface,” in *Proceedings of ICSLP ’00*, Oct. 2000.

[3] Ronald Rosenfeld, Dan Olsen, and Alexander Rudnicky, “A universal human-machine speech interface,” Tech. Rep. CMU-CS-00-114, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Mar. 2000.

[4] W. Ward, “The cmu air travel information service: Understanding spontaneous speech,” in *Proceedings of the DARPA Speech and Natural Language Workshop*, June 1990.

[5] X.D. Huang, F. Alleva, H.W. Hon, M.Y. Hwang, K.F. Lee, and R. Rosenfeld, “The sphinx-ii speech recognition system: An overview,” *Computer, Speech and Language*, vol. 2, pp. 137–148, 1993.

[6] Alan W Black, P. Taylor, and R. Cayley, “The festival speech synthesis system,” 1998, http://www.cstr.ed.ac.uk/projects/festival/.