Chapter 1

A FRAMEWORK FOR DEVELOPING CONVERSATIONAL USER INTERFACES

James Glass, Eugene Weinstein, Scott Cyphers, Joseph Polifroni
MIT Computer Science and Artificial Intelligence Laboratory,
Cambridge, MA 02139, USA
{glass, ecoder, cyphers, joe}@csail.mit.edu

Grace Chung
Corporation for National Research Initiatives,
Reston, VA, USA
gchung@cnri.reston.va.us

Mikio Nakano
NTT Corporation,
Atsugi, Japan
nakano@atom.brl.ntt.co.jp

Abstract In this work we report our efforts to facilitate the creation of mixed-initiative conversational interfaces for novice and experienced developers of human language technology. Our focus has been on a framework that allows developers to easily specify the basic concepts of their applications, and rapidly prototype conversational interfaces for a variety of configurations. In this paper we describe the current knowledge representation, the compilation processes for speech understanding, generation, and dialogue turn management, as well as the user interfaces created for novice users and more experienced developers. Finally, we report our experiences with several user groups in which developers used this framework to prototype a variety of conversational interfaces.

Keywords: Conversational interaction, spoken dialogue systems

1. Introduction

In recent years, many sophisticated conversational interfaces have been developed that enable fluent, spoken communication between humans and machines. Such systems are developed by speech and language experts, and require significant effort over a sustained period to achieve good performance. For this reason, non-experts must overcome a significant hurdle to use human language technologies (HLTs) for their own applications. To address this issue, we have been developing a utility (called SPEECHBUILDER), which enables rapid prototyping of spoken dialogue systems by both novice and expert developers. In this paper we motivate the need for this research, describe our approach and progress, and describe several experiments we have performed with novice users creating their own speech-based interfaces.

In the following section, we briefly provide additional background on the current state of directed and mixed-initiative dialogue interaction, and motivate the need for mechanisms to facilitate the development of mixed-initiative conversational interface prototypes. We then describe the approach that we have taken for our research in this area, and give an overview of the user interface we have created. We then describe the speech understanding, generation, and dialogue framework used, and describe several experiments we have conducted with different groups of users. Finally we compare our research to related work, and describe our ongoing research in this area.

1.1 Background

Although all spoken dialogue systems can be considered conversational to some degree, they may be differentiated by the degree with which the system maintains control of the conversation, and the inherent amount of flexibility provided to the user to ask for a) what they want, b) in the way they want to ask for it, and c) when they want to ask it. In the most conservative approach, the computer takes complete control of the interaction. These *directed-dialogue* applications typically require that the user answer a set of prescribed questions, much like the touch-tone implementation of interactive voice response systems. Since the user's options are restricted, completion of such transactions is easier to attain, and it is therefore not surprising that such systems have been the first to be successfully deployed on a wide scale [1–3].

An alternative approach to human-computer interaction is based on the idea of *mixed-initiative* dialogue between the user and the machine. This approach employs a more flexible dialogue strategy that allows both the user and the machine to participate actively to solve a problem interactively using a conversational paradigm. Systems which are built with the mixed-initiative paradigm must typically process more complex queries than their directed-dialogue counterparts [4], and are inherently more difficult to design and deploy. For this reason, the majority of these kinds of systems remain under development in research laboratories [5–9], although some are beginning to be deployed publicly as well [10].

1.2 Motivation

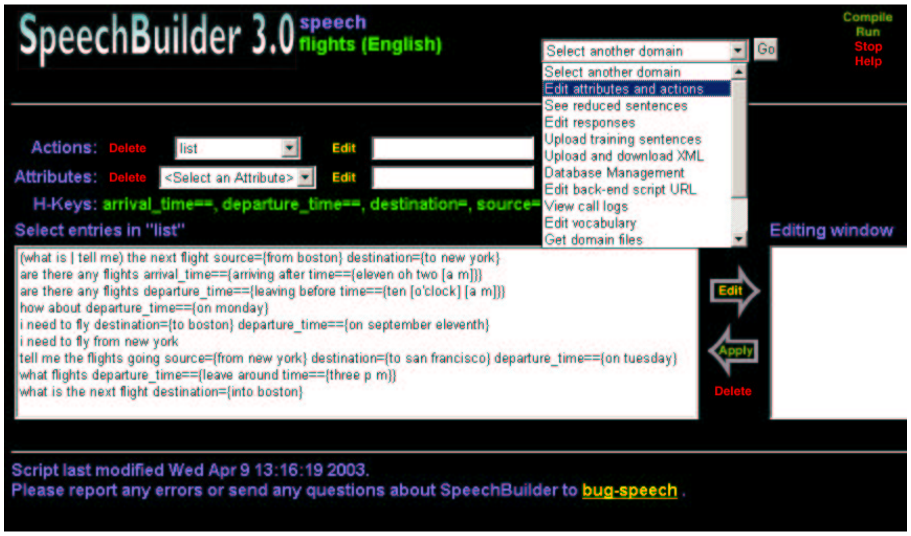
Although mixed-initiative conversational interfaces are a natural and efficient means of communication, there are two fundamental technical barriers which limit their widespread use. First, it is difficult to configure the HLT required to create a prototype system, and second, performance optimization is typically an iterative process that is application specific, and not fully automated. Creating a robust, mixed-initiative conversational interface for a new application area currently requires a tremendous amount of effort from speech and language experts. The development of speech recognition and language understanding technologies is mostly domain and language specific, requiring a large amount of annotated training data. Dialogue management is typically also fine-tuned for the application, often with domain-dependent functionality. System development proceeds iteratively, with prototypes being used to collect data that can then be used for system development, training, and evaluation. This iterative process is crucial to achieve good performance. For example, the initial prototype of a mixed-initiative weather information system trained from several thousand utterances collected from a simulated “wizard-of-oz” scenario saw its error rates more than triple when it was first deployed over the telephone to a wide user population [11]. As utterances were continuously collected, the performance slowly improved to the point where it ultimately exceeded the original laboratory performance. However, this level of performance was only achieved through continuous data collection and system refinement over a period of time.

For conversational interfaces to become as ubiquitous as the telephone, researchers must make it easier for developers to create systems that learn and improve their performance automatically. However, there are many hurdles to even allowing developers to create an initial prototype. For example, we must address the problems of producing a conversational system in a new domain and language given at most a small amount of domain-specific training data. To achieve this goal, we must strive to cleanly separate the algorithmic aspects of the system from the application-specific aspects. We must also develop automatic or semi-automatic methods for acquiring the acoustic models, language models, grammars, and semantic structures for language understanding, and dialogue models required by a new application. The issue of portability spans across different acoustic environments, databases, knowledge domains, and languages. The following section describes the approach we have taken to begin to address some of these challenging issues.

2. Approach

The approach we have adopted is to leverage the basic technology which has been successfully deployed in more sophisticated conversational systems (e.g., [12]). There are many reasons for this. First, we have devoted considerable effort over the last decade to developing HLT to support conversational interaction. 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 users to include both novice and expert developers, since the latter can use the web-interface to rapidly prototype a new domain 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 prototype systems to eventually scale up to the same level of sophistication as our most capable systems. Lastly, by focusing attention on portability, we can identify weaknesses in existing HLT, which can lead to better solutions which can benefit all of our conversational systems.

We have also attempted to use as simple a user interface as possible, while providing mechanisms to incorporate any needed complexities. To accomplish this, we have developed a web-based interface, illustrated in Figure 1.1, that is used by developers to specify information about the nature of the interactions that will take place between a human and a spoken dialogue system. More experienced developers wishing to bypass the web-interface, but still desiring to leverage SPEECHBUILDER to configure HLT components may use a voice



H-Keys: arrival_time==, departure_time==, destination=, source=

Select entries in "list"

```
(what is | tell me) the next flight source={from boston} destination={to new york}
are there any flights arrival_time=={arriving after time=={eleven oh two [a m]}}
are there any flights departure_time=={leaving before time=={ten [o'clock] [a m]}}
how about departure_time=={on monday}
i need to fly destination={to boston} departure_time=={on september eleventh}
i need to fly from new york
tell me the flights going source={from new york} destination={to san francisco} departure_time=={on tuesday}
what flights departure_time=={leave around time=={three p m}}
what is the next flight destination={into boston}
```

Script last modified Wed Apr 9 13:16:19 2003.
Please report any errors or send any questions about SpeechBuilder to bug-speech .

Figure 1.1. The SPEECHBUILDER user interface used to prototype conversational interfaces.

```
<actions>
<request_name> = i would like a restaurant
  | can you (show|give) me a Chinese restaurant in Arlington;
</actions>

<attributes>
  <cuisine> = Chinese|Taiwanese;
  <city> = Washington |  Boston | Arlington;
</attributes>

<discourse>
name masks(city cuisine neighborhood);
</discourse>

<constraints>
<request_name> (city|neighborhood) {prompt_for_city};
</constraints>
```

Figure 1.2. Partial VCFG file for a restaurant query domain.

configuration syntax (VCFG) illustrated in Figure 1.2. To configure understanding, the developer defines semantic concepts, known as *attributes*, and general functions, known as *actions*, that may be invoked by a user in the domain. The developer can also configure system responses and dialogue functionality for their application. This information can be automatically generated from uploaded database tables, or via third-party programs, or entered manually by the developer. All information is stored as a human-readable description (XML) that is compiled to configure the appropriate HLT components.

Once a developer has configured their application domain, they use the web-interface to *compile* it. This process uses the specified information, along with example sentences provided by the developer to configure all necessary HLT components. This process is usually quite rapid (i.e., one or two minutes), although it depends on the domain complexity. Once the domain has been compiled, the developer can examine the resulting grammar, *deploy* the system, talk to it, and subsequently iteratively refine aspects of the understanding, generation, dialogue, etc. Using this interface, a spoken language interface to query database content can thus be created without requiring any programming on the part of a developer. Applications requiring connections to external functionality (e.g. controlling the lights in a house) require the developer to provide code to invoke the external functions.

3. Human Language Technologies

When a user speaks, audio data is sent through a speech recognizer to a natural language understanding component that produces a plausible context-independent semantic representation of what the user spoke. Then context resolution is used to incorporate dialogue history context to resolve unknown references in the sentence. For example, if the user asked,“Is there a cheaper one?" and hotels were being talked about, "hotel" might replace "one" in the semantic representation. Next, the dialogue manager determines how to respond, either by taking an action or asking for additional information. In either case, it generates a semantic representation for a response which a natural language generator converts into the words for the response to the user.

All HLT components use the open source GALAXY architecture [13]. The recognizer, natural language, context resolution, dialogue management, and generation components are all configurable. The SPEECHBUILDER compiler generates the appropriate configuration files for each of these components. SPEECHBUILDER also allows the dialogue management component to invoke arbitrary user code. The following sections describe the configuration of some of these components in greater detail.

3.1 Understanding

The speech recognizer [14] uses generic telephone-based acoustic models, phonetic descriptions of the words in the vocabulary, and an n-gram grammar, which provides likelihoods for sequences of n words, to describe the ways words occur in sentences. The recognizer finds sequences of words that maximize the combined likelihood of the word sequence based on its component n-grams and the likelihood of each individual word given the waveform. The n-gram grammar is derived from the language understanding grammar rules and example sentences provided by the developer, while likelihoods for individual words are based on pronunciations. Baseform word pronunciations can come from large on-line dictionaries, be generated by rule [15], or be provided by the domain developer. We have incorporated an out-of-vocabulary model to handle spoken words which are not in the vocabulary [16].

The recognizer produces a ranked list, called an N-best list, of most likely word sequences. Because an n-gram grammar is used, each word sequence is only locally grammatical, i.e. each sub-sequence of n words is likely, but the entire sequence may not be. The natural language component, TINA [17], uses a probabilistic context-free grammar to combine the word sequences in the N-best list into the semantic representation of the most likely parse tree.

Developers do not actually specify a parsing grammar for their application. Instead, a grammar is inferred using example-based specification [18] from *attributes*, which are semantic concepts specified by phrases, and *actions* which are sentences that use attribute phrases. When one phrase of an attribute appears in an action sentence, the generated grammar will permit any other phrase for the same attribute. For example, in Figure 1.2, the "cuisine" attribute can be "Chinese" or "Taiwanese." In the "request_name" action, only "Chinese" is listed, but the user could ask about any other cuisine. The actual phrase used in a spoken sentence will be associated with "cuisine" in the semantic representation. The generated grammar provides a complete parse where possible, and backs off to concept spotting when a complete parse is not found. Although this mechanism works fairly well, the generated parsing grammar is very simple compared to those that are written by experts, and does not allow domains to reuse sub-grammars for concepts such as times, dates and prices. We have begun to address this issue by allowing developers to incorporate sub-grammars catering to common semantic concepts such as dates and times. We have also developed a new process that converts these concepts into a standard semantic representation.

Figure 1.2 also shows an example of configuring the context resolution component [19]. The "discourse" (context resolution) section states that if a name is specified, then the city, cuisine, and neighborhood should not be inherited from the dialogue history context into the semantic representation of the sentence.

3.2 Dialogue Management

Developers must be able to configure complete mixed-initiative conversational interfaces. Our initial interface constrained developers with no programming experience to database query applications. Those with programming experience could perform more sophisticated dialogue functions via a remote CGI script. However, these two alternatives clearly limited the ability of inexperienced developers to create the kinds of conversational interfaces that can be created by experts.

To provide for more flexible dialogues, we have been developing a more generic dialogue manager [20]. As part of this work, we have begun to abstract a suite of easily configurable dialogue flow functions from our mixed-initiative dialogue systems using a text-based domain specification format. As illustrated in Figure 1.2, in the "constraints" section, if the user seems to be invoking the "request_name" action and a city or neighborhood has not already been specified, then the dialogue manager should invoke the "prompt_for_city" routine to ask for the city. The generic dialogue manager and functions supporting common semantic concepts have been applied to several new domains.

3.3 Language Generation

The natural language generation process converts semantic representations to text [21]. The most obvious role for generation is to produce a response for the user via a speech synthesizer. These responses can be configured by the developer via the web-interface, by modifying default generation templates generated by the initial compilation process.

The language generation process is also used to generate other internal representations. For CGI-based applications, a generated URL-encoded version of the semantic representation is passed to the remote application via an HTTP GET request. For database query applications, the generation process also formulates an SQL query from a semantic representation of a request. Finally, when the generic dialogue manager is used, the generator is used to create an internal "E-form" representation used by the context resolution and dialogue components. E-forms are a simple semantic representation of the meaning of the query, and can be augmented by the discourse and dialogue components based on the query's context.

3.4 HLT Infrastructure

Over the last few years, we have developed several ways to deploy domains to fit particular needs. Originally, users accessed their domains by a shared telephone line with dialogue processing running on a remote server. We now also offer support for developers who want to run domains on local hardware, and, as part of our research on pervasive computing, there is support for handheld devices [22]. We have made it easier for speech-interfaces to communicate with external applications. This was initially accomplished via HTTP requests which provided an encoded version of the semantic frame to an application running on a web server. To eliminate the need for a web server, and to allow applications to incorporate state information when desired, we provide ways to extend a system with Perl, Python, or Java.

All systems generate log files showing the details of user interactions and individual component input and output. We provide tools to allow domain developers to view the logs as hypertext and listen to recordings of the dialogue. To further ease discourse and dialogue testing and debugging, a system can also be run in *batchmode*. In this mode, previously recorded waveforms, N-best lists, or text input can be sent through the system and the output examined. These capabilities are helpful for developers trying to improve the performance of their initial prototype.

4. Deployment

Over the last few years, we have had numerous experiences with developers using the SpeechBuilder utility to create a wide variety of speech-based applications. To familiarize new developers with the SpeechBuilder system, we have developed an introductory laboratory exercise. Our first experiences were with users interacting with database applications with limited dialogue functionality, or controlling applications via the CGI interface. As the dialogue management component was made available, we extended the laboratory to explore this component in more depth within the context of a hotel information domain. In the current laboratory, students develop a restaurant query system, starting from a database table containing simple attribute-value pairs (i.e., names, addresses, cuisines, etc., plus associated values). An initial recognizer and natural language component are created automatically using the values in the table. Discourse and dialogue components are then configured and modified within the VCFG file; system responses are also modified. The students use both batchmode and a telephone to communicate with their systems.

The laboratories we have developed have been used both locally and remotely. In a recent remote workshop on pervasive computing [23], a class of over 30 researchers created a speech interface to an instant messenger client. The web-based utility has also been used as a laboratory for Computational Linguistics students at Georgetown University, and as part of summer school classes at Johns Hopkins University. Courses have varied from one to three sessions where students with little prior background have learned to build simple restaurant and hotel query applications using the web-based interface.

Finally, as part of a collaboration with speech researchers at NTT, we have been developing a Japanese version of this technology. As part of this work, researchers have built several prototype applications including a bus timetable information system and a weather information system.

5. Related Work

Other research groups have also been attempting to make it easier for non-experts to create new domains. Systems which modularize their dialogue manager try to take advantage of the fact that a dialogue can often be broken down into a set of smaller sub-dialogues (e.g., dates, addresses), in order to make it easier to construct dialogue for a new domain (e.g., [3]). For example, researchers at OGI have developed a rapid development kit for creating spoken dialogue systems, which is freely available, and which has been used by students to create speech-based systems [24]. Starkie et al. describe a spoken dialogue system creation toolkit that is able to infer complex natural language grammars from examples specified by the developer [25]. On the commercial side, there has been a significant effort to develop the Voice eXtensible Markup Language (VoiceXML) as a standard to enable internet content and information access via voice and phone [26]. To date these approaches have been applied only to directed-dialogue strategies. In addition, example-based specification of user interfaces has also been addressed in the literature (e.g. [27]), but this work has focused on visual interfaces, and has not been significantly explored in the area of spoken dialogue.

Additionally, several attempts have been made at simplifying the process of creating dialogue systems to query databases. Toth et al. [28] have created a toolkit to allow a developer to efficiently configure human language technology components around relational database tables. However, the user must first learn keywords to converse with the system. Microsoft English Query is a commercial product that allows a developer to configure a typed natural language interface to database content [29]. The system provides a great deal of flexibility in specifying mappings from linguistic content to database constructs; however, no speech interface is provided, and there is no support for dialogue or discourse. [30] describes a tool similar to English Query in purpose, which achieved better NL-to-SQL translation performance on a corpus of queries from several popular domains.

6. Summary and Ongoing Activities

This paper has summarized our progress in developing a utility to enable rapid prototyping of spoken dialogue systems. However, as was pointed out initially, and as any experienced developer knows, a prototype is only the initial step in the creation of a conversational interface. Creating a high-performance system requires sustained data collection, continuous development, evaluation, and refinement. To help developers achieve this goal will require additional work on unsupervised learning. In our lab, we have begun to improve system performance by processing untranscribed utterances [31], but there is clearly much more research necessary.

The current compilation process configures the speech recognizer and language understanding grammar in parallel, based on the domain description. Recently, we have added the ability to configure our recognizer based on information in the natural language parsing grammar [32]. We plan to integrate this new method into the compiler, to provide increased flexibility to developers in the future. In other areas of research, we have been developing a dynamic vocabulary capability within our speech recognizer and language understanding components [33]. This will give the developer increased flexibility to modify system capabilities during run-time. This work will also allow us to take advantage of more flexible response planning techniques we are developing [34].

Finally, as part of our recent efforts on multimodal interfaces, we have augmented attribute values with timing information to indicate when a concept was spoken. We have used this information to modify a SpeechBuilder-created application, so that it can incorporate pen-based input. Future versions of the SpeechBuilder utility will probably include a connection to our new multimodal component to enable more flexible interactions.

Acknowledgements

This research was supported in part by DARPA under contract N66001-99-8904 monitored through Naval Command, Control and Ocean Surveillance, and contract N66001-00-2-8922, monitored through SPAWAR Systems Center, San Diego, under an industrial consortium supporting the MIT Oxygen Alliance, and by NTT.

References

[1] R. Billi, et al., “Automation of Telecom Italia Directory Assistance Service: Field Trial Results,” *Proc. IVTTA,* 11–16, 1998.

[2] Nuance Communications, http://www.nuance.com

[3] E. Barnard, et al., “A consistent approach to designing spoken-dialog systems,” *Proc. ASRU Workshop,* Keystone, CO, 1999.

[4] V. Zue and J. Glass, “Conversational interfaces: Advances and challenges,” Proceedings of the IEEE, 88(8), 1166–1180, 2000.

[5] A. Rudnicky, et al., “Creating natural dialogs in the Carnegie Mellon Communicator system,” *Proc. Eurospeech,* 1531–1534, 1999.

[6] V. Souvignier, et al., “The thoughtful elephant: Strategies for spoken dialogue systems,” *IEEE Trans. SAP*, 8(1), 51–62, 2000.

[7] J. Allen, et al., “The TRAINS project: A case study in defining a conversational planning agent,” *J. Exper. and Theoretical AI*, 7, 7–48, 1995.

[8] M. Blomberg, et al., “An experimental dialogue system: Waxholm," *Proc. Eurospeech*, 1867-1870, 1993.

[9] S. Rosset, et al., “Design strategies for spoken language dialog systems,” *Proc. Eurospeech,* 1535–1538, 1999.

[10] A. Gorin, G. Riccardi, and J. Wright, “How may I help you?,” *Speech Communication,* 23, 113–127, 1997.

[11] V. Zue et al., “JUPITER: A telephone-based conversational interface for weather information,” *IEEE Trans. SAP,* 8(1), 85–96, 2000.

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

[13] S. Seneff, et al., “GALAXY-II: A reference architecture for conversational system development,” *Proc. ICSLP,* 931–934, 1998.

[14] J. Glass, “A probabilistic framework for segment-based speech recognition,” *Computer, Speech, and Language*, 17, 137-152, 2003.

[15] A. Black, K. Lenzo, and V. Pagel, “Issues in building general letter to sound rules,” *Proc. ESCA Speech Synthesis Workshop*, 1998.

[16] I. Bazzi and J. Glass, “Modeling out-of-vocabulary words for robust speech recognition,” *Proc. ICSLP,* 401–404, Beijing, China, 2000.

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

[18] J. Glass and E. Weinstein, "SpeechBuilder: Facilitating spoken dialogue systems development," *Proc. Eurospeech*, 1335–1338, Aalborg, 2001.

[19] E. Filisko, and S. Seneff, "A context resolution server for the GALAXY conversational systems," *Proc. Eurospeech*, 197–290, Geneva, 2003.

[20] J. Polifroni and G. Chung, "Promoting portability in dialogue management," *Proc. ICSLP*, 2721–2724, Denver, CO, 2002.

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

[22] K. Steele, J. Waterman, and E. Weinstein, "The Oxygen H21 handheld," MIT Lab. for Computer Science Research Summary, March 2003.

[23] MIT Project Oxygen web site, *http://oxygen.lcs.mit.edu*

[24] S. Sutton, et al., "Universal speech tools: The CSLU toolkit," *Proc. ICSLP*, 3221–3224, Sydney, 1998.

[25] B. Starkie, et al., "Lyrebird: Developing spoken dialog systems using examples, " *Proc. ICGI*, 309–311, 2002.

[26] http://www.w3.org/TR/voicexml/

[27] M. Derthick, et al., "Example-Based Generation of Custom Data Analysis Applications," *Proc. IUI*, 57–64, Santa Fe, 2001.

[28] A. Toth, et al., "Towards every-citizen's speech interface: An application generator for speech interfaces to databases," *Proc. ICSLP*, 1497–1500, Denver, CO, 2002.

[29] http://www.microsoft.com/sql/evaluation/features/english.asp

[30] A. M. Popescu, O. Etzioni, and H. Kautz, "Towards a Theory of Natural Language Interfaces to Databases," *Proc. IUI*, Miami, 2003.

[31] M. Nakano and T. Hazen, "Using untranscribed user utterances for improving language models based on confidence scoring," *Proc. Eurospeech*, 417–420, Geneva, 2003.

[32] S. Seneff, et al., "Automatic induction of n-gram language models from a natural language grammar," *Proc. Eurospeech*, 641–644, Geneva, 2003.

[33] J. Schalkwyk, et al., "Speech recognition with dynamic grammars using finite-state transducers," *Proc. Eurospeech*, 1969–1972, Geneva, 2003.

[34] J. Polifroni, et al., "Towards the automatic generation of mixed-initiative dialogue systems from web content," *Proc. Eurospeech*, Geneva, 2003.