"Build Your Own" Spoken Dialogue Systems: Automatically Generating ISU Dialogue Systems from Business User Resources

Oliver Lemon, Xingkun Liu, and **Helen Hastie**
School of Informatics
University of Edinburgh
Informatics Forum
10 Crichton Street
Edinburgh, EH8 9AB
{olemon,xliu4,hhastie}@inf.ed.ac.uk

Abstract

Building effective spoken dialogue systems (SDS) is currently a complex task requiring expert knowledge. Our tools give control of SDS application development to non-experts, who need only use a Graphical User Interface or GUI to develop state-of-the-art "Information State Update" (ISU) dialogue systems. Behind the GUI is a set of Advanced Dialogue Tools (ADT) that generate complete SDS based on Business User Resources. These resources include a database and a Process Model that captures the structure of an application, for example, banking or restaurant information. Also generated are speech recognition Language Models and grammars for robust interpretation of spontaneous speech. We will demonstrate how our simple GUI allows developers to easily and quickly create and modify SDS without the need for expensive speech application service providers. This demonstration shows the interface, the ADT components, and discusses some of the research issues involved. We also show an example application built with the tools: a tourist information system running on an ultra-mobile PC.

1 Introduction

As automated call centres are becoming more and more commonplace, new challenges are emerging such as having to rely on expensive service providers to build systems, the inability to quickly and easily modify live systems, and the time and cost needed to create new SDS applications. This paper describes a solution to these problems using our Advanced Dialogue Tools (ADT). This prototype system allows developers to take already established business user resources such as Business Process Models (BPM) and databases, and use them to automatically generate spoken dialogue systems. Simple customisations can then be made through the easy-to-use ADT interface or GUI, which is example-driven. This radically new way of creating spoken dialogue systems will put control into the hands of the business user who is familiar with customer needs and business goals, thus improving usability and making spoken dialogue systems more widely and rapidly available.

Currently, VoiceXML is widely used for such tasks. However, VoiceXML applications are difficult to build and maintain, because developers must anticipate and represent every possible dialogue path in the finite-state VoiceXML model. ADT will generate VoiceXML dynamically, but the easy-to-use interface allows developers to select, deploy, and monitor different advanced dialogue strategies without needing to code VoiceXML directly. We apply the "Information State Update" (ISU) approach (Lemon, 2004) that enables more robust, flexible and natural conversations than VoiceXML. ISU uses a more concise and maintainable representation of dialogue flow, based on rules operating over dialogue contexts, which can generalise to unforeseen states.

2 The ADT Architecture

Figure 1 shows the ADT architecture whereby the main algorithm takes business user resources and databases as input and uses these to automatically

generate the spoken dialogue system. Figure 2 shows part of one such resource, namely a BPM for hotel bookings. First the caller will hear an introduction, then they will be asked what price range they want, and then whether they want a hotel in the centre of town or not. Advantages of using BPMs include the fact that graphical interfaces and authoring environments are widely available for them, for example: Eclipse, IBM Websphere-Process Server, BEA WeblogicWorkshop etc.. In addition, Business User Resources can contain a lot of additional information as well as call flows including context, multi-media, and multiple customer interactions.

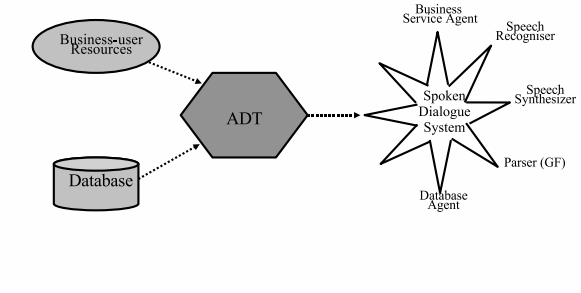

Figure 1: The ADT Architecture

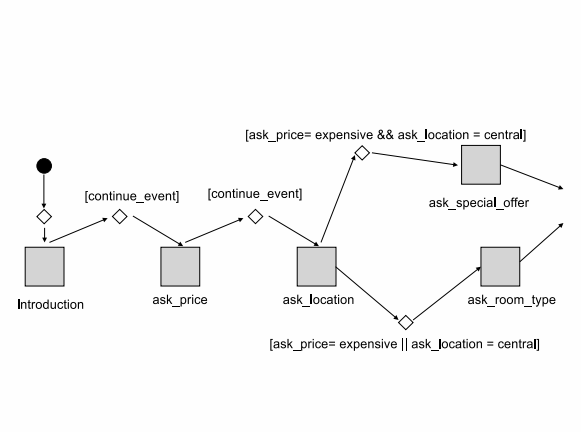

Figure 2: Part of an example Business Process Model for searching for Hotels

The resulting spoken dialogue system deploys the following main modules:

- Speech Recogniser module, e.g. ATK/HTK (Young, 2007; Young, 1995) or Nuance (Nuance, 2002)
- Spoken Language Understanding module, e.g. Grammatical Framework (GF) parser (Ranta, 2004)
- BPM and Database modules
- Speech synthesiser e.g. Festival (Taylor et al., 1998) or Cereproc (Aylett and Pidcock, 2007)

2.1 Generic Dialogue Modelling

Sophisticated research systems have been developed only for specific applications and cannot be easily transferred to another, even very similar task or domain. The problem of components being domain specific is especially prevalent in the core area of dialogue management. For example MIT's Pegasus and Mercury systems (Seneff, 2002) have dialogue managers (DM) that use approximately 350 domain-specific hand-coded rules each. The sheer amount of labour required to construct systems prevents them from being more widely and rapidly deployed. Our solution uses BPMs and related authoring tools to specify *domain-specific* dialogue interactions which are combined with a *domain-general* dialogue manager. Specifically, the DM consults the BPM to determine what task-based steps to take next, such as asking for a cinema name. General aspects of dialogue, such as confirmation and clarification strategies, are handled by the domain-general DM. Values for constraints on transitions and branching in the BPM, for example "present insurance option if the user is business-class", are compiled into domain-specific parts of the DM's update rules. XML format is used for BPMs, and they are compiled into finite state machines consulted by the spoken dialogue system through the BPM module. The domain-general DM was mostly abstracted from the TALK system (Lemon et al., 2006).

2.2 Compiling Grammars for Business User Resources and Databases

For Spoken Language Understanding, ADT currently uses Grammatical Framework (GF) (Ranta, 2004) which is a language for writing multilingual grammars, on top of which various applications such as machine translation and human-machine interaction have been built. A GF grammar not only defines syntactic well-formedness, but also semantic content.

Using ADT, system developers do not have to write a single line of GF grammar code. The sys-

tem compiles all database entries and their properties into the appropriate "slot-filling" parts of the GF grammar for each specific BPM.

For example, a generated GF rule is:

```
Bpm_generalTypeRule_4:
town_info_hotels_name->Utt=>{ s = np.s}
```

This rule was generated because "name" is a database field for the subtask `hotels` in the "town_info" BPM. It specifies that all hotel names are valid utterances.

A core GF grammar has been developed to cover basic information-seeking interactions. This is combined with a domain-specific grammar which is automatically generated from the BPM, database and the example utterances provided by the developer in the GUI. Finally, GF is a robust parser – it skips all disfluencies and unknown words to produce an interpretation of the user input if one exists.

2.3 Speech Recognition and Text To Speech

The grammars for Spoken Language Understanding generated by ADT are also compiled to grammar-based language models (LM) for speech recognition. ADT is plug-and-play and adheres to industry standards such as GSML, GrXML. This allows for greater flexibility since the application developer is not tied to one recogniser or TTS engine. For this demonstration, the speech recogniser is ATK (Young, 2007; Young, 1995) and the speech synthesiser is Cereproc (Aylett and Pidcock, 2007). Future work will involve automatically generating context sensitive language models (Lemon and Gruenstein, 2004).

2.4 ADT GUI

As mentioned above, the prototype ADT GUI can be used to define system prompts and add likely user responses to the grammar. Figure 3 shows the developer associating "spotter" phrases with subtasks in the BPM. Here the developer is associating the phrases "hotels, hotel, stay, room, night, sleep" and "rooms" with the hotels task. This means that, for example, if the user says "I need a place to stay", the hotel-booking BPM will be triggered. Note that multi-word phrases may also be defined. The defined spotters are automatically compiled into the GF grammar for parsing and speech recognition. By default all the lexical entries for answer-types for the subtasks will already be present as

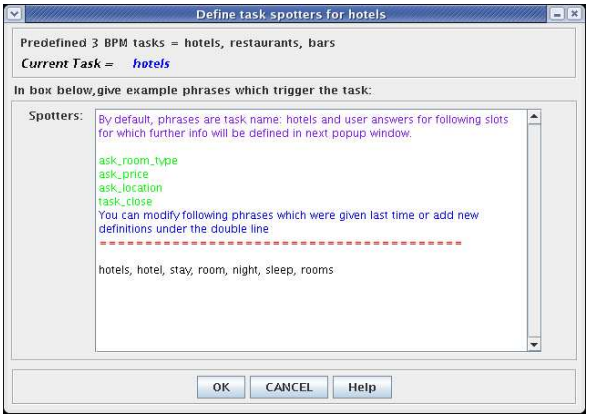

Figure 3: Example: using the ADT GUI to define "spotter" phrases for different BPM subtasks

spotter phrases. ADT also checks for possible ambiguities, for example whether "pizza" is a spotter for both cuisine_type for a restaurant task and food_type for a shopping task, and it uses clarification sub-dialogues to resolve them at runtime.

Figure 4 shows the developer's overview of the subtasks of a BPM, in this case hotel information. The developer can navigate this representation and edit it to define prompts and manipulate the associated databases.

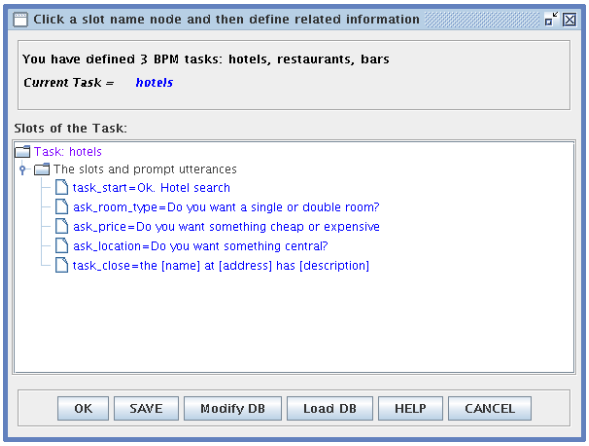

Figure 4: Sub-dialogue structure generated from the Hotel booking BPM

Figure 5 shows the developer specifying the required linguistic information to automate the ask_price subtask of the hotels BPM. Here the developer specifies the system prompt for the information "Do you want something cheap or expensive?"; a phrase for implicit confirmation of provided values "a [X] hotel", where [X] is the semantics of the speech recognition hypothesis for the user input; and a clarifying phrase for this subtask

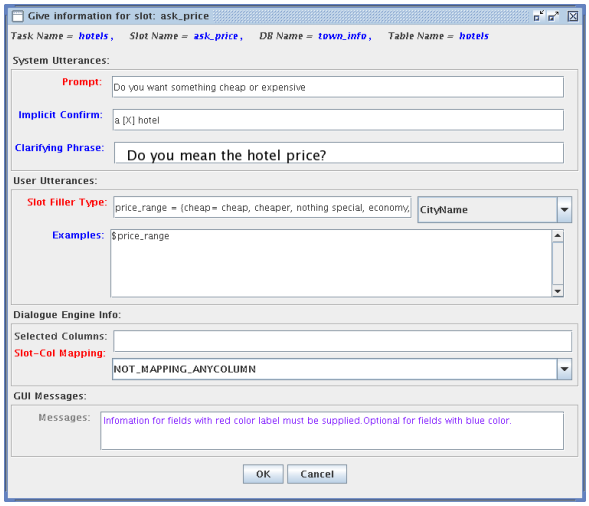

Figure 5: Example: using ADT to define prompts, answer sets, and database mappings for the ask_price subtask of the BPM in Figure 4

"Do you mean the hotel price?" for use when disambiguating between two or more tasks. The developer also specifies here the answer type that will resolve the system prompt. There are many predefined answer-types extracted from the databases associated with the BPM, and the developer can select and/or edit these. Optionally, they can give additional example phrases that users might say to answer the prompt, and these are automatically added to the GF grammar.

2.5 Usability

Several demonstration systems have been built using ADT with an average development time of under an hour. However, our planned evaluation will test the ability of novice users, with some knowledge of BPMs and databases, to iteratively develop their own ISU dialogue systems.

3 Summary

This paper describes the Advanced Dialogue Tools for creating Information State Update based dialogue systems automatically from Business User Resources such as BPMs and databases. The tools include automatic generation of grammars for robust interpretation of spontaneous speech, and uses the application databases and BPMs to generate lexical entries and grammar rules for speech recognition language modelling. We also demonstrate an easy-to-use prototype interface that allows the user to easily and quickly modify aspects of the dialogue, thus eliminating the need for third party service providers. This paper describes ADT, its main components, and some of the research issues involved in its development.

4 Acknowledgement

This project is funded by a Scottish Enterprise Proof of Concept Grant (project number 8-ELM-004).

References

Aylett, Matthew P. and Christopher J. Pidcock. 2007. The cerevoice characterful speech synthesiser sdk. In *AISB*, pages 174–8.

Lemon, Oliver and Alexander Gruenstein. 2004. Multithreaded context for robust conversational interfaces: context-sensitive speech recognition and interpretation of corrective fragments. *ACM Transactions on Computer-Human Interaction (ACM TOCHI)*, 11(3):241– 267.

Lemon, Oliver, Kallirroi Georgila, James Henderson, and Matthew Stuttle. 2006. An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the TALK in-car system. In *Proceedings of EACL*, pages 119–122.

Lemon, Oliver. 2004. Context-sensitive speech recognition in Information-State Update dialogue systems: results for the grammar switching approach. In *Proceedings of the 8th Workshop on the Semantics and Pragmatics of Dialogue, CATALOG'04*, pages 49–55.

Nuance, 2002. http://www.nuance.com. As of 1 Feb 2002.

Ranta, A. 2004. Grammatical framework. a type-theoretical grammar formalism. *Journal of Functional Programming*, 14(2):145–189.

Seneff, Stephanie. 2002. Response Planning and Generation in the Mercury Flight Reservation System. *Computer Speech and Language*, 16.

Taylor, P., A. Black, and R. Caley. 1998. The architecture of the the Festival speech synthesis system. In *Third International Workshop on Speech Synthesis, Sydney, Australia*.

Young, Steve. 1995. Large vocabulary continuous speech recognition: A review. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 3–28.

Young, Steve. 2007. ATK: An Application Toolkit for HTK, Version 1.6. Technical report, Cambridge University Engineering Department.