Programming by Voice: enhancing adaptivity and robustness of spoken dialogue systems

Kallirroi Georgila and Oliver Lemon
School of Informatics, University of Edinburgh
kgeorgil,olemon@inf.ed.ac.uk

Abstract

This demonstration system allows users to reconfigure dialogue systems by using speech dialogues to build simple programs for devices and services. This is a novel type of adaptivity – where the user is able to explicitly adapt some aspects of the dialogue system to their own needs, and is a capability beyond any commercially deployed systems. The main idea is to extend command-based and information-seeking dialogue systems so that users can reconfigure them to perform common tasks, or to behave in specific ways in certain contexts that are of interest to the user. We present a voice-programming (VP) system for device control and information seeking, using the extended in-car "TownInfo" dialogue system (Lemon et al., 2006) developed in the TALK project[1] and built using DIPPER (Bos et al., 2003) and ATK (Young, 2004).

1 Introduction

Most users do not want to learn complex operating instructions for devices and services, so an alternative is to allow them to create their own commands and programs. Users may also wish to configure their devices to carry out specific actions which are tailored to their needs and preferences. One way of doing this is to allow users a level of "programming" access to the interfaces themselves.

For example, in an automated home environment, by uttering a simple phrase such as "I want to relax" a user may request the home control system to perform a number of pre-defined tasks such as turning down the lights, playing classical music, and switching the telephone off. In a similar way, users could call their house when away and define external-event-driven programs such as "Only turn the heating on if the temperature falls below 10 degrees". Similar scenarios can be imagined for in-car device-control dialogues, for example "'I need some peace' means turn the stereo off and close the windows", or "Open the sun roof if the temperature goes above 20".

The idea of voice programming for services is similar – it is much faster, easier, and more robust for the user of a tourist information service to say, for example "show me my favourites" than "show me all expensive French restaurants in the centre of the city".

1.1 Improved robustness

Voice programming is not only a matter of convenience and efficiency for the user but also leads to potential robustness gains. Considering speech recognition limitations, especially in noisy environments such as cars, shorter and more precise commands will in general lead to fewer errors and increase overall user satisfaction. Likewise, if users can define the semantics of their utterances through voice programs, fewer clarifications and confirmations will be required in dialogues.

1.2 Related work

The Metafor project (Liu and Lieberman, 2005) explored the idea of using descriptions in natural language as a representation for programs (Python code). Metafor does not convert arbitrary English to fully specified code, but uses a reasonably expressive subset of English as a visualization tool. Simple descriptions of program objects and their behaviour generate scaffolding (underspecified) code fragments, that can be used as feedback for the designer. In contrast, our system allows users to generate fully working programs via speech dialogues alone.

[1] http://www.talk-project.org

brandial 2006: Proceedings of the 10th Workshop on the Semantics and Pragmatics of Dialogue, pages 199–200.

2 The demonstration system

The current system focuses on controlling devices and services using programs which are:

- activated by speech commands or environmental events
- defined by the user via speech dialogues.

The basic system that we will demonstrate shows programming by voice of macros and conditionals for a tourist information service, and uses the ATK speech recogniser (Young, 2004) and DIPPER dialogue manager (Bos et al., 2003).

The capabilities of the demonstration system[2] are implemented by extending the Information State definitions with fields for macro and conditional names, which can take appropriate arguments (sequences of commands and/or slot values), and adding update rules for interpreting and processing voice programming utterances. In addition, we compiled a language model for voice programming from a GF grammar (Ranta, 2004), and extended the system's parser.

2.1 Defining Macros

A macro is a way for the user to automate a complex task that he/she performs repeatedly or on a regular basis. It is a series of commands or information slots that can be stored and run/accessed whenever the user needs to perform the task. The user can record or build a macro, and then play the macro to automatically activate the series of actions.

The syntax for a macro is:

```
macro_name = slotValue/command_1 and ...
slotValue/command_N
```

In the tourist-information service demo a supported macro is: "When I say 'romantic dinner' I mean an expensive Italian restaurant in the town centre".

After a Wizard-of-Oz data collection for voice programming dialogues, we have extended the coverage of our system (Lemon et al., 2006) to interpret some types of user utterances as macro definitions.

For example user inputs such as:

- When/If/Whenever I say T, it means/I mean X_1 ... X_n
- X_1 ... X_n when/if/whenever I say T

are interpreted as defining a macro with trigger phrase T and which stands for commands/information slot values X_1 ... X_n.

[2]Macros and Conditionals are functional at the time of writing, and we expect Loops and Iteration to be supported by the time of the conference.

Note however that in terms of the dialogue context, the effect of "X_1 ... X_n" is not the same as if the user had actually uttered the individual X_i. For example the salient NPs in each X_i are not available for anaphoric reference. Exactly what the effects on the dialogue context should be is a matter for ongoing research.

The previous example ('romantic dinner') is stored in the information state as a list. When the user utters the macro name, the system will retrieve the associated slots with their values and try to satisfy the user's request.

```
[macro, 'romantic dinner', restaurant,
[[price_range],[food_type],[location]],
[[expensive],[italian],[central]] ]
```

2.2 Defining Conditionals

The syntax for conditionals is: `if condition=true then slotValue/command_1 ...slotValue/command_N` or `execute macro_name`

A typical example of a conditional for programming services in the demo system is: "When I ask for pizza make it expensive". This conditional is stored in the information state as follows:

```
[cond, restaurant, [food_type], [pizza],
[[price_range]], [[expensive]] ]
```

3 Summary

We demonstrate a novel dialogue system for Programming by Voice which leads to enhanced adaptivity and robustness of spoken dialogue systems.

References

Johan Bos, Ewan Klein, Oliver Lemon, and Tetsushi Oka. 2003. DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture. In *4th SIGdial Workshop on Discourse and Dialogue*, pages 115–124, Sapporo.

Oliver Lemon, 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*.

Hugo Liu and Henry Lieberman. 2005. Metafor: Visualizing stories as code. In *Proceedings of the ACM International Conference on Intelligent User Interfaces, IUI*. ACM.

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

Steve Young, 2004. *ATK: An Application Toolkit for HTK, version 1.4*.