Conversational Dialogue Management in the FASiL project

Kerry Robinson, David Horowitz, Emilio Bobadilla, Mark Lascelles, Ana Suarez
Vox Generation Limited, 8 Duncannon Street, London, England.
{krobinson, dhorowitz, ebobadilla, mlascelles, asuarez}@voxgeneration.com

Abstract

The FASiL dialogue manager is described in the context of the commercial deployment of conversational interfaces. A practical dialogue model is presented that uses a list-like structure to manage mixed-initiative dialogue using highly modularised, and independently specified dialogue components.

1 Introduction

In the context of the FASiL project[1] we investigate dialogue management for conversational user interfaces. We are developing a virtual personal assistant (VPA) application that provides a user with access to personal information like email. This paper describes the motivation and design of the FASiL dialogue manager (DM). The next section surveys models of dialogue management that have informed our design decisions. Section 3 discusses the particular requirements for the commercial deployment of dialogue systems and the objectives of FASiL in this regard. Section 4 describes the FASiL DM including a step-by-step analysis of an example dialogue. Section 5 explains the distributed execution model.

2 Models of Dialogue Management

The basic role of the DM in a spoken user interface can be reduced to two basic actions (Levin et al., 1999):

- Interpret observations (usually of user input) in context, and update the internal representation of the dialogue state.
- Determine the next action of the dialogue system according to some dialogue policy.

While these steps are in common with all dialogue managers, each step is non-trivial and as such, has lead to a proliferation of different approaches. In the design of the FASiL DM we have drawn features from several different dialogue management paradigms. These are briefly reviewed in the following section.

[1] FASiL: Flexible Adaptive Spoken Language and Multi-Modal Interface.

2.1 Finite State Models

Some of the first models to be used for dialogue management in spoken dialogue systems were based on Finite State networks (e.g. McTear, 1998). Each node in the network represents a state of the system, and determines the prompts and grammar that are used at that point in the dialogue. The result of user input is to transition to a different node. This paradigm begins to become unwieldy when the number of possible states and transitions becomes as large as in the VPA application, but we adopt the use of transition networks for their transparency in specifying the dialogue policy.

2.2 Form Filling

Approaches based on form filling have been used to support mixed initiative dialogue, where the user does not have to respond to the system prompts directly, but can give different or additional information. Form-based approaches (Goddeau, 1996) are characterised by a data-structure that allows developers to specify a set of slots that must be filled by user-input, and a set of prompts to elicit the values for the required slots. The dialogue policy is usually determined by a general algorithm that seeks to complete the form by prompting the user for unfilled slots. We avoid the use of a general algorithm to define the dialogue policy, but draw upon the concept of a form in maintaining the dialogue state.

2.3 State of the Art Approaches

Many of the more sophisticated approaches to dialogue management can be characterised by a separation between the data structure that stores the current state of the dialogue, and the specification of the dialogue policy. This is the case with the FASiL DM. Differences arise in the way in which the dialogue state is structured, and the way the dialogue policy is specified. Some use rules with pre-conditions that depend on the dialogue state, and actions that determine the next dialogue step (Seneff, 1997; Traum and Larsson, 2003). Others use transition networks to specify the dialogue policy (Lemon et al., 2001; Catizone et al., 2003). A structure may be adopted that allows the dialogue management task to be broken down into sub-tasks, perhaps recursively (e.g. Wang and Lin, 2000)

In spite of the myriad different configurations, Catizone et al. (2003) note that there are many similarities

between modern dialogue systems, and suggest that all consist of a subset of basic functional components. If this is the case, then perhaps the major philosophical differences exist in decisions around the programming methodology, architecture and communications protocols. While these aspects of system design may not directly influence the performance of a system in the lab, they are key to the practical performance of deployed systems. Issues of scalability, robustness, maintainability, and the availability of suitably skilled development and design professionals directly affect the quality of systems that can be realistically deployed.

3 Commercial Systems

There is a significant gap between the level of sophistication of commercially deployed spoken dialogue systems and the impressive technology demonstrators that have been developed by academic and other research institutions. Of course, this is partly due to the immaturity of some components: the robustness and scalability requirements of a commercial product are significantly more stringent than for a technology demonstrator, but we still believe that there is a potential for the widespread deployment of more conversational systems. In the FASiL project, we have tried to identify factors that contribute to the gap, and seek to address them in the design and development of the DM:

- In support of rapid development, maintainability and extensibility we modularize the dialogue development: dividing applications into tasks, and tasks into commands that can be independently authored and maintained.
- To allow potential clients to capitalize on their existing investment in voice infrastructure, we use VXML as the interface with the ASR and TTS resources, and have in mind support, and easy porting to other emerging standards.
- To capitalise on investments in dialogue systems, a key aim of the project is to develop technology that supports straightforward localisation.

4 The Dialogue Manager

We wanted our DM to support dialogues where the user could take initiative at any time, so expert users could dictate the flow of conversation and all users could interrupt and correct recognition and understanding errors. At the same time a system-driven backbone must be provided to guide new users through the system. We also had in mind our objectives related to the commercial deployment of such systems.

Applications are divided into several independent tasks that are completed by executing commands derived from users' speech. Users can expedite task completion by speaking in longer sentences that imply several commands, but the tasks and commands are still implemented independently of one another, allowing several developers to work on the system. To support a mixed initiative interaction with independently defined commands, we introduce the concept of a User Intention Set (UIS): a list structure that manages commands that are currently being processed. Kronenberg and Regel-Brietzman (2001) use a similar approach to manage reusable dialogue modules, while Catizone et al. (2003) use a modified stack structure to keep track of the conversation. Stack structures are also used by Traum and Larsson (2003), to keep track of 'Issues' and 'Questions Under Discussion', but there is a closer similarity with the stack of 'discourse obligations' described by Traum and Allen (1994).

4.1 Task and Command Frames

The dialogue state is represented using frames. Task frames deal with the current task, like sending an email and store information that is agreed with the user, like the recipient list or the priority of the mail. Command frames deal with actions that a user takes toward completing a task, like adding a recipient to an email. The user issues commands that operate on the task frame until a task is complete.

4.2 Specifying the Dialogue Policy

In contrast to form-based approaches, our frames do not imply anything about the dialogue policy. Instead of a generic algorithm, we specify separate dialogue policies for each task-frame, and for every command. This introduces a small overhead in the implementation of simpler dialogues, but the designer has absolute freedom in the specification of the strategy. Still, if there is an opportunity for generic behaviour, policies can be reused (as we do for the add, copy, blind copy and delete commands in the VPA). In common with Catizone et al. (2003) we choose to specify dialogue policies using transition networks instead of rules (Seneff, 1997). We feel this supports a more transparent specification, aiding development, maintenance and quality assurance.

4.3 Dialogue walkthrough

This section describes the processing behind the example interaction in Figure 1. We choose the send email task for simplicity and begin with the first system turn.

The system decides what step to take by traversing the network describing the policy for the send-email task. Task policies consist of three main branches. The first deals with task completion: if there is sufficient information to complete the task (including any necessary confirmation from the user) then the system will carry out the required action, for example, sending the email. The second branch deals with user-initiative: if there are any commands from the user that have not yet

been dealt with, these are given priority. The third branch deals with system-initiative.

At the beginning of the example interaction, the send-email task-frame is empty (so the email cannot be sent), and there are no commands from the user, so the system must take the initiative. In the send-email task, if there are no recipients, a traversal of the dialogue policy network will end up at a system initiative node that asks: "Who would you like to send the email to?" (Figure 1, turn S1). To which the user may respond with U1: "Anna Brown and copy David Smith". The semantic interpretation of the utterance is mapped to one or more new commands. In this case a 'To' command and a 'CC' command.

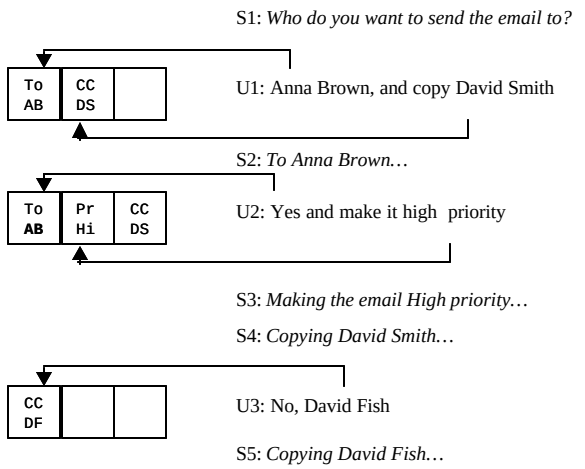

Figure 1

Commands are created and completed by mapping functions. Context-dependent mapping functions are associated with system actions, and allow the system to interpret user-input in the context of the previous system turn. The first command in the example above is the result of context-dependent mapping. The user only gave a name, but the context dependent mapping is used to infer, and generate the appropriate command ('To') and command content (the name Anna Brown, abbreviated to AB in the figure). There are also generic mapping functions that map any remaining user-input to the appropriate command. The 'CC' command in the above example is the product of a generic mapping function.

Commands derived from user input are stored in a list-like structure called the User-Intention Set (UIS). The UIS stores all commands that have been created from user-input, but have not yet been executed. The boxes on the left hand side of figure 1 show the UIS at various points in the example dialogue.

After an utterance is mapped into the intention set, the system must again decide the best next step in the dialogue. Continuing with the example, the preconditions for sending an email are still not met, so the first branch of the dialogue policy is ignored, but this time the UIS contains commands that have not yet been dealt with. There are two pending commands: a 'To' command and a 'CC' command. Commands are dealt with one at a time, so the system selects the first one from the UIS and delegates control to the sub-network specifying the dialogue policy for that command.

The dialogue policy for a command frame is specified in the same way as for a task frame. In the case of the 'To' command, a traversal of the policy network ends in a node that requests a confirmation from the user (S2). The user may stay silent, implicitly confirming the command, or respond with an explicit confirmation and additional commands, as in U2. The confirmation is mapped into the command at the front of the UIS (shown in figure 1 as an emboldened abbreviation of the name) and the additional command is inserted after it. The policy network for the 'To' command is again traversed. This time the preconditions for command execution are met (there are one or more confirmed recipients), so the command 'executes', with the effect that the recipient Anna Brown is added to the send-email task-frame, and the command is removed from the UIS.

The system then proceeds to complete and execute the remaining commands in the UIS by implicitly confirming their contents (S3, S4). The user remains silent for the first confirmation, but interrupts with a correction on the second (U3). The correction is mapped into the in-focus command, and the system attempts confirmation again (S5).

Command policies don't just deal with confirmation. They can define command completion and disambiguation behaviour as well. For example, if a user just gives the command "copy someone", a traversal of the CC command policy results in a system initiative prompt: "Who do you want to copy?"

4.4 Managing the User Intention Set

The UIS consists of a list of command frames that have not yet been executed. The command at the head of the list is always the next to be dealt with, and when the user gives commands, they are added to the beginning of the list. In common with the stack of discourse obligations employed by Traum and Allen (1994), this mechanism ensures that the DM always responds to the user's most recent utterance (or system obligation) first. This is necessary for the system to appear responsive to user initiative, but it means the system must sometimes return to topics that were raised earlier in the dialogue (like the confirmation in S4).

A single utterance may consist of several commands. They are stored in the UIS in the order that they are said, so that they will be dealt with in the same order. We don't currently deal with utterances like "do X, but first do Y", preferring instead to guide the user into a more straightforward mode of communication.

5 Distributed DM Architecture

We adopt a distributed approach to DM based on the Model, View, Controller (MVC) pattern (Krasner and Pope, 1988), and similar to Komatani et al. (2003). The VXML browser takes responsibility for managing simpler aspects of the interaction, while more complex information processing is the responsibility of the FASiL DM. This allows us to conveniently factor out some generic aspects of the dialogue implementation, leading to a simpler dialogue specification, while the use of open-standards like VXML to interact with external components (telephony, ASR, TTS) supports portability between different platforms. The Open Speech Application Framework (Carpenter 2002), developed by a member of the FASiL consortium provides the MVC pattern, and recursive transition networks.

When a user calls the system, the browser makes a request to the DM, which dynamically generates and returns a page of VXML. The user interacts with the VXML page until a valid end-point is reached. The result of the interaction is then communicated back to the DM as part of the request for a new page.

The logic of the VXML page extends only to interpreting and responding to miss-recognition, silence and requests for help from the user. As soon as meaningful input is received, control is passed to the FASiL DM.

5.1 View Generator

Every VXML page is created by the view-generator, which is implemented as a Java Server Page (JSP). The view generator receives at least two parameters as part of the request for a new page – a state-type, and a state number. The state-type defines how the VXML page handles conditions such as no-input (user stays silent), no-match (miss-recognition) and requests for help. Several state types are defined covering common prompting patterns that can be re-used in any application. The state number references a list of prompts defined in a spreadsheet. A spreadsheet macro compiles a JavaScript file that is used in the VXML page created by the view generator. The separation of the prompt definitions into a separate spreadsheet allows them to be easily managed by UI designers. They can also be used to generate scripts for prompt recording, and support easy translation into different languages. In FASiL we used the spreadsheets to support translation of the prompts from English into Swedish and Portuguese.

6 Conclusion

We have described the motivation for, and design of the FASiL dialogue manager. In particular, we described a dialogue model based on frames, transition networks and a list structure called the UIS. At the time of writing we are preparing user-trials through which we hope to analyse the performance of the system, and derive corpora and usability feedback that will inform further development of the technology.

This research was partially funded by the European Commission under the FASiL project, contract number: IST-2001-38685.

7 References

B. Carpenter, S. Caskey, K. Dayanidhi, C. Drouin, R. Pieraccini. 2002. *A Portable, Server-Side Dialog Framework for VoiceXML*. In Proceedings, ICSLP 2002

R. Catizone, A. Setzer, Y. Wilks. 2003. *Multimodal Dialogue Management in the COMIC Project.* In Proceedings, EACL 2003

D. Goddeau, H. Meng, J. Polifroni, S. Seneff, S. Busayapongchaiy. 1996. *A Form-Based Dialogue Manager For Spoken Language Applications.* In Proceedings, ICSLP 1996

K. Komatani, F. Adachi, S. Ueno, T. Kawahara, H. G. Okuno. 2003. *Flexible Spoken Dialogue System based on User Models and Dynamic Generation of VoiceXML Scripts.* In Proceedings SIGdial 2003.

G. E. Krasner and S. T. Pope. 1988. *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*. Journal of Object-Oriented Programming, 1(3):26-49, August/September 1988.

S. Kronenberg and P. Regel-Brietzman. 2001. *Bridging the gap between mixed initiative dialogues and reusable sub-dialogues*. In Proceedings, ASRU 2001

O. Lemon, A. Bracy, A. Gruenstein, S. Peters. 2001. *The WITAS Multi-Modal Dialogue System I.* In Proceedings, Eurospeech 2001

E.Levin, R.Pieraccini, W. Eckert, P. Di Fabbrizio, S. Narayanan. 1999. *Spoken language dialogue, from theory to practice.* In Proceedings, ASRU 1999

M. McTear. 1998. *Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit*. In Proceedings, ICSLP 1998

S. Seneff. 1997. *Discourse and Dialogue Modelling in the Galaxy System.* In Proceedings, Dialogue System and Discourse Analysis Workshop, Taiwan, 1997.

D. Traum and J. Allen. 1994. *Discourse obligations in dialogue processing.* In Proceedings, ACL 1994

D. Traum and S. Larsson. 2003. *The Information State Approach to Dialogue Management*. To appear in Smith and Kuppevelt (eds.): Current and New Directions in Discourse & Dialogue, Kluwer Academic Publishers.

H. Wang and Y. Lin. 2000. *Goal-oriented Table-driven Design for Dialogue Manager*. In Proceedings, ICSLP2000.