Automatic Generation of Information State Update Dialogue Systems that Dynamically Create Voice XML, as Demonstrated on the iPhone

Helen Hastie, Xingkun Liu and **Oliver Lemon**
School of Informatics
University of Edinburgh
{hhastie,xliu4,olemon}@inf.ed.ac.uk

Abstract

We demonstrate DUDE[1] (Dialogue and Understanding Development Environment), a prototype development environment that automatically generates dialogue systems from business-user resources and databases. These generated spoken dialogue systems (SDS) are then deployed on an industry standard Voice XML platform. Specifically, the deployed system works by dynamically generating context-sensitive Voice XML pages. The dialogue move of each page is determined in real time by the dialogue manager, which is an Information State Update engine. Firstly, we will demonstrate the development environment which includes automatic generation of speech recognition grammars for robust interpretation of spontaneous speech, and uses the application database to generate lexical entries and grammar rules. A simple graphical interface allows users (i.e. developers) to easily and quickly create and the modify SDS without the need for expensive service providers. Secondly, we will demonstrate the deployed system which enables participants to call up and speak to the SDS recently created. We will also show a pre-built application running on the iPhone and Google Android phone for searching for places such as restaurants, hotels and museums.

[1]Patent Pending

1 Introduction

With the advent of new mobile platforms such as the iPhone and Google Android, there is a need for a new way to interact with applications and search for information on the web. Google Voice Search is one such example. However, we believe that this simple "one-shot" search using speech recognition is not optimal for the user. A service that allows the user to have a dialogue via their phone opens up a wider set of possibilities. For example, the user may be visiting a foreign city and would like to have a discussion about the types of restaurants, their cuisine, their price-range and even ask for recommendations from the system or their friends on social networking sites. The Dialogue Understanding Development Environment or DUDE makes this possible by providing a flexible, natural, mixed initiative dialogue using an information state update dialogue engine (Bos et al., 2003).

Currently, if a company wishes to deploy such a spoken dialogue system, they have to employ a costly service provider with a long turn around time for any changes to the system, even minor ones such as a special promotion offer. In addition, there is steep competition on application sites such as Google Market Place and Apple App Store which are populated with very cheap applications. DUDE's Development Environment takes existing business-user resources and databases and automatically generates the dialogue system. This reduces development time and, therefore, costs and opens up the technology to a wider user-base. In addition, the DUDE environment is so easy to use that it gives the control back into the business-user and away from independent services providers.

In this paper, we describe the architecture and

Proceedings of SIGDIAL 2009: the 10th Annual Meeting of the Special Interest Group in Discourse and Dialogue, pages 148–151,
Queen Mary University of London, September 2009.

technology of the DUDE Development Environment and then discuss how the deployed system works on a mobile platform.

2 The DUDE Development Environment

Figure 1 shows the DUDE Development Environment architecture whereby the main algorithm takes the business-user resources and databases as input and uses these to automatically generate the spoken dialogue system which includes a Voice XML generator. Advantages of using business-user resources such as Business Process Models (BPM) (Williams, 1967) include the fact that graphical interfaces and authoring environments are widely available (e.g. Eclipse). In addition, business-user resources can contain a lot of additional information as well as call flow including context, multi-media and multiple customer interactions.

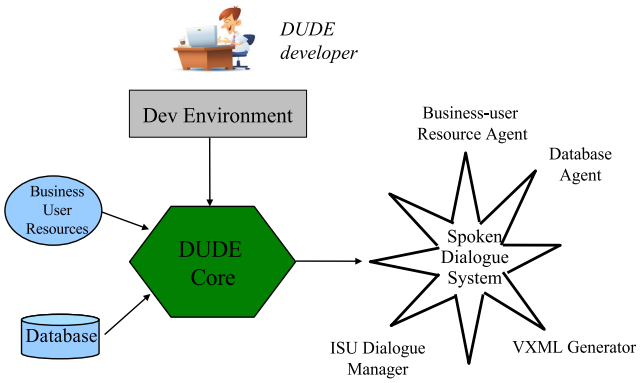

Figure 1: The DUDE Architecture

2.1 Spoken Dialogue System Generation

Many sophisticated research systems are developed 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. We present a solution whereby BPMs and related authoring tools are used to specify *domain-specific* dialogue interactions which are combined with *domain-general* dialogue managers. Specifically, the DM consults the BPM to determine what task-based steps to take next, such as asking for price range after establishing preferred cuisine type. 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 offer if the user is business-class", are compiled into domain-specific parts of the Information State. XML format is used for BPMs, and they are compiled into finite state machines consulted by the spoken dialogue system. The domain-general dialogue manager was mostly abstracted from the TALK system (Lemon et al., 2006).

Using DUDE, developers do not have to write a single line of grammar code. There are three types of grammars: (1) a core grammar, (2) a grammar generated from the database and BPM, and (3) dynamically generated grammars created during the dialogue. The core grammar (1) was developed to cover basic information-seeking interactions. In addition (2), the system compiles relevant database entries and their properties into the appropriate "slot-filling" parts of a SRGS GRXML (Speech Recognition Grammar Specification) grammar for each specific BPM node. Task level grammars are used to allow a level of mixed initiative, for example, if the system asks "what type of cuisine?" the user can reply with cuisine and also any other slot type, such as, "cheap Italian". The dynamically generated grammars (3), such as for restaurants currently being recommended, minimizes grammar size and makes the system more efficient. In addition to the above-mentioned grammars, developers are able to provide task spotter phrases and synonyms reflecting how users might respond by using the DUDE Development Environment. If these are not already covered by the existing grammar, DUDE automatically generates rules to cover them.

The generated SRGS GRXML grammars are used to populate the Voice XML pages and consequently used by the Voice XML Platform Speech recogniser. In this case, we deploy our system to the Voxeo Platform (http://www.voxeo.com). As well as the W3C standard SRGS GRXML, DUDE is able to generate alternative grammar specifications such as SRGS ABNF (Augmented Backus-Naur Form), JSGF ABNF (Java Speech Grammar Format) and Nuance's GSL (Grammar Specifica-

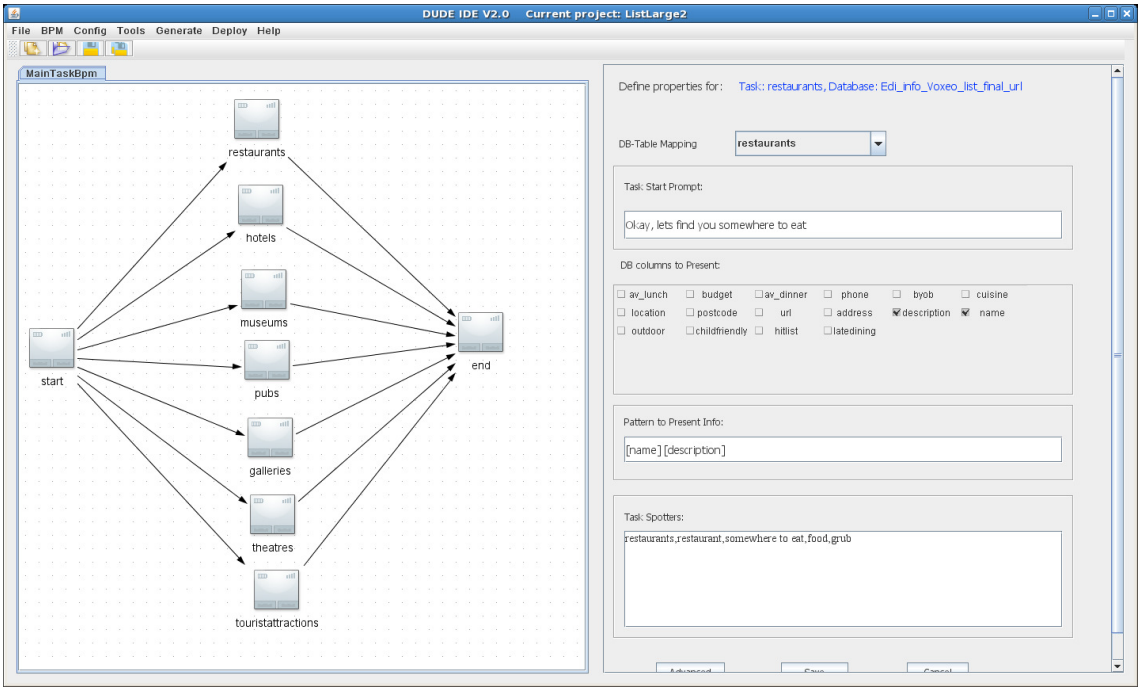

Figure 2: Example: using the DUDE Development Environment to define spotter phrases and other information for the different BPM tasks

tion Language).

2.2 The Development Environment

As mentioned above, the DUDE Development Environment can be used to define system prompts and add task spotter phrases and synonyms to the grammars. Figure 2 shows the GUI with the BPM on the left hand side and the properties pane for the restaurants task on the right hand side. In this pane the developer can define the system prompt, the information to be presented to the user and the spotter phrases. Here the developer is associating the phrases "restaurants, restaurant, somewhere to eat...." with the restaurant task. This means that if the user says "I want somewhere to eat", the restaurant part of the BPM will be triggered. Note that multi-word phrases may also be defined. The defined spotters are automatically compiled into the grammar for parsing and speech recognition. By default all the lexical entries for answer-types for the subtasks will already be present as spotter phrases. DUDE checks for possible ambiguities, for example if "pizza' is a spotter for both cuisine_type for a restaurant task and food_type for a shopping task, the system uses a clarification subdialogue to resolve them at runtime.

Figure 3 shows the developer specifying the required linguistic information to automate the cuisine subtask of the restaurants task. Here the developer specifies the system prompt "What type of cuisine do you want?" and a phrase for implicit confirmation of provided values, e.g. "a [X] restaurant", where [X] is a variable that will be replaced with the semantics of the speech recognition hypothesis for the user input. The developer also specifies here the answer type that will resolve the system prompt. There are predefined answer-types extracted from the databases, and the developer can select and/or edit these, adding phrases and synonyms. In addition, they have the ability to define their own answer-types.

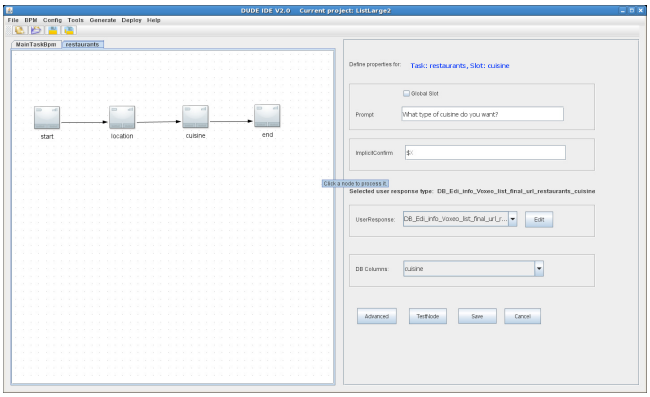

Figure 3: Example: using the DUDE Development Environment to define prompts, answer sets, and database mappings for the cuisine subtask

3 Deployment of the Generated Spoken Dialogue System

The second part of the demonstration shows a pre-built multimodal application running on the iPhone (http://www.apple.com) and Google Android phone (http://code.google.com//android). This application allows the user to have a dialogue about places of interest using The List website (http://www.list.co.uk). Figure 4 shows screenshots of the iPhone, firstly with The List homepage and then a page with content on Bar Roma, an "italian restaurant in Edinburgh" as requested by the user through spoken dialogue.

Figure 4: DUDE-generated iPhone List Application pushing relevant web content

Figure 5 shows the architecture of this system whereby the DUDE server runs the spoken dialogue system (as outputted from the DUDE Development Environment). This system dynamically generates Voice XML pages whose dialogue move and grammar is determined by the Information State Update Dialogue Model. These Voice XML pages are sent in real time to the Voice XML platform (in our case Voxeo) which the user talks to by placing a regular phone call. In addition, DUDE communicates the relevant URL via a server connection.

4 Summary

This paper describes a demonstration of the DUDE Development Environment and its resulting spoken dialogue systems as deployed on a mobile phone, specifically the iPhone and Google Android. With the emergence of web-enabled smart-phones, a new and innovative interactive method is needed that combines web-surfing and dialogue in order to get the user exactly the information required in real time.

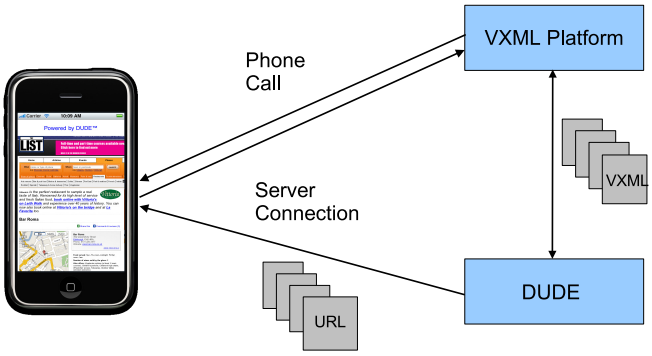

Figure 5: Architecture of deployed DUDE Application on a mobile phone (e.g. the iPhone)

5 Acknowledgement

This project is funded by a Scottish Enterprise Proof of Concept Grant (project number 8-ELM-004). We gratefully acknowledge The List for giving us data for our prototype application.

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.

Adam Cheyer and David Martin. 2001. The Open Agent Architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1/2):143–148.

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*, pages 119–122.

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

S Williams. 1967. Business process modeling improves administrative control. *Automation*, pages 44–50.