DiaMant: A Tool for Rapidly Developing Spoken Dialogue Systems

Gerhard Fliedner, Daniel Bobbert
CLT Sprachtechnologie GmbH
Stuhlsatzenhausweg 69
D-66123 Saarbrücken
{fliedner,bobbert}@clt-st.de

1 Introduction

Following the recent advances in speech recognition, now allowing the reliable speaker-independent recognition of one to two thousand words, an increasing number of natural language dialogue systems has been developed. These aim at two main areas: information access dialogues (e.g. travel information and bank account management) and machine control (e.g. controlling car functions, VCRs, and robots). There is a growing demand for tools that allow the rapid development of such dialogue systems. The users increasingly expect systems that allow a free dialogue without the need to learn a special command vocabulary and a menu-oriented hierarchical dialogue structure.

At CLT Sprachtechnologie, we have developed DiaMant (Dialogue Management Tool), a tool that allows the rapid development of dialogue systems based on extended finite state dialogue models. The development is done inside a graphical user interface that intuitively shows the dialogue model as an automaton in a graph representation. The dialog flow can be edited without the need to write a single line of code. The tool has already been successfully used in a number of applications.

As an important feature, the tool allows setting up Wizard of Oz experiments with one mouse click, where a human 'wizard' can replace input devices such as a speech recognizer in early development phases. This has proven especially important in testing and improving dialogue systems, leading to a higher acceptance of the final systems.

2 Dialogue Modeling Using FSA

Finite state automata (FSA) are an intuitive way of modelling dialogues (cf. e.g MacTear 99). An FSA can be represented as a graph with nodes corresponding to FSA states and edges to transitions.

In this graph representation of an FSA-based dialogue model, then, an automaton state implicitly represents a dialogue state (e.g. a state were a certain information has just been provided by the user). The user's dialogue moves are represented by the edges of the graph. Executing a dialogue in this model can then be thought of as traversal of the graph, selecting transition edges according to the user's input in the corresponding state.

This basic model, however, is only suited for modelling very simple dialogues. For more complex dialogues, it is useful to add a possibility of explicitly storing information in slots (variables). Such an enhanced FSA with slots forms the basis of our dialogue development tool.

3 DiaMant: Overview

DiaMant has been implemented in Java. It is therefore platform independent, running on all operating systems for which a Java VM is available (including Windows, Linux, Solaris, and MacOS). Hardware requirements for the system itself are modest (Pentium II @ 200 MHz, 64 MB or comparable).

Client modules (among them speech recogniser and TTS systems but also databases or other devices) can be integrated using a simple-to-use interface. Since the interface is TCP/IP based, clients can even be running on other machines. For a number of commercial speech recognisers and TTS systems, the required modules already exist. Others can easily be added via a simple Java wrapper provided by CLT that allows users to develop new modules simply by extending a Java class without the need to deal with network and protocol details.

Developing a dialogue model is done with a graphical interface. The dialogue model is repre-

sented as a graph. A number of different types of nodes is available: input/output and slot manipulation nodes, sub-automata calls, and others.
Each of the different node types allows a suitable parametrization. An output node, e.g., allows the user to specify an output device (e.g. a TTS system or a display window) and a text string or data record to send to it. Input nodes allow giving any number of different possible input values to be matched against the input of a specified device. The input to be matched is given as a regular expression, allowing to match natural language input, especially from a speech recogniser, with a high degree of flexibility. For each possible input value, an outgoing edge is added to the input node that can be connected to other nodes by drag and drop.
Input can be stored in typed slots that can hold, e.g., numbers or strings. These slots can be accessed anywhere in the dialog, in order to branch depending on a slot value or to produce context sensitive output. Slots and constant values can be combined and evaluated in Boolean and arithmetic expressions.
Recurring dialogue tasks can be stored in sub-automata. These sub-automata have their own, local dialogue slots. When calling a sub automaton, parameters can be passed and resulting values can be returned. Thus, sub-automata can be readily compared with sub-routines and functions in programming languages, allowing to modularise the dialogue and keep it compact.
This intuitive way of dialogue modelling within an easy-to-use graphical user interface allows designing simple dialogs within a matter of hours.

4 Wizard of Oz Experiments

An important point in dialogue development is providing a possibility for user experiments (Muntteanu and Boldea 2000). These experiments should, in general, be employed as early as possible in the dialogue development cycle: Very often, small matters decide on the user acceptance of a system, sometimes as little as the wording of a system prompt. We have found it useful in dialogue development to be able to directly use the current dialogue model in a user experiment.
DiaMant provides a convenient way of setting up Wizard of Oz-type experiments. Here, one or more input devices are simulated by a 'wizard', one or more persons, possibly identical with the experimentator. This allows testing a system early, when the full functionality is not necessarily present yet.
When run in the 'Wizard mode', the system uses a window on the wizard's machine to show information about the dialogue state to the wizard. Whenever an input is expected, the different possibilities provided for in the dialogue model are presented to the wizard to choose from (by mouse or keyboard).
User experiments are extensively logged (including exact time stamps), allowing the easy identification of problematic areas. Using the log file, an experiment can be replayed, helping with a detailed analysis.

5 Example Systems

Several example systems have been built using DiaMant. One of them is a speaking elevator that allows the user entering an elevator carriage to speak their destination floor aloud (including just naming a person, whose office is on that floor).
In a recent seminar (winter term 2002/2003) at the Saarland University's Computational Linguistics dept., DiaMant has been successfully used by student groups to implement dialogue models for controlling robots built with LEGO Mindstorms kits. A similar course is now taking place at the University of Kassel (summer term 2003).
DiaMant was also used successfully for user experiments while developing a complex information seeking dialogue for a flight information system.

6 Further Development

DiaMant has been successfully employed in a number of tasks, commercially as well as for educational purposes. We currently plan to make the tool available for interested researchers/developers.
Important goals are the adaptation of the tool to process standard data formats, especially import and export Voice XML, and a tighter integration of grammar development for the speech recogniser.

7 References

Munteanu, C. and Boldea, M. 2000. "MDWOZ: A Wizard of Oz Environment for Dialog Systems Development." *Proceedings of LREC*.

McTear, Michael F. 1999. "Software to Support Research and Development of Spoken Dialogue Systems." *Proceedings of the Eurospeech.*