A Framework for Information-State based Dialogue

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

1 Introduction

For the development of flexible and user-adaptive natural language dialogue systems, one needs a powerful dialogue engine. This engine must allow representing the dialogue in a dialogue model and executing this model. One framework for dialogue description that has evolved over the past years is based on the notion of information states and their update via rules.

At CLT Sprachtechnologie, we have developed a tool based on information state update (ISU). The tool allows the user to model dialogues using an information state, update rules and static plans. The execution of these models is based on an interaction of input processing and plan execution. The system provides the user with a dialogue modelling framework that is well-integrated into an execution system, obviating the need of worrying over basic implementation details, yet flexible enough to allow for different sorts of dialogues. Input and output devices (speech recognizer, TTS, screens, buttons, etc.) as well as databases can be integrated over an easy-to-use client interface.

Our system has been successfully used in developing, among others, an airport flight information system, giving e.g. information on departure times, gates, and flight status, with a high degree of flexibility in user input. However, for accommodating novice users the system automatically switches into a more system-guided mode asking the user for their flight information one step at a time. In usability tests, this dialogue system performed very well, with usability scores around 70%. This shows that the underlying system is well-suited for flexible dialogue systems. As the demand increases with the recent advances in dialogue systems, we expect a growing interest in such tools.

2 Information State Update

The idea of information state update for dialogue modelling is centred around the information state (IS). Within the IS, the current state of the dialogue is explicitly represented. Dialogue moves involve an update of the information state. Thus, user inputs are matched against a set of possible update rules that change the IS in the appropriate places (e.g. a new value is entered into a slot).

ISU or related frameworks have been used in a number of academic work recently. However, this work has often relied on implementing tailor-made solutions. General solutions (most prominent among them TrindiKit, Traum et al. 99) have provided a flexible framework, but left much of the actual implementation details to the user.

When developing our system, we have strived for a solution that is, on the one hand, flexible enough to accommodate a wide range of different dialogue types and I/O modalities, on the other hand, has a full-fledged execution system allowing a rapid development of dialogue systems.

3 System Overview

Our system has been implemented in Java. It is thus platform independent and runs on any computer system for which a Java VM is available (among others, Windows, Linux, Solaris, and MacOS). Hardware requirements for the system itself are modest (Pentium II processor @ 200 MHz, 64 MB main memory or comparable), plus hardware requirements for devices (e.g. speech recognisers).

In our system, the information state is realised as a typed, attributed structure. This structure can be freely defined by the user for the task in hand. The systems offers a full type system, including basic

types such as integers, strings, and Booleans, and complex types such as lists and records of these basic types. Attributes allow, e.g., the easy representation of meta information (such as grounding).
User input can be pre-processed by a built in context-free grammar parser with semantic tags to allow a semantic interpretation of the user utterance. Input is then passed to a set of rules, which try to match the input with patterns for the current information state, the input device and the actual input. Patterns can be simple value tests but also structural patterns or regular expressions, allowing the input or part of it to be bound to variables for further processing. The first matching rule's body is then executed. A rule can update the information state, generate system output, and push new plans that try to fulfil the systems goals. These plans are defined statically within the dialog model.
Plans are written in a procedural programming language, closely resembling JavaScript. The power of this language combined with the parameterisation of plans enables the generation of context sensitive, user adaptive dialog moves. An important additional feature of our plans is that they can carry information about their own 'fulfilment'. If a plan for eliciting some information from the user (e.g. a time) has been successful (i.e., the user has given the relevant time in answer) is called a second time, it would find that the information is already present and therefore not ask the user again to provide it. This has proven especially important in cases when there is more than one way of eliciting a certain information (e.g. if the user provides a flight number, asking for the departure time becomes unnecessary).
Closely connected with this is the notion of plan failure: The system maintains a stack of currently executed plans. Whenever a plan signals its failure, a mechanism resembling exception handling in many programming languages is used to identify a higher plan that is 'willing' to take over control.
Every input is usually handled in its own thread allowing for example barge-in (if the speech recogniser supports this). On the other hand, plans can specify that they need to wait for user input in order to be able to proceed. This turn-taking between system and user (corresponding to rules and plans) allows for a very flexible dialogue management.
The execution system gives a graphical representation of the current information state and client communication, allowing the in-place modification of values and attributes. Simulation of specific dialog situations for testing purposes therefore becomes very easy.

4 One Example System

As an example system, we have used our dialogue engine to implement an information seeking dialogue system in German based on a database with typical flight information of an airport. The user can enquire about a number of different flight details (ETA/ETD, delays), but also airport information (departure gate, parking facilities).
The system lays great stress on flexibility (i.e., the user can give information in any sequence, give more than one piece of information in one turn, use a wide variety of different words and sentence constructions). This, of course, necessitates various different manners of grounding (implicit to explicit, including clarification). An especially important point for novice users is that the system can switch to a more system-initiated mode where it asks the user for pieces of information. Even then, however, the user is always free to give information that the system had not requested or to correct a piece of information given earlier.
We have found, that our dialogue management tool has been very helpful in implementing this free example dialogue system. This is especially due to the fact that the rules allow the matching of the users' inputs even in places where the system did not actually expect them, combined with the plans carrying information about their own fulfilment

5 Further Development

After the recent, very promising experiences from our example systems, we plan to offer our development tool to interested users in the community.
As an important addition, we plan to enhance our system by adding a graphical debugging interface that allows the close inspection of the running system in addition to the existing logging facilities. Also, we plan to replace our custom plan description language with ECMA-Script, making the software more standards compliant.

6 References

David Traum et al. 1999. *A model of dialogue moves and information state revision, Trindi Deliverable D2.1*, http://www.ling.gu.se/research/projects/trindi/