TITLE	:	WP6D4 – Chapter 1: Introduction
PROJECT	:	P2218 - SUNDIAL
AUTHOR	:	Nigel Gilbert
ESTABLISHMENT	:	University of Surrey
ISSUE DATE	:	May 9, 1993
DOCUMENT ID	:	
VERSION	:	1
STATUS	:	Draft
WP NUMBER	:	
LOCATION	:	Nellie:Projects Folder:...:WP6D4intro.tex
KEYWORDS	:	
SIGNED	:	

Abstract

This is a draft of the first, introductory chapter for the WP6D4 deliverable. PH added "Table of Customisable things", and some words in teh main text.

Contents

1 Overview of previous documents

This is the fourth and final deliverable from Work Package 6, the work package concerned with dialogue management. It reports an evaluation of the generic Dialogue Manager that has been designed and implemented by a European team over the last four years for the Sundial project. The first deliverable, WP6D1, submitted in July 1990, set out the rationale for having a dialogue management component in a speech understanding system and proposed a set of detailed requirements for the Dialogue Manager. The second deliverable, WP6D2, July 1991, proposed a software architecture and principles for the Dialogue Manager, in addition reporting experience with some pilot and experimental implementations. The third deliverable, WP6D3, July 1992, described the so called 'intermediate demonstrator', a complete implementation of the principles set out in WP6D2, but one that still required considerable development to remove bugs and increase performance to approach real-time. This final deliverable describes the progress made since the writing of WP6D3 and also looks back to the initial requirement specification, WP6D1, comparing our ideas and understanding then with what we have achieved during the project.

2 Purpose of this document

The Technical Annex of the project proposal, which defines the technical objectives and expected output of the project, notes that the present deliverable is to be an evaluation. As suggested by the description in the Annex, it offers two rather different kinds of evaluation. Firstly, it evaluates the current performance of the Dialogue Manager against the requirements initially specified in WP6D1, comparing the system's capabilities, point by point, with the capabilities listed in the requirement specification. Second, it examines the performance of the system when tested with real dialogues from a sample of users and with dialogues constructed to exercise the full range of the system's capabilities. In addition, this deliverable serves as an introduction to the Dialogue Manager as currently implemented and includes a 'User Manual' to guide others on customising the system for different applications and different dialogue strategies.

3 Structure of this document

Following this introductory chapter, which reviews the advantages of dealing explicitly with dialogue management in speech understanding systems, Chapter 2 summarises the requirements for a dialogue management system that were defined in the first deliverable. Chapter 3 provides a general description of the current implementation of the Dialogue Manager, focusing on its core functions and interfaces. To indicate what dialogue management can achieve in real user

dialogues, Chapter 4 examines the course of one dialogue in detail, showing how the system helped to make the interaction coherent and effective in reaching a conclusion. In Chapter 5, the methods that were used to evaluate the Dialogue Manager are summarised and the results of the evaluation reported. This chapter also includes a list of those dialogue phenomena that cannot be handled by the current system, with suggestions for the further research and design that is still needed. An appendix lists the test files used to develop and evaluate the system. There are also user manuals for those who wish to adapt the system. In the rest of this introductory chapter, we consider the basic principles of dialogue management, beginning by outlining the reasons for including a dialogue management component in a speech understanding system. We then set out the principles which underlie the design of the Sundial Dialogue Manager and review the methods we have used to evaluate the design. The latter are described in more detail in Chapter 5. Finally, we describe the constraints within which a practical system must work given the current state of the art in speech systems.

4 Why include dialogue management in a speech understanding system?

Our experience with the Sundial system has shown that there are three main advantages in including a dialogue management component in a speech understanding system.

4.1 Coherent dialogue

Simulation experiments (reported in the deliverables from WP3) indicate that users expect speech understanding systems to be able to engage in 'conversations' with openings and subsequent interaction comparable with what might be expected from a human agent. For example, they expect an initial greeting and introduction from the system, and when they have proposed an enquiry, they expect a series of clarifications and confirmations until the system is able to provide an answer. This suggests that the system must be capable of organising its interaction with the user in terms of openings, question-answer pairs and closings. It also suggests that the system must be capable of maintaining a developing context that describes the user's query and the growing knowledge about it (e.g. the accumulating parameters which specify the query with increasing precision). These capabilities demand that the system must consider the structure and organisation of the dialogue as a whole, and not just react to users' utterances one at a time. Furthermore, it must have the ability to modify its understanding of the user's current utterance, and to design its own replies, according to the previous course of the dialogue. These capabilities can be summarised under the heading of 'maintaining dialogue coherence'.

4.2 Predictions

Because the dialogue management component can keep track of the developing interaction, it is able to predict the user's likely next moves. The predictions it makes can be used to guide the recognition component of the speech understanding system, allowing conversationally incoherent possible interpretations to be discarded at an early stage and improving the recognition accuracy and the processing speed of the whole system.

4.3 Repair

No speech understanding system is now or is ever likely to recognise users' utterance with perfect accuracy; indeed, even people often make recognition errors. On the other hand, in oral dialogue speakers display the characteristics of "spontaneous speech", i.e. they make 'mistakes' ranging from pronounication mistakes to 'slips-of-the-tongue', incorrect grammar, and, finally, simple error: People don't always say what they mean. So, even a speech understanding system with perfect recognition accuracy would have to be able to deal with such communication problems.

In human-human conversation, there are well developed, although as yet imperfectly understood mechanisms for the 'repair' of recognition and other problems. The dialogue management component can implement similar repair mechanisms. The Dialogue Manager is also able to track the overall degree with which the system has understood a particular user and can modify the dialogue strategy and repair mechanisms to suit. For example, it can arrange for the system to ask only questions expecting a 'Yes' or 'No' answer if the understanding rate has been very poor, relying on the fact that it is much easier to recognise a choice of two words than to recognise a much wider vocabulary. If understanding problems persist, the Dialogue Manager can gracefully close down the interaction, referring the user to a human source of information. All three of these basic functions are available in the Sundial Dialogue Manager.

4.4 Interpretation in dialogue

In any dialogue that has more than one turn for each of the partners, linlguistic means may be used to refer to expressions that occurred in a previous turn; these are called anaphoric expressons (e.g. pronouns). Also, the situation of the dialogue may be such that it is possible to leave out parts of expressions that "go without saying"; this phenomenon is called ellipsis. Thirdly, referring expressions can have different meanings at different stages of the dialogue, e.g. "the next flight" at the beginning of a dialogue refers to the one that is temporally the next from that moment (in french "le prochain vol", whereas it means "the flight after the one we talked about", when one flight has already been mentioned

("le vol suivant").

To deal with these kinds of phenomena, it is necessary for the semantic interpretation to know the stage in the dialogue that has been reached and to have access to previous dialogue turns. Thus, interpretation has be be made in a context considerably larger than one user utterance. Also, it has to use a wide range of knowledge sources to find the most fitting interpretation at any stage in the dialogue. The Sundial Dialogue Manager can cope with most of these phenomena for all of the languages involved because it does the dialogic semantic interpretation on the basis of a surface semantic desciption delivered by the parsers.

5 Principles of the Sundial Dialogue Manager

A number of basic principles have informed our development of the Sundial Dialogue Manager.

5.1 A generic system

From the beginning of the project, it was apparent that, because none of the partners had previously built a Dialogue Manager (at least one of the sophistication envisaged), it would be possible and desirable to build a system which could be designed, implemented and used by all the four national demonstrators, despite the differences in language and application domain. Our ability to do this would provide a demonstration by example of the flexibility of the design, would confirm the hypothesis that dialogue strategies were similar across four European languages at least within a limited domain, and would contribute to the goals of ESPRIT in promoting collaboration across Europe.

We have achieved substantially this objective, despite the considerable challenges of developing experimental software across seven sites distributed through Europe. Only the French national demonstrator is not using the 'generic' Dialogue Manager. The Dialogue Manager has been customised to suit the English, Italian and German languages and the German and Italian train enquiry and the British flight enquiry applications. In all three demonstrators, identical code is run, but with different data and knowledge bases to suit the application and the chosen dialogue strategies.

Because the Dialogue Manager has been designed to be customisable, it would be straightforward to adapt it to information enquiry and ordering domains beyond the flight and train domains used in the demonstrations. In principle, any domain in which callers want to make enquiries to a large, fairly static database could be accommodated with ease. So could applications in which callers wish to place orders for one of a variety of items on offer.

5.2 Modularisation

While careful modularisation of a complex design is a fundamental principle of all software engineering, it was particularly important in the case of the Dialogue Manager, because those designing and implementing it were so geographically distributed and had such limited communications (in practice, electronic mail and meetings every two to three months). As Chapter 3 indicates, the Dialogue Manager is divided into four modules with a message passing protocol used to communicate between them. As well as having implementation advantages, the modularisation meant that the functionality of the system could be cleanly divided into interaction with the database, interaction with the parser, and so on. This in turn meant that specific knowledge bases could be associated with each module, aiding customisation. For example, to change the dialogue strategy used by the system, it is only necessary to edit a set of dialogue rules in one file and recompile.

5.3 Customization

The Dialogue Manger is language and task independent. The same software can handle any combination of the four languages involved in SUNDIAL with any of the tasks, provided the respective parser for that language can build the required parsed structure using the Semantic Interface Language (SIL). The system can be configured to the desired combination of language and task at initialisation time. A set of software flags determines, for example, which task knowledge base will be loaded, which application system interface and which parser for which language will be used. So, the Dialogue Manager itself is a generic system, and any running configuration is one instance of that generic system.

Apart from the testing, debugging and logging facilities, which are also switchable by means of software flags, several crucial parameters for the dialogue behaviour of the system can be set by simple changes in software flags. The confirmation strategy, the tolerance to recognition errors, the maximum number of answers given in one set to a user request and several other parameters, can all be modified very easily during the operation of the system to accommodate the needs of that particular environment (phone quality, etc.). Changes in the setup of the system that do not require developers' skills can thus be made by system operators without having to deal with the program code. This adds considerably to the flexibility of the Dialogue Manager.

Because of the modular nature of the system and the uniform SIL interface (see below), developers can exchange modules inside the Dialogue Manager or add or exchange external components (e.g. a generation module starting from semantic descriptions instead of a template generator) after they have tested these components by means of 'test files', without having to deal with the internals of other modules. Thus, the Sundial Dialogue Manager can be adapted to new

technologies easily, while the main body of the system remains fully operational. This is a very valuable feature in terms of software re-useability.

The modular architecture, in principle, also permits a developer to do away with the message passing module and to run the different sub-modules as different processes on parallel hardware. As this has not been tested yet, it is not clear whether the resulting speed-up would justify the (nonetheless rather small) expense of doing so.

5.4 SIL

Communication between modules and between the Dialogue Manager and the linguistic processor is conducted using the Semantic Interaction Language (SIL), a language formalism designed in the Sundial project. Requiring all interfaces to be defined in terms of this language aids the transparency of the code and reduces the amount of intermediate processing which otherwise would be required to translate from one protocol to another. The overhead involved in such translations when SIL is not used became obvious from the experience of the French, whose parser does not generate SIL and who, finding the overhead intolerable, decided not to use the generic Dialogue Manager. The Italians, who do use the Dialogue Manager, have to accommodate a processing delay while the output from their parser is translated to SIL. One of the major features of the SIL language is that it allows semantic description generated by a parser to consist of arbitrarily many sub-decriptions. Thus, the parser does not necessarily have to find a semantic description spanning the whole utterance, but can send the semantics of any sub-piece that it has found for dialogic manager to "make sense of it". This adds considerably to the robustness of the system, as corrupted speech input, with non-words like "emm" or excessive noise in single places, or utterances consisting of more than one sentence still can be parsed to a certain degree, and some interpretation and an appropriate system reaction will neverthless take place.

6 Implementation

6.1 Hardware and software

The Dialogue Manager is implemented using Quintus Prolog 3 running on a Sun workstation. In order to achieve an acceptable processing speed, a Sun Sparcstation 10 or better is required. The Dialogue Manager may communicate with other components of the Sundial system directly, through Unix pipes or with remote procedure calls to processes on other machines. All these communication media are used in the various national demonstrators.

6.2 Development process

As noted above, the first step in the design of the Dialogue Manager was the specification of requirements (WP6D1). This was followed by a period of design, intermingled with the development of a functional specification (WP6D2). At a early stage, the modular architecture and message passing protocol was agreed, which meant that, at least to some extent, each module could be developed independently of the others, provided that the interface specifications were kept unchanged. In practice, however, increasing the functionality of a module meant that new or amended messages to other modules were required, so there were continuing difficulties arising from module interface inconsistencies.

To help uncouple the development of the individual modules, a method of creating 'test files' consisting of a record of the stream of messages between modules was devised. Because a test file precisely specified the expected interaction between the Dialogue Manager modules for a given dialogue, a developer could check the operation of his or her module against the test file without having any of the other modules available. Chapter 5 provides more details about the test file methodology.

Initial versions of all the modules were developed for the 'intermediate demonstrator' described in WP6D3. However, this development had also taught us a great deal about the problems of dialogue management that had not been apparent when the modules were first designed. Two modules (the task module and the dialogue module) were completely re-implemented to take advantage of our improved understanding. The re-implementation had the additional benefit of yielding modules which were much smaller, simpler and faster than the original versions.

There followed a period, lasting almost a year, of gradually improving the functionality of the whole Dialogue Manager and amending it to cater for problems encountered in dealing with test dialogues. An increasing number of test dialogues (encoded in SIL) were accumulated. Some of these were crafted 'by hand' to test particular aspects of the system; others were taken directly from or were slight modifications of dialogues collected from the simulations. These dialogues were contributed by all partners and now number over a hundred (see Chapter 2). They provided a pool of test material against which all new versions of the Dialogue Manager were run to ensure its quality.

In the final phase of development, the Dialogue Manager was integrated with the rest of the Sundial system in Italy, Germany and England to form the national demonstrators. At this stage, the processing speed of the Dialogue Manager became crucial, and considerable efforts were made to speed up the Belief module, the one module that involves an appreciable amount of computation.

The concluding period of the project was concerned primarily with the evaluation of the Dialogue Manager (see also Chapter 5).

7 Scope and constraints

In order to implement a system which satisfied the requirements of a usable speech understanding system, it had to be recognised that the Dialogue Manager would be limited in a number of respects. The principal ways in which its scope and coverage are constrained are:

7.1 Limited domain

The Dialogue Manager is designed for applications in which the main function to be supported is a database lookup by an untrained user interacting with the speech understanding system over the public telephone network. While this imposes requirements for real-time understanding of connected 'natural' speech over a telephone line, it also means that it is possible to work on the assumption that the users' vocabulary and the semantics of the user's utterances are confined to a fairly small set (e.g. a lexicon of around 1000 words). It also means that the tasks demanded of the Dialogue Manager are quite limited.

7.2 Limited task knowledge

In particular, the task knowledge required by the Dialogue Manager is confined to simple database lookups and updates (the latter for flight reservations etc.).

7.3 Limited self knowledge

The Dialogue Manager is able to recognise if the user's request is slightly out of its scope (for example, in the British national demonstrator, if the user requests information about a flight operated by an airline other than British Airways), but is otherwise not able to handle out of scope enquiries. This is partly due to limitations in the coverage of the system's lexicon, which prohibit enquiries that are well out of scope being recognised by the system.

7.4 Limited speed of processing

Ideally, the Dialogue Manager ought to respond to user's input 'immediately'. Because processing takes a finite time, the only way in principle that this could be achieved is to process input incrementally while the user is speaking. In fact the architecture of the system as a whole means that the Dialogue Manager only obtains its input after the user has finished speaking, after the front end has recognised the end of the user utterance and after the linguistic processor has done its job. The Dialogue Manager is then able to begin processing and in the current implementation, computes the semantics of the system's next utterance in between 0.1 and 3 seconds depending on the complexity of the utterance and

the previous history of the interaction (on a Sun Sparcstation 10-41). This output is then transferred to the speech generation component, which imposes a further slight delay.

Using the current architecture, the Dialogue Manager could be made to react more quickly by running it on faster hardware. Alternatively, it is possible that more efficient search algorithms could be used in the Belief module. The main hurdle, however, remains the 'pipeline' arrangement of front-end, linguistic processor, dialogue manager and speech generation in the current architecture.