The Science and Art of Voice Interfaces[*]

Emiel Krahmer
e.j.krahmer@kub.nl

[*] This report was written for Philips Research (Eindhoven) while the author was working at IPO, Center for User-System Interaction, TU/e, Eindhoven University of Technology. The report benefitted from discussions held in IPOs *dialogue reading group*. Thanks are due to René Collier, Huub Prüst, Frans Blommaert and Steffen Pauws for comments. The author is avaibable for further questions: until the end of July at IPO, and starting August 1, 2001 at the Faculty of Arts, Tilburg University, P.O. Box 90153, NL-5000 LE, Tilburg, The Netherlands.

Contents

1 Prologue

The recent successes of speech technology have led to a renewed interest in an old ideal: communicating with machines by voice. Of course, Science Fiction provides a nearly unbounded resource for examples of such *'voice interfaces'*, ranging from HAL, the hyper-intelligent and very communicative computer in Stanley Kubrick's *2001, A Space Odyssey* (1968),[1] to K.I.T.T., the talking car which helped David Hasselhoff during his fight against crime in *Knight Rider* (1982-1986). But also, and this is more relevant for our current purposes, in Artificial Intelligence communicating with machines has long been a topic of interest. This interest derived from a famous article written by Alan Turing in 1950, in which Turing outlined a test for computational intelligence. This test, which is currently known as the *Turing test*, is a game consisting of a conversation on any topic between a judge and two agents, where the judge has to determine which of two agents is human and which a computer. If the judge is unable to decide which of the two agents is the machine, the machine may be called intelligent. The underlying idea is that any program that can be mistaken for a human by its conversational behavior must be intelligent. Turing predicted that in the year 2000 computers would be able to pass the test to a large extent. More in particular, he suspected that on average an interrogating judge would not be able to identify the computer in 70% of the cases after a conversation of 5 minutes.

One of the first programs, and arguably the most famous AI program to date, which could enter the Turing test is ELIZA, the computational therapist developed by Joseph Weizenbaum at MIT in the mid-sixties.[2] Here is (part of) a transcript, taken from Weizenbaum (1966):

Patient:	Men are all alike.
Eliza:	In what way?
Patient:	They are always bugging us about something or other.
Eliza:	Can you think of a specific example.
Patient:	Well, my boyfriend made me come here.
Eliza:	Go on.
Patient:	Perhaps I could learn to get along with my mother.
Eliza:	Tell me more about your family.
etc.	

This may look like a quite impressive dialogue, but in fact the program is of a stunning simplicity. Essentially, all of Eliza's conversation capabilities are stored in a ranked database of key-words. This database contains entries such as the following:

word	**rank**	**pattern**	**output**
mother	10	?X	"Tell me more about your family."
are	3	?X are ?Y	"What if they were not ?Y."

Thus, each word is associated with a number (the rank, maximum value 10) an input pattern and an associated specification for the output. Eliza's entire algorithm can now be specified as follows:

[1] For an intriguing comparison of the techniques simulated by HAL and the current state of the art, see Stork (1996). This book, which is partly on-line, contains chapters by leading scientist such as Donald Norman (usability and design), Raymond Kurzweil (speech recognition), Joseph Olive (speech synthesis) and Daniel Dennett (philosophy/ethics). See `http://mitpress.mit.edu/e-books/Hal` (checked on 15/09/2000).

[2] Note that ELIZA only took typed not spoken input.

- Scan the patient's input for a keyword of which the pattern in the database matches the input.
- If there are several of these key-words, pick the one with the highest priority rank.
- Use the relevant specification to generate output.
- If there are no matching key-words say, "tell me more", "hmm, continue" etc.

Two important lessons can be learned from ELIZA. The first one is that, even though ELIZA would have a hard time passing the Turing test, *cooperative* people were fooled (at least for some time) by the system. To Weizenbaum's dismay, various psychotherapists actually recommended using ELIZA or similar systems. One of these is Kenneth Colby, the psychotherapist responsible for ELIZA's output statements as well as for PARRY[3]. In fact, Colby and associates have developed a "talk therapy" program called "Overcomming depression" (Colby 1999), which has reportedly been used by thousands of people, and which functions along exactly the same lines as ELIZA (but the database of Overcomming Depression is much larger, containing more than 40.000 key-words). Thus, such extremely simple techniques can bring you quite far. It illustrates that building conversational interfaces is to some extent an *art*; it requires careful attention to the way the system responds to the user's input and to the way the system expresses itself.

However, and this is the second lesson, fine tuning ad hoc techniques will only bring you so far. Given the simplicity of ELIZA's algorithm, it will be obvious that ELIZA has no knowledge about natural communication whatsoever. And in fact, a sceptic will soon find out that he is being fooled: some of the replies make no sense at all[4] and over time it soon becomes clear that ELIZA keeps no track of the progress of the conversation: there are many repetitions and the system clearly does not "remember" the information that was exchanged during earlier turns in the dialogue. In general, ELIZA completely lacks knowledge about language and about conversation.

That the lack of knowledge about language and conversation is troublesome becomes especially clear when looking at the winners in the yearly Loebner contest, which is a watered-down version of the the Turing test initiated by the New Jersey philanthropist Hugh Loebner. The Loebner contest offers two prizes: a big one ($100.000) for the first system which fully passes the test as Turing devised it exactly 50 years ago, and an annual smaller one ($2000) for the system which, in a limited setting, is judged the most human-like by a jury of experts. Fifty years after Turing's seminal paper, it is evident that there are no conversational systems forthcoming which are able to pass the Turing test (not even with a probability of .7 as Turing prophesied in 1950). The winners of the small prize tend to be collections of tricks in the spirit of ELIZA (applying random insertions of typing errors, for instance). As Shieber (1994), among others, pointed out: this is not the way to go. If we want to be able to devise better conversational machines, we should not opt for a bigger bag of tricks, but focus on the *science* of conversational interfaces. In the long run, it is essential to have knowledge and computational models of how to communicate.

[3] PARRY is one of the many ELIZA-inspired programs to have emerged over the past decades, this one mimicking a paranoid mind (e.g., Schank and Colby, 1973). Such programs are currently often referred to as "chat bots", a comprehensive overview can be found at `http://www.toptown.com/hp/sjlaven/index.html` (checked on 15/09/2000).

[4] If Saddam Hussein, say, were to consult ELIZA and remark that *The gulf war is the mother of all wars* ELIZA would reply with *Tell me more about your family*, given the high rank the key-word mother has in the database.

2 INTRODUCTION

This report addresses both the art and the science of designing and building voice interfaces. It provides an overview of the techniques for designing and building such interfaces which are currently available or which will be available in the near future. The report consists of two parts: Part I addresses the *science* of voice interfaces, while Part II focusses on the *art*. In Part I, the emphasis will be on the architecture of voice interfaces and on various computational models of dialogue. Different voice interfaces are based on different computational models of dialogue, each with its own advantages and disadvantages. Unfortunately, there is no single reference describing and comparing each of these models, hence the description in Part I is primarily based on separate articles, integrated here for —as far as we know— the first time.

Part II looks at the design and development of voice interfaces, more or less independent of the technological issues. Recently a number of articles and books have appeared describing this process (e.g., Bernsen et al., 1998, Gardner-Bonneau 1999 and Hulsteijn 2000). It is interesting to observe that these references essentially recast many of the findings from the general field of user interface design in terms of voice interfaces. Here, we have opted for taking a general user interface perspective and apply it to voice interfaces.

In both parts the emphasis is on the underlying theories and processes rather than on systems, in this way we can largely abstract over task- and implementation-dependent features. It is hoped that this makes the contents of the report more generally applicable. It is also worth stressing that this report does not provide a state of the art of the "enabling technologies" for voice interfaces. Thus, we do not discuss the current state of affairs for speech recognition, natural language processing or speech synthesis, even though the limitations of these technologies have a strong impact on how voice interfaces should be designed.

It is important to realize that a voice interface is not always the most suitable interface for a particular system. "Speech is the bicycle of user-interface design," according to Shneiderman (1998:328), "it is great fun to use (. . .), but it can carry only a light load. Sober advocates know that it will be tough to replace the automobile: graphic user-interfaces." The 'bicycle' status of speech is probably due to two factors. First, correcting speech recognition errors in spoken mode is difficult and reduces user satisfaction (see e.g., Weegels, 1999, Krahmer et al. 2000, Swerts et al. 2000 and section 7 of this report). A second limitation is that for some tasks (such as presenting lists) speech is a sub-optimal modality. Nevertheless, Shneiderman's statement is highly misleading for two reasons. First, and Shneiderman acknowledges this, there are various situations in which a graphic user-interface is of limited usage, e.g., when one is away from the PC, in so-called eyes-busy-hands-busy situations, etc. Second, it suggests that speech and graphical interfaces are mutually exclusive. This is certainly not the case, as shown by the recent emergence of sophisticated multimodal interfaces which combine the advantages of speech, graphics and other modalities (see e.g., Oviatt and Cohen 2000). It seems likely that multimodal interfaces with speech as an in- and/or output modality will gain popularity in the future. Even though we do not discuss multimodality in this report, most of the contents applies to multimodal interfaces with speech as well.

This report is primarily aimed at people with an interest in voice interfaces, both from a technical and from a usability point of view. Especially in the first part I have tried to be very explicit about the various models of dialogue, and consequently some parts can be skipped by those readers who only want to have a basic understanding of the problems and prospects of developing voice interfaces. I have explicitly marked such technical digressions as digressions, and readers can skip these without losing track of the main line of this report.

The report is structured as follows. Section 3 gives a short overview of the kind of voice interfaces that exist and introduces some of the terminology. Then, in section 4 we discuss the main components of voice interfaces and describe two common architectures for voice interfaces. The two next sections form the core of part I. In section 5, a number of structural models of dialogue are described. These are models which allow navigation through a predefined structure of possible dialogue states. In particular, we discuss finite state models of dialogue and slot-filling approaches to dialogue. We also pay attention to methods which use statistics and machine learning techniques to learn optimal dialogue strategies automatically. In section 6, two alternative methods are discussed, one treats dialogue as a goal-directed process, the other aims at developing conversational agents. In section 7, extra attention is paid to one of the central problems of current voice interfaces: dealing with communication problems. Subsequently, in Part II we focus on the design and development process of voice interfaces. In section 8, it is defined what it means for a voice interface to be *usable*, and one usability engineering method is discussed which aims at optimizing the chances of developing such a usable voice interface. Section 9 describes a collection of general guidelines for user interface design, supplemented with specific guidelines for voice interfaces. Finally, section 10 discusses various methods to evaluate voice interfaces. In the final section various pointers for further reading are given.

Part I: The science of voice interfaces

3 Dialogues with voice interfaces

In a nutshell, a voice interface enables users to interact with some application using spoken language. This application can be a variety of things, and it goes without saying that the kind of application has a strong impact on the nature of the voice interface. For example, the application in question can be a piece of hardware (e.g., a PC, a PDA or a TV set). In that case, we speak of *voice control* or *command & control*, allowing users to operate the piece of hardware using their voice. This can be handy for a variety of reasons, for example, operating a PC using voice is very practical for someone who has RSI (repetitive strain injury). Similarly, the trend towards ever more miniaturization and disappearing keyboards, makes voice control an attractive alternative for operating PDAs. Voice control is also beneficial in "eyes-busy, hands-busy" situations. A good example is the voice control of a car stereo or navigation system. Finally, a good voice control interface may simply be easier to use and arguably is more fun than a conventional interface. However, currently little is known concerning the perceived benefits of voice control during prolongated usage. A voice control interface will typically be used by a limited set of users (for instance, all members of a family) in a specific environment (the home), they will use a limited vocabulary (primarily consisting of brief commands, e.g., "start recording") and the system's responses will mainly be feedback messages (a substantial amount of which need not even be spoken ones; e.g., the system starts recording).

A different kind of application is sometimes referred to as *interactive voice response (IVR)*.

IVR systems are the ’modern’ counterparts of the touch-tone (or DTMF) dialogues. Instead of ”press 1 if you have questions about your latest telephone bill”, people are now offered the possibility of *saying* 1. In this case, the application is centered around a highly structured form of information exchange. IVR systems are typically used by a non-homogeneous set of users, and they have a very limited vocabulary (for instance, only the digits, from 0 to 9, plus ”yes” and ”no”). The dialogue is extremely rigid; the system asks a series of questions, forcing users to stay within the highly limited confines of the dialogue (”Enter your access code digit by digit!” (*user utters access code*) “Your access code is 1234. Is this correct, say ’yes’ or ’no’!”). In general, these IVR dialogues have the same ’functionality’ and the same ’feel’ as DTMF dialogues and one might conjecture that the use of voice does not really add much in this case. However, one should not underestimate the number of non-touch tone telephones still in circulation, and moreover one can hardly overestimate the number of mobile (GSM) telephones in circulation, and touch tone dialogues are obviously not suitable for GSMs. A simple but very efficient IVR-like system is one used by AT&T for the management of collect and credit-card calls. It asks only questions such as ”Do you accept this call?” and accepts only answers such as ”yes” and ”no”. Nevertheless, AT&T reported that this service has saved several hundred million dollars over a period of six years (Business Week, February 23, 1998).[5]

The application can also be a kind of database, which users can access over the telephone. In that case, the voice interface is often referred to as an *spoken information systems*. Such spoken information systems have been developed for, for instance, weather information (Zue *et al.* 2000, Sadek and de Mori, 1998), jobs (Sadek and de Mori, 1998), theatre information and booking (van der Hoeven et al., 1995), telephone and address information (Souvigner et al. 2000), and train or air travel (Aust et al., 1995, *inter alia*). These systems are intended for a large set of users, who will typically query the system in natural language and the system accordingly produces spoken replies. Spoken information services are currently an active area of research and development. This is partly due to the fact that many information providers cannot handle the increasing amount of requests for information. A variant of the spoken information systems are the systems which support user’s in performing a certain task, like using an electron microscope (Ahn *et al.* 1995), constructing a logistical plan (Allen et al. 1995, 2000) or repairing an electrical circuit (Smith & Hipp, 1994). Various researchers refer to spoken information systems and task-support systems as *(real/full-fledged) spoken dialogue systems* or *conversational interfaces*. It is true that of the systems discussed here these come closest to natural, human dialogues. Nevertheless, many of the theoretical and practical issues discussed in this report apply to the whole range of systems and hence we prefer to use the more neutral *voice interfaces*.

Still, different kinds of voice interfaces have to deal with different technical constraints (size and perplexity of the vocabulary, number of speakers that need to be recognized etc.) and, more importantly, require different dialogue strategies. As the application becomes more complex or the interaction becomes more natural, the burden on the communicative abilities of the voice interface increases. As our running example throughout this report, we shall consider developing a voice interface for a TV and VCR combination, where the user has the possibility of issuing spoken commands (”Switch to BBC1!”), browsing an electronic

[5] This special issue of Business Week contained an interesting special report devoted to the applications of speech technology, entitled *Let’s Talk*. The articles are available on-line at `http://www.businessweek.com/datedtoc/1998/980223.htm` (checked on 29/11/2000).

TV guide (”Can I watch a nice western movie tonight after nine ’o clock?”) and may receive spoken guidance for programming the VCR. In this way, the application encompasses essentially all kinds of voice interfaces that we just distinguished, and this allows us to discuss a variety of dialogue models.

4 Architectures of Voice Interfaces

In this section the main components of a voice interface are introduced, and two different architectures using these components are described.

4.1 The circular pipe-line architecture

Figure 1 shows one way in which a voice interface can be constructed. This architecture, modulo some minor variations, can be found in, for instance, Aust et al. (1995), Souvigner et al., (2000), Cole et al. (1996), and Bernsen et al. (1998). Here we refer to it as the “circular pipe-line architecture” because the flow of information proceeds in a loop and each module takes as input the output of the module which precedes it in the loop.[6]

Automatic Speech Recognition (ASR) This module takes the speech signal of the user as its input, and tries to determine which words were actually spoken. The output typically consists of a *word graph*, i.e., a lattice consisting of word hypotheses.

The majority of the ASR engines is based on a statistical technique known as Hidden Markov Modelling (for a comprehensive overview of the underlying statistical methodology, see Jelinek 1997). The occurrence of acoustic and phonetic variability in the speech signal (no two speakers have the same acoustic properties, even worse, a speaker never utters a given word in exactly the same way twice), implies that misrecognitions are unavoidable. Nevertheless, the current generation of ASR systems performs very well. Of course, the error-rates increase if the vocabulary of words to be recognized increases of if vocabulary contains many similar sounds. Many state-of-the-art recognizers are claimed to have an error-rate of less than 5%. Such good recognition results are typically obtained in a laboratory setting. When the speech recognized is embedded in a dialogue system, and used by ’naive’ (untrained) users, the error rate tends to show a marked increase (various articles report error-rates of between 30% and 40%, see e.g., Weegels 1999, Walker et al. 2000).

An important distinction is the one between *speaker-dependent* and *speaker-independent* recognition. A speaker-dependent recognized “adapts” to the voice characteristics of the user. The main advantage of speaker-dependent recognition is that the vocabulary of words that the system can recognize can be large (in the order 50.000 word forms). The down-side is that the user has to spend some time *training* the recognizer. A speaker-independent recognizer, by contrast, does not require such a training phase. The recognizer is “pre-trained” using the data of a large collection of speakers. However, speaker-independent recognition only works for limited vocabularies (a few hundred word forms).

[6] This, and the other alternative architectures to be discussed below are slightly misleading in that they suggest that the separate modules have no internal structure. This is typically not the case. For instance, in the ASR module usually a strict distinction is made between the acoustic part and the language model part. In the DM module, one typically finds separate modules dedicated to reference resolution, user modelling, reaction planning, context management etc. This ‘internal modularity’ has obvious advantages from a development point of view (easier to build, debug and modify). The current section is primarily devoted to the interaction between the various building blocks.

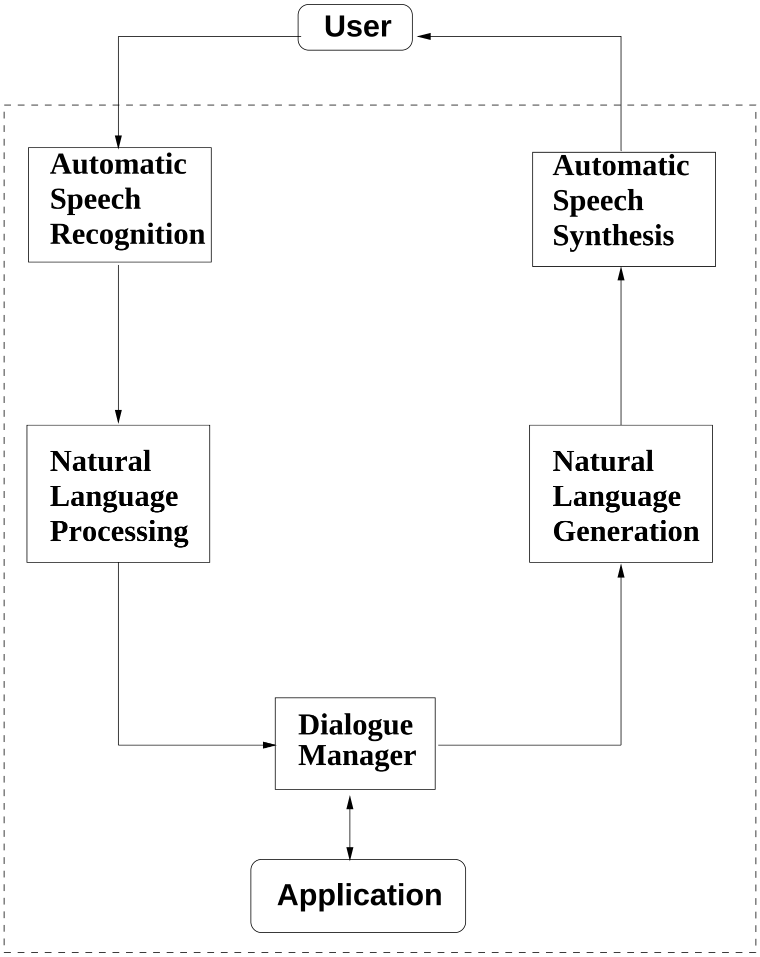
User
Automatic Speech Recognition
Automatic Speech Synthesis
Natural Language Processing
Natural Language Generation
Dialogue Manager
Application

Figure 1: The circular pipe-line architecture

NATURAL LANGUAGE PROCESSING (NLP) This module takes a word graph as input, and outputs a meaning representation. First, it has to be determined which of the hypotheses in the word graph is the most likely one, and subsequently, what the 'meaning' of the most likely word sequence is. In many practical voice interfaces, the first step is done on the basis of *acoustic confidence scores* and the second step is performed using *concept spotting*. Acoustic confidence scores are scores which indicate how confident the ASR module is about the various word hypotheses. They are currently an active field of research (see also section 7). Concept spotting techniques do not perform a complete linguistic analysis of what the user said, but rather try to extract some pieces of information from the input. For example, after a system question like (1):

(1) Which program do you want to record?

the concept-spotter would scan the input for words like *record* and store the program name immediately following it.

One of the advantages of concept-spotting is that it nicely circumvents the disfluencies inherent in spoken spontaneous communication (e.g., *Euh, I eh I want eh to record Pokémon*). On the other hand, in some cases *only* spotting concepts is definitely too limited (consider the following response to (1): *Well I definitely do* not *want to record "het spijt me"*).[7] Recent work by van Noord et al. (1999), among others, shows that it is possible to use full-fledged grammatical analysis on word-graphs.

DIALOGUE MANAGER (DM) The dialogue manager is arguably the central module of a voice interface, in the sense that it functions as an intermediate agent between the user and the application and is responsible for the interaction between them. In fact, the lion share of this document will be about dialogue management. Here it suffices to give a brief overview of its main functions.

The dialogue manager operates on a meaning representation, modelling what the user (presumably) has said. On the basis of this information, the dialogue manager can do a number of things. It has direct access to the application, and thus, for instance, has the option to change the state of the application (in the case of voice control) or to retrieve a piece of data from the database (in the case of an information service). In addition, the DM can send messages to the user. These messages can be of different kinds: the DM can provide feedback about the current state of the system, it can ask questions for further information or it can present information. Usually, the DM module produces a symbolic meaning representation, containing a specification of the form and the content of the message that should be conveyed. The Natural Language Generation module, discussed below, takes this symbolic representation as its input and converts into natural language.

In general, the dialogue manager is responsible for the *dialogue strategy* that the voice interface uses. Usually, the kind of application, and thus the kind of voice interface, codetermines the dialogue strategy. For example, IVR systems, with their limited vocabulary, tend to employ *rigid*, *system initiative* dialogues. That is: the system asks very specific questions, and the user can do nothing else but answer them. Of course, such a strategy is required due to the highly limited set of inputs the system can cope with. Command & control applications tend to have rigid, *user initiative* dialogues; the system has to wait for input from the user before it can do anything. The ideal of many researchers and developers is a *natural, mixed initiative* dialogue strategy; user and system would both have the possibility of taking the initiative when this is opportune given the current state of the dialogue, and the user can converse with the system as (s)he would with another human. Such a strategy is difficult to obtain in general, for at least two reasons. First, it is technically difficult, because the user should have the freedom to say basically anything at any moment, which is of course a severe complication from a speech recognition and language processing point of view. Second, apart from such technical hurdles, mixed initiative dialogue is difficult from a dialogue point of view, because it is not always an easy matter for a system to track initiative (but see Chu-Carroll & Brown, 1997) and to flexibly react to or initiate a shift in initiative.

Another ideal which is difficult to obtain but which would have a large impact on perceived user-friendliness of a voice interface is *cooperativity* (see e.g., Sadek and de Mori, 1998). Suppose the user asks "Is there an episode of Tatort at 21.00 this evening?" If there is no Tatort at 21.00 this evening, then "no" would be a perfect answer. But now suppose that there is a Tatort starting at 22.00, or that there is an episode of Derrick instead. In such cases it would be highly cooperative of the system to inform the user of this. For this to work, the system should have some idea of the underlying intention of the user's query; in this case,

[7] Notice that there is a certain similarity with ELIZA here: the concept spotter scans the input for known keywords, and this only works for cooperative speakers.

the underlying intention is presumably something like ‘user wants to watch a crimi tonight’. For instance, it would not be very cooperative of the system to inform the user that there was an episode of Tatort yesterday.

NATURAL LANGUAGE GENERATION (NLG) The natural language generation module takes a non-linguistic meaning representation as input and converts it into natural language.[8] In many existing voice interfaces the number of potential messages to be conveyed is limited, and moreover, often there is a one-to-one mapping between the kind of meaning representations which the DM can produce and the way they should be expressed. This means that it is possible to simply list all potential output strings, and that a straightforward string manipulation technique suffices for “generating” natural language output. In general, it is to be expected that as the range of possible inputs increases, the demands on natural language generation will increase as well (see e.g., Cole et al. 1996, Zue, 1997).[9]

AUTOMATIC SPEECH SYNTHESIS The speech synthesis module converts the generated language into speech. It is important to keep in mind that in a pure voice interface the spoken output is the part of the system that the user perceives directly and as such has a strong impact on the overall judgement of the system.[10] This implies not only that the quality should be good (understandable and preferably pleasant to listen to), but also that special care should be taken so as to select the ‘right’ synthesis method. There is a trade-off between quality and flexibility. Diphone synthesis is an example of a synthesis technique which is highly flexible, since it is based on the concatenation of very small speech segments, namely diphones (basically, a diphone is a transition between two speech sounds or phonemes). The quality of diphone synthesis is not optimal; the output is understandable but rather unnatural. One alternative method is phrase concatenation, in which pre-recorded phrases are combined to form new utterances. If done properly, the result approaches natural speech, but of course this method is rather inflexible. Unit selection is a synthesis technique which tries to combine the advantages of the two methods by allowing the concatenation of units of arbitrary length from a given corpus. For an extensive discussion of these methods see Klabbers (2000).

The choice of output technique is not only a balancing act of quality and flexibility. It has been suggested that the high quality of concatenative speech may backfire, because subjects may feel more social pressure to converse with the system if the speech output is of a very high quality (Leiser 1993). In addition, Nass & Lee (2000) have shown that even changing the parameters of a single speech synthesis system, has consequences on how subjects judge the system; if pitch range, fundamental frequency and speaking rate are high, subjects tended to judge the system as extravert, whereas if the pitch range, fundamental frequency and speaking rate are low, subjects tend to find the system more introvert.

[8] In some systems (such as OVIS), the NLG module does not output plain but *enriched* text. This is text with additional markers, indicating which words should receive a pitch accent and at which positions a phrase boundary is required. To determine placement of pitch accents and phrase boundaries, syntactic, semantic and discourse information is required, which is readily available in more advanced NLG systems (see e.g., Theune 2000).

[9] See also section 9 on the importance of having balanced interfaces.

[10] Souvigner et al. (2000:60) report on the usability studies for PADIS-XXL: *A striking (and from a technology-oriented point of view disappointing) experience was that people judged the system mainly by the speech output and did not regard speech recognition as a difficulty at all.*

4.2 *The DARPA communicator architecture*

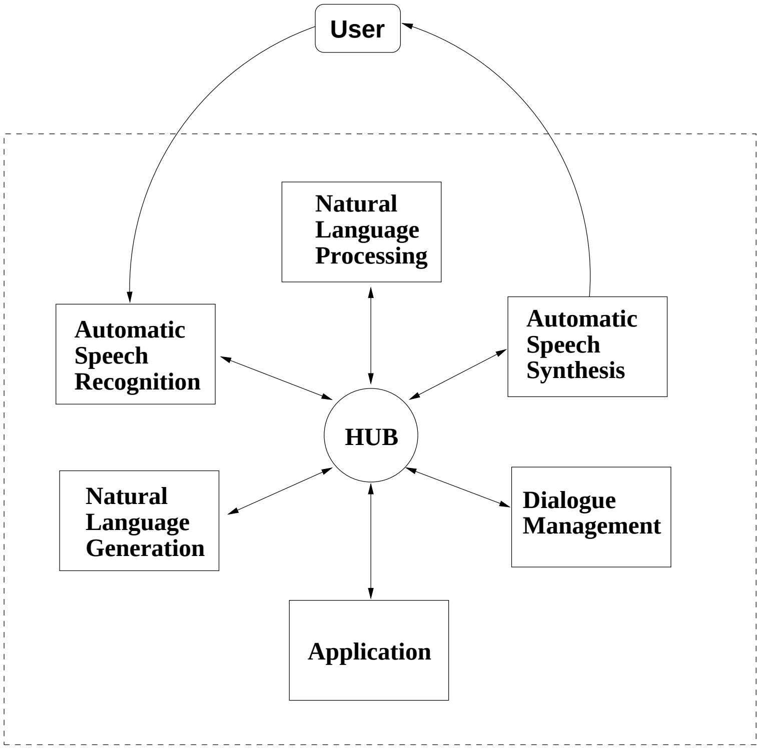
User
Natural Language Processing
Automatic Speech Recognition
Automatic Speech Synthesis
HUB
Natural Language Generation
Dialogue Management
Application

Figure 2: The Darpa communicator architecture

Recently another architecture has been proposed, the Darpa communicator architecture in figure 2 (Goldschen & Loehr 1999).[11]

The architecture is organized around a central process called the *Hub*, which is connected with a variety of *servers* performing specialized tasks (such as speech recognition, natural language processing, etc.). The Hub has thee primary functions: (*i*) it handles the message transfer between the various servers, (*ii*) it keeps track of the current state information which is accessible for all servers and (*iii*) it makes sure that an utterance is processed in the right way. Note that the flow of information does not necessarily follow a prescribed "circular pipe-line" architecture. Rather, the Hub may be 'programmed' (using so-called hub scripts) so as to start the servers in a variety of orders.[12] This becomes particularly relevant when additional servers are included the architecture (e.g., for gesture recognition and interpretation, which should probably run in parallel with speech recognition and natural language processing. In addition, if the Hub notes that communication problems arise, it may re-start earlier servers with the request to compute alternative hypotheses. But it may also initiate a new sub-dialogue with the user.

In general, the expectation is that the communicator architecture enhances the development of mobile and multi-modal voice interfaces. It is also expected to promote resource

[11] `http://fofoca.mitre.org/` (checked on 29/08/2000).

[12] The architecture of TRIPS (see section 6.1) is also centered around a hub, but crucially does not make use of hub scripts. Their motivation for not using such scripts is that is takes the initiative away from the modules themselves, and consequently makes it impossible for 'intelligent' modules to make their own decisions about which other module or service they want to use.

sharing and development at multiple sites. These expectations are based on the fact that it is relatively easy to plug-in new modules; they only need to be interfaced with the HUB, and the HUB script needs to be updated to provide for the proper flow of information through the system.

The first system which was build using this architecture is JUPITER, a telephone service for weather information developed at MIT.[13]

5 STRUCTURAL MODELS OF DIALOGUE

The previous section indicates that the Dialogue Management (DM) module is the central module of a voice interface, because it is responsible for the structure of the communication between the user and the application. This implies that every DM incorporates some kind of dialogue model. These models have to meet at least one specific criterion: they should be computationally tractable.

In this section we discuss a number of structural models to dialogue. In such approaches a dialogue basically follows a predefined path. Even though this is usually a practical approach, it is not very principled and tends to lead to relatively rigid dialogues. In the section 6 we describe a number of approaches to dialogue which are not structural, but which are driven by independent 'dialogue principles'.

Even though all the dialogue models discussed here are explicitly aimed at *spoken* dialogues, they do not pay special attention to the particular problems associated with spoken dialogue systems (in particular, the possibility of recognition errors and the resulting inherent uncertainty of the communication). The models discussed here simply assume that dealing with communication problems is just another aspect of the dialogue model. However, since dealing with communication problems is such an important aspect of voice interfaces, we shall pay special attention to it in section 7.

5.1 *Dialogue as a finite state model*

Probably the most basic way to describe a dialogue is as a *finite state model*. This is based on the idea that during a dialogue the voice interface can only be in a limited number of states and that for each state there is a limited number of actions which move the dialogue to a possibly different state. Formally, a finite state dialogue model is a quintuple $\langle \mathcal{S}, s_0, s_f, \mathcal{A}, \tau \rangle$, where $\mathcal{S}$ is a set of states, $s_0 \in \mathcal{S}$ is the initial state and $s_f \in \mathcal{S}$ is the final state, $\mathcal{A}$ is a set of actions (including the empty action, ε), and $\tau : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a transition function. This τ function determines for each state, which actions lead to which follow-up states. A dialogue can now formally be defined as a path in the state space starting in the initial state and ending in the final state.

Figure 3 contains an example of a finite state dialogue model which aims at finding out which tv channel someone wants to watch. What kinds of dialogues can this model give rise to? We enter the model via the 'start' state (s_0), and from there we immediately move to the next state (technically this is done by performing the empty action). In this state, the system asks the user which tv channel he or she wants to watch. Depending on the number of word hypotheses the speech recognition module returns, three things can happen. If the speech recognition module returns no output, for instance because the user failed to answer the question, then the system repeats the question. If the speech recognition module returns only one tv channel, say "BBC1", the aim of the dialogue has been reached and the system

[13] Call 1-617-258-0300. Or visit `http://www.sls.lcs.mit.edu/jupiter`.

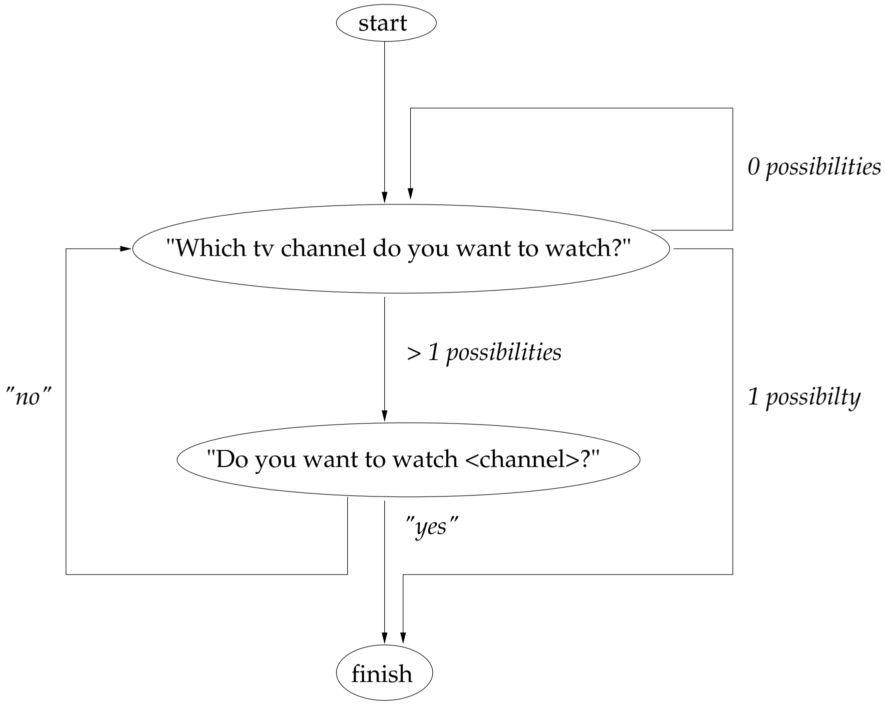
start
"Which tv channel do you want to watch?"
0 possibilities
> 1 possibilities
"no"
1 possibilty
"Do you want to watch <channel>?"
"yes"
finish

Figure 3: A simple finite state dialogue model

moves to the final, exit state (s_f). If a non-empty set of possibilities is returned, for instance {"BBC1", "BBC2"}, the system asks whether the user wants to watch one of these (e.g., "Do you want to watch BBC1?"). If the speech recognizer returns "yes", the dialogue has also reached its goal. If the recognized answer is "no", the system repeats its initial question.

It is worth stressing that even though this is an extremely simple example, it can already describe an infinite number of dialogues. But the model should obviously not be implemented in this way; for instance, it does not take the possibilities of misrecognitions into account, and if the speech recognizer returns more than one possibility, it would probably be more efficient to iterate through the list of possibilities than to repeat the initial question if the first possibility is rejected. However, extending the finite state model along these lines is not a difficult matter.

Advantages and disadvantages

The idea to describe dialogue as a finite state model has a number of advantages. First of all, finite state models are formally well-understood and computationally attractive (e.g., Larson 1992). Second, as figure 3 illustrates, it is conceptually very simple. One can imagine designing a voice interface by, starting with the start state, considering what could possibly happen in that particular state and iterating this process for all consecutively created dialogue states. This process, often referred to as *design by intuition*, is even enhanced by the existence of various toolkits for finite state methods. The most relevant one for current purposes is the CSLU toolkit (see e.g., Sutton et al. 1998) developed at the OGI, the Oregan Graduate Institute of Science and Technology.[14] The CSLU toolkit is a general toolkit for

[14] `http://cslu.cse.ogi.edu/toolkit`, checked 28/08/2000.

the development of voice interfaces. It integrates most of the standard components required for voice interfaces: two speech recognition engines (one based on neural networks, one on hidden markov models), a so-called semantic parser (an advanced kind of concept-spotter), the Festival speech synthesis engine (from Edinburgh, see e.g., Black and Taylor, 1997) and various animated characters (most notably, Baldi, developed at UCSC, see e.g., Massaro et al. 2000). In addition, it contains a *rapid application developer*, which allows the user to specify the dialogue as a finite state model using a drag-and-drop interface. The CSLU toolkit has been used by children in elementary school to make simple voice interfaces.

However, when moving beyond the simplest applications, it soon becomes clear that the strategy of defining all the possible states in which the dialogue can be, as well as all the transitions between these states, is very cumbersome. First of all, the method quickly leads to an explosion of states and transitions. More seriously, the resulting dialogues are very rigid; in a given state, the system can only accept a predefined number of actions. This implies, among other things, that it is difficult to deal with recognition errors which, almost by definition, cannot be anticipated. A different limitation is that the finite state methodology is completely domain- and application dependent: porting a finite state dialogue model to a new domain or application, *de facto* amounts to developing a completely new finite state model. One reason for this is that there is no systematic distinction between the task (what the dialogue manager want to achieve) and the dialogue strategy (how the dialogue manager proceeds towards its goal). A final limitation is related to the ”design by intuition” strategy that are commonly used for the development of finite state models of dialogue. The problem with this strategy is that it is generally very difficult to predict which situations the voice interface may encounter when really being used. The only way to find out is through extensive experimentation, and iterative development.

These limitations are well-known and broadly two kinds of alternative approaches have been investigated. One is still based on finite state methods, but either tries to describe them at a higher, more general level (see section 5.2) or tries to enrich them with probabilistic reasoning (see section 5.3). The other approach takes a different perspective and focussed on the goal directedness of dialogue. On this approach, either explicit plans are constructed how to reach this goal (section 6.1, or the underlying communicative principles are formalized and the dialogue process is a consequence of applying these principles (section 6.2). As we shall see, different approaches are useful for different kinds of voice interfaces, and each approach has its own its own advantages and disadvantages. The problems with iterative design are discussed in 5.3, where an alternative is explored, namely automatic learning of dialogue strategies.

5.2 Dialogue using a slot filling metaphor

A number of problems of the finite state model of dialogue can be solved using the *slot filling metaphor*. This is probably the most common method used for practically oriented voice interfaces, and consequently we shall pay special attention to it here. This approach is particularly useful in situations where the voice interface needs to obtain a number of pieces of information (in terms of the metaphor: fill a number of ”slots” with these pieces of information) from the user to do database look-up or to perform an action. For example, a train time table information system can only provide information about a definite train journey. For that purpose, it needs to know the required departure and arrival station and the time and date of travel. Once it has these pieces of information, it can look in the underlying database for the requested information. Similarly, a spoken TV guide can only make suggestions for viewing if it knows something about the current preferences of the user. In such cases, there is a potentially large set of *a priori* possibilities (each day, there are thousands of train journeys one can make, and not much less tv programs one might want to watch). The task of the dialogue manager is to come up with a number of constraints which select the right one(s). A dialogue strategy now corresponds with a sequence of questions about slots (or attributes) obtaining the relevant pieces of information (the slot fillings, or values). Notice that in this way the task and the dialogue strategy are separated; the task is to fill the slots, and various strategies can be used to achieve this. Since these strategies are now independent of the contents of the slots, they can be reused when the system is ported to a new domain. Notice that when there is a finite set of slots, and for each slot a finite set of values (and this typically is the case), the underlying dialogue model is still a finite state model, but the states and the transactions between them are not made explicit.

General slot-filling dialogue models are described by e.g., Aust et al. (1995), Pierracini et al. (1997), Veldhuijzen van Zanten (1999, 2000), Souvigner et al. (2000) and Langley et al. (1999). The latter contains a particularly concise way of formulating the basic algorithm, which we shall consider in some detail here. The application Langley and co-workers describe is a ”restaurant advisor”, here we take the spoken TV guide as our running example. A spoken TV guide, which may recommend programs for viewing, needs to have a database of program descriptions. A program can be represented as a feature structure, that is a collection of attribute-value pairs (or properties). Typical attributes for TV programs are program name, channel, duration, start time, kind of program etc. A simple example is given here:

$$\begin{bmatrix} \texttt{program} & \texttt{Teletubbies} \\ \texttt{channel} & \texttt{Nederland 3} \\ \texttt{broadcaster} & \texttt{teleac} \\ \texttt{date} & \texttt{23/10/2000} \\ \texttt{start} & \texttt{16.00} \\ \texttt{duration} & \texttt{25 min.} \\ \texttt{genre} & \texttt{child TV} \end{bmatrix}$$

It is possible that certain attributes take a feature structure as value themselves (for example, one could describe genre in terms of different properties). In this way, a hierarchy of slots emerges.

During the interaction, the spoken TV guide will try to find out a number of requirements of the programs that the user may want to watch, and on the basis of this a number of recommendations can be made. At each stage in the conversation, we can keep track of the user’s wishes using a *partial* description of items. For instance, the system may know that the user wants to watch a movie today, but does not yet know what kind of movie nor at what time. This is modelled by a description of the following kind: $\texttt{genre = movie} \wedge \texttt{date = today}$. With each partial description D there is a set associated of TV programs which match it (notation: $\texttt{match}\ (D)$). In the case of the current example, this is the set of movies which are on today.

The particular dialogue strategy described here is aimed at arriving as soon as possible at partial description which is matched by 1 to 3 items which can subsequently be presented to the user. To do this in an efficient way, the system has to keep track of information about attributes that have not been asked so far, attributes that the user has indicated are undesired or conversely are so important that they should be fixed, etc.

```
INIT       Let Descr be an empty description
           Let Unasked be all possible attributes
           Let Asked be the empty set
           Let Undesired be the empty set
           Let Fixed be the empty set
           Let Rejected be the empty set
           Let Recommended be the empty set
           Welcome the user

CONSTRAIN  If Match(Descr) $> 4$
           Then let Attribute be Select-Constrain (Descr, Unasked, Undesired)
               Remove Attribute from Unasked
               Add Attribute to Asked
               Ask-Constrain (Attribute)

RELAX      If Match(Descr) $= 0$
           Then letAttribute be Select-Relax (Descr, Asked, Fixed)
               Ask-Relax (Attribute)

PRESENT    If $0 <$ Match(Descr) $< 4$
           Then let Recommend be Select-Candidate (Descr, Rejected)
               Recommend-Item (Recommended)
```

Figure 4: Control rules for presenting questions to the user in a slot-filling dialogue, after Langley et al. 1999

The four rules in figure 4 determine the kind of question that the system may ask, depending on the number of items which meet the description under construction. INIT initialized the interaction. CONSTRAIN tries to add constraints to the description under construction when the set of matching items is still too large to be presentable to the user. It does so by selecting an attribute from the set of attributes which have not been asked to the user and which are not known to be undesired by the user. It can happen that the description is so constrained that no program matches the required specification. In that case the system will try to relax the description by proposing to the user to leave out an attribute from the description (though not an attribute that in one of the previous turns has been rejected as a candidate for relaxation by the user). Finally, if there is a highly limited set of items which match the description, one of these is selected and recommended to the user (but not if in one of the previous turns the user has rejected this particular recommendation).

The rules in figure 5 are responsible for handling user inputs.[15] Rule R1 handles situations in which the user answers a question with some value, thus adding a constraint to the partial description under construction. Rule R2 operates when the user has rejected an attribute. Where rules R1 and R2 are concerned with constraining, rules R3 and R4 deal with

[15] Langley et al. (1999) contains four additional rules, not shown here. One covers situations in which the user rejects the current attribute under discussion or re-addresses an attribute discussed earlier. Two other rules allow users to ask questions about the set of attributes the system knows or about the set of values which are allowed for a given attribute. A final rule states that when there is no Response the system has to wait for one.

what happens when a user accepts c.q. rejects a proposal for relaxing the partial description. The last two rules operate in an analogous fashion, but now for acceptance and rejection of a recommendation.

R1 If `Response` is `Answer-Constrain(Attribute, Value)`
Then add `Attribute = Value` to `Descr`.

R2 If `Response` is `Reject-Constrain(Attribute)`
Then add `Attribute` to `Undesired`,
Remove `Attribute` from `Asked`.

R3 If `Response` is `Accept-Relax (Attribute)`
Then remove `Attribute = Value` from `Descr`,
Remove `Attribute` from `Asked`.

R4 If `Response` is `Reject-Relax (Attribute)`
Then add `Attribute` to `Fixed`.

R5 If `Response` is `Accept-Item(Recommended)`
Then say farewell to the user.

R6 If `Response` is `Reject-Item(Recommended)`
Then add `Recommended` to `Rejected`.

Figure 5: Control rules for handling user responses in a slot-filling dialogue, after Langley et al. 1999

[begin digression] One important open issue is how to select an attribute from the set of `Unasked` attributes, for which the user is subsequently asked to provide a constraining value. One option is asking a question about the attribute that provides the most information and thus leads to the most drastic reduction of uncertainty in the set of items C which match the partial description constructed so far (i.e., the attribute which rules out the most alternative candidates still available in C). For this, Langley et al. propose to use the entropy measure (well-known from information theory, Shannon 1948). Formally, they select the attribute a which minimizes, for the random variable c of items in the set C, the entropy measure:

$$H(c|a) = -\sum_{v_j \in a} \sum_{c_k \in C} P(c_k, a = v_j) \log_2 P(c_k | a = v_j),$$

where c_k ranges over the items in the set C and v_j is a potential value for attribute a. The probability $P(c_k, a = v_j) = P(c_k)$ if item c_k matches the revised partial description and is 0 otherwise. The probability $P(c_k|a = v_j)$ can be calculated as follows:

$$P(c_k|a = v_j) = \frac{P(c_k)}{\sum_{c_j \in C'} P(c_j)},$$

where C' is the set of items which match the revised description that includes $a = v_j$. To simplify, we may assume that all items are equally likely. Thus,

$$P(c_k) = \frac{1}{|C|}.$$

Using this entropy-based measure, the function `SelectConstrain` can calculate for each attribute the amount of information to be gained by asking it, selecting the one with the lowest entropy, making a random selection in the case of equally informative alternatives. This already has some nice consequences. For instance, if the system has the choice between asking a question that divides the remaining items in two roughly equally sized groups and one that results in a highly unbalanced split, the first one is chosen. **[end digression]**

This is a concise and neat way of formalizing a slot-filling algorithm, and an approach such as this one, modulo some relatively minor changes, can be found in various practically oriented voice interfaces. The description above is still lacking in two respects. First of all, there are no rules to deal with communication problems. This is due to the fact that Langley et al. (1999) use this dialogue strategy in a graphical user interface. Thus, there is no danger of speech recognition or interpretation errors and hence there is no necessity to deal with them. However, incorporating verification rules in the previously outlined dialogue strategy is relatively straightforward.[16] The described strategy also does not specify what *form* the system question should take. For instance, should it be an open question ("What kind of program do you want to watch?), an alternative question ("Do you want to watch a thriller or a comedy?") or a yes/no question ("Do you want to watch a thriller?")? Veldhuijzen van Zanten (1999, 2000) presents an interesting solution to this problem which crucially involves the slot-hierarchy; if the communication is running smoothly, the system can ask high-level, open questions. In the case of communication problems, the dialogue manager can zoom in and ask more specific questions. This approach is fully compatible with the approach of Langley et al. 1999.

Advantages and disadvantages

The slot-filling approach to dialogue management effectively circumvents a number of problems associated with the finite state approach to dialogue. It does not suffer from an explosion of possible states, since states are no longer made explicit. There is an elementary distinction between the task and the dialogue strategy, which enhances portability. If, instead of an electronic TV guide we want to develop an air travel information system we only need to make some minor modifications of the rules presented in figure 4 and 5. For such an application we are only interested in *one* solution (one plane connection). All this makes the slot-filling approach the most practical of dialogue management techniques described here and as such it is not surprising that it is the most frequently used one in practical systems. Again, tools exist, most notably Philips' HDDL. This is the *dialogue description language* which is used in SpeechMania. The advantage of this particular system is that, just like the CSLU toolkit, it is a full package allowing for the development of complete, integrated voice interfaces. It is presumably not usable for children in elementary school, but a one-week specialized course offers the basic knowledge for developing voice interfaces using SpeechMania. The resulting voice interfaces are more advanced and robust than those developed

[16] For instance, an explicit verification strategy can be incorporated as follows: add a fifth rule VERIFY to figure 4 which states that if an response is still unverified, it should be verified using a yes-no question. In addition, the rules in figure 5 should now only apply to verified responses, and additional rules need to be added to handle responses to verification questions.

using the CSLU toolkit.[17]

The slot-filling approach also inherits a number of limitations of the finite state approach. It is arguably less rigid, but still the resulting dialogues are not very flexible. It is an open question how scalable the approach is. Most applications based on the slot-filling metaphor are concerned with only a handful of slots. Finally, even though task and dialogue strategy are separated, which is beneficial for portability, the dialogue strategies themselves are still hand-crafted. Developing a new dialogue strategy implies the development of a new set of rules.

5.3 Stochastic Dialogue Management

There is no standard way to build a voice interface. Currently, most systems rely on hand-crafted rules, primarily based on the designer's intuitions and implemented in a trial-and-error way. As noted above, this approach has at least two problems: (*i*) the systems are very much domain and application dependent; and (*ii*) it is generally very difficult to predict in advance which situations the system may encounter when really being used. To reach a good level of performance it is therefore necessary to do an iterative design and perform extensive evaluations of each successive version of the system.

One recent, promising approach to tackle these problems is based on the data-driven approach which has proven to be so successful for speech and language technology. The idea is to formalize the dialogue process in a mathematical way and use machine learning techniques to find optimal dialogue strategies on the bases of dialogue data, see e.g., Singh et al. (2000), Roy et al. (2000), Young (2000), Litman et al. (2000b) and Levin et al. (2000).. In this section we describe this data-driven approach to dialogue, primarily following Levin et al. 2000. Levin and co-workers show that their approach works for both the finite state approach using a tutorial day-and-month dialogue system, and for the slot-filling approach (using the ATIS (air travel information systems) task.[18] Here we focus on the former.

Consider a simple voice interface which asks the user for the current date (day and month). In terms of the slot-filling metaphor: the system tries to fill two slots: one for the day (d) and one for the month (m). The underlying finite state model can be described as follows:[19] we have a set $\mathcal{A}$ of actions consisting of three questions (*day?*, *month?* and *date?* asking for the current day, month and date respectively) and a closing action (*bye!*). We have a set $\mathcal{S}$ of 411 states. This includes the initial state s_0 in which both the day and month slots have no value ($d = 0, m = 0$) and the final state s_f (with $d = -1$ and $m = -1$). There are 12 states where the month variable has a value but the day variable does not and 31 states where the opposite holds. Moreover, there are 366 complete dates. Finally, we have to specify the transition function τ, which specifies, for each state, what is the next action to be invoked. Different dialogue strategies correspond with different transition functions.

Figure 6 depicts three possible strategies (there are more options) : in strategy 1 the system immediately ends the dialogue, in strategy 2 the system first asks for the date (using a single action) before terminating the dialogue, in strategy 3 the system first asks for the day, then for the month and finally terminates the dialogue.

[17] From a dialogue perspective a disadvantage of SpeechMania is that everything is intertwined to such an extent that modular development is impossible. Typical SpeechMania rules integrate information about speech recognition, verification, utterance generation *and* dialogue management.

[18] Interestingly, automatically learned optimal dialogue strategy for this task is exactly the one of Langley et al. (1999) just described.

[19] Remember that a finite state model is fully specified by a quintuple $\langle \mathcal{S}, s_0, s_f, \mathcal{A}, \tau \rangle$, see section 5.1.

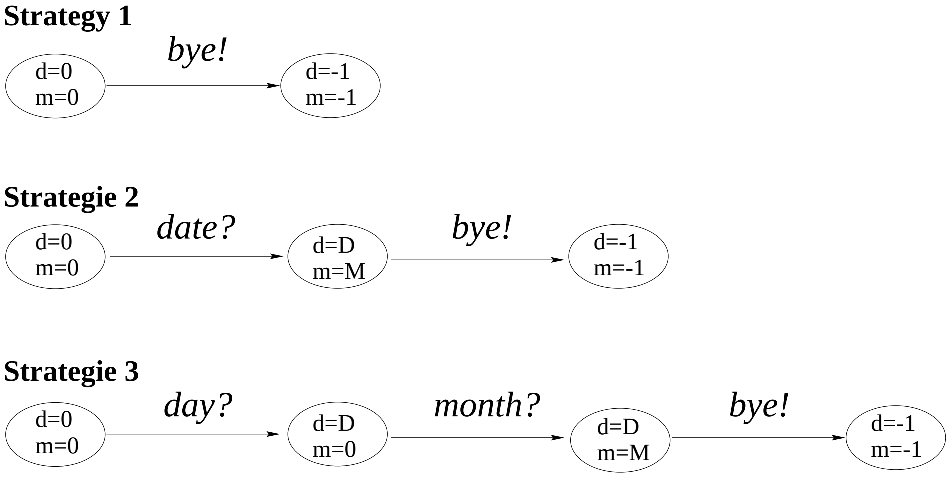

Figure 6: Three possible strategies for the Day-and-Month dialogue

The question now arises which of these strategies is the best one. The answer is: that depends on the (derived) goal of the system. Let us assume that the goal is to get the right day and month values from the user as quickly as possible. On the basis of this goal, an objective function C can be defined which measures the costs of a particular dialogue. Three terms are relevant: the number of turns (N_t), the number of errors (N_e) and the number of missing values (N_m). Each term is associated with its own weight (w) indicating the relevant importance of the particular terms.

$$C = w_t \langle N_t \rangle + w_e \langle N_e \rangle + w_m \langle N_m \rangle$$

On the basis of this cost function, we can determine the costs associated with the three strategies in figure 6. Strategy 1 takes only one turn, there are no errors, and there are two missing, hence $C_1 = w_t + 2.w_m$. If we assume that the chance for misrecognitions following a complex question like *date?* is P_1, then the costs associated with the second strategy are $C_2 = 2.w_t + 2.P_1.w_e$: the dialogue takes two interactions, there may be 2 recognition errors (with probability P_1) and there are no missing values. Finally, let us assume that the probability of speech recognition error following a simple question like *day?* or *month?* is P_2, then the costs associated with the third strategy are $C_3 = 3.w_t + 2.P_2.w_e$. Now an *optimal* strategy can be defined as a strategy which minimizes the costs.[20] For example, strategy 1 is more efficient than strategy 3 when the likelihood of recognition errors is too high.[21] Similarly, strategy 3 is more efficient than strategy 2 when the difference in error probabilities justifies an extra turn, that is, when $P_1 - P_2 > w_t/2.w_e$.

In sum: given an objective cost-function, it is possible to determine the costs of a given strategy and compare it with the costs of other strategies. However, since the number of potential strategies is exponential (there are $\|\mathcal{A}\|^{\|\mathcal{S}\|}$ possible strategies), it would be useful to have a method for automatically finding the optimal strategy. Fortunately, there is a

[20] In the literature one also encounters *reward functions*, where an optimal strategy is one that maximizes the reward.

[21] To see this, consider when the costs associated with strategy 1 (C_1) are less than those of strategy 3 (C_3): $w_t + 2.w_m < 3.w_t + 2.P_2.w_e \Longrightarrow 2.w_m - 2.w_t < 2.P_2.w_e \Longrightarrow w_m - w_t < P_2.w_e \Longrightarrow (w_m - w_t)/w_e < P_2$.

mathematical model, closely related to the finite state model, which allows for the automatic determination of the optimal strategy, namely *Markov Decision Processes* (MDPs).

[begin digression] An MDP differs in two essential respects from the finite state model. First of all, instead of a transition function τ, we specify transition *probabilities*. That is: when at time t, an action a_t is taken in state s_t we move to state s_{t+1} with probability $P_T(s_{t+1}|s_t, a_t)$. Notice that the probability of moving to state s_{t+1} *only* depends on the previous state and action, and not on what might have happened before time t. This property is known as the *limited horizon* Markov property.[22] The second addition concerns the stochastic modelling of costs. If in state s_t action a_t is performed, this costs the system c_t, with probability $P_C(c_t|s_t, a_t)$. The cost distributions need to be assigned in such a way that at the end of dialogue the accumulated costs will be equal to the objective cost function C. Thus:

$$\sum_{t=0}^{t=T_f} c_t = C$$

where T_f is the moment in time when the final state is reached ($s_{T_f} = s_f$). Spelling out the cost distributions is relatively straight forward now. Any time a question is asked (*day?*, *month?* or *date?*), the system incurs a constant cost w_t with a probability of 1: $c(s, \textit{day?}) = c(s, \text{month?}) = c(s, \text{date?}) = w_t$, for any state s. Furthermore, when the dialogue is closed (using the *bye!* action), this costs $w_t + w_e\langle N_e\rangle + w_m\langle N_m\rangle$, where N_e is the number of errors and N_m is the number of missing values.

Now, we can calculate the best action to be taken from a state s: is will be that action a which minimized the expected costs. More precisely, the best action to be taken in a state s, notation $V^*(s)$, is that action that minimizes the sum of the directly incurred costs and the expected costs for the next state using the best action available from that state:

$$V^*(s) = \min_a[c(s, a) + \sum_{s'} P_T(s'|s, a).V^*(s')]$$

The optimal strategy π^* can be computed simply as the chain of actions which minimizes the expected costs. The optimal value function V^* is unique, and can be calculated using standard *value iteration* techniques (see Sutton 1991), provided that all the parameters of the model are known.[23]

However, in the context of voice interfaces, not all model parameters (transition probabilities and cost distributions) will be known in advance. In that case the parameters can be obtained automatically using *reinforcement learning*. Using this method, the optimal strategy is learned through interactions. For this to work, a large number of interactions are required (typically in the order of tens of thousands, depending on the number of states and the number of actions). Clearly, it is not feasible to collect these interactions with real users. Therefore, Levin et al. (2000) propose an alternative: using a *simulated user*. A simulated user is simply a stochastic process that generates replies with a certain probability

[22] Sometimes we do want to keep track of history. For example, sticking to our date-example, it seems reasonable to assume that the chances of misrecognition depend on the kind of question; a simple question like *day?* is assumed to have a lower error rate than a complex question like *date?*. Therefore we want to know what kind of action was performed to fill the day d or month m slot. This can quite easily be done by introducing additional variables q_d and q_m which are set to 1 if the corresponding slot was filled using a simple question and to 0 of it was filled using a complex action.

[23] And provided that the state space is finite.

in response to dialogue acts. The probabilities can be estimated on the basis of a corpus of complete, annotated dialogues.[24]

One way to use reinforcement learning to learn the optimal strategy is known as "Monte Carlo with exploring starts" (see Sutton and Barto 1998 for an overview). The goal of this approach is to estimate the optimal state-action value function, denotated as $Q^*(s, a)$. which is defined as the expected costs of a session starting in s and moving to state s' with action a, and proceeding in the optimal way until the final state is reached.

$$Q^*(s, a) = c(s, a) + \sum_{s'} P_T(s'|s, a).\text{min}_{a'}Q^*(s', a')$$

(And notice that $V^*(s) = \text{min}_a Q^*(s, a)$.) This algorithm is an iterative algorithm, starting with an arbitrary setting of $Q^*(s, a)$ and iteratively improving it. In each iteration, the algorithm explores for each state s and action a what the costs are of a dialogue session which starts in state s with action a and proceeding with the current estimation of the state-action function Q^* until the final state has been reached. The algorithm will quickly learn that it is generally very expensive to immediately terminate the dialogue using the *bye!* action. Levin et al. (2000) show that, as the number of iterations in the reinforcement learning algorithm grows, the learned strategy converges to the optimal one. **[end digression]**

Advantages and disadvantages

The use of reinforcement learning for finding the optimal strategy great potential for the development of voice interfaces. The procedure provides a quantitative criterion for developing good interfaces, using well-understood mathematical concepts. The task of designing an optimal dialogue strategy is taken over by the computer; there is no need for a dialogue expert during development. Moreover, the approach is essentially independent of the task and the application: all that is needed for developing a new application, is a new corpus of dialogues or a simulated user tailored to the new application and task. Of course, collecting a corpus of dialogues is generally a costly undertaking. However, devising a new simulated user may be done quite effectively. Recent work on reinforcement learning suggests that it may not be necessary to estimate the parameters of the simulated user in a precise way from a corpus, since it has been shown that only a very low accuracy of the transition probabilities is needed for finding a good approximation to the optimal strategy (Kearns and Singh, 1998). Finally, it has been shown (Litman et al. 2000b) that the approach can lead to a measurable improvement of the performance in an experimental system. It seems likely that stochastic dialogue management will be an active area of research in the near future.

Nevertheless, there are various open questions. So far, the learning approaches have been applied to a carefully hand-crafted state space. Naturally, the choice of states has a strong influence on the learned strategy: a state which is not present in the state space will never be visited using the learned optimal strategy. One way to solve this limitation is by devising methods for learning the state space. A second question concerns the objective function. Clearly, the objective function is the main determinant of the learned, optimal strategy. However, determining the objective function is not always straightforward. The terms may be rather straightforward, but how to assign weights to them? And how to deal with highly subjective features such as user-satisfaction? These questions are related to the

[24] Notice that it is not possible to use this corpus of dialogues directly for learning the optimal strategy; the system would only learn the strategy that is already implicit in the dialogues themselves.

general problems associated with evaluation of voice interfaces. We discuss this issue in section 10. A more principled problem concerns the notion of an optimal strategy. Is it plausible that there is *one* optimal strategy which suits everyone? It may well be that a strategy which is optimal for an expert user differs from a good strategy for a novice user, just like 'sheep' and 'goats' may be served by different strategies.[25] In general, it seems likely that some people have a subjective preference for strategy A, while other may prefer strategy B. This points in the direction of a "meta-strategy", which given particular user characteristics learns the optimal strategy. Whether such meta-strategies are indeed required and how they should be learned is an open question.

6 NON-STRUCTURAL MODELS OF DIALOGUE

6.1 *Dialogue as a goal directed process*

So far, the dialogue models we discussed are all based, in one way or another, on a finite set of states and a finite set of actions, and the dialogue model attempts to couple states and actions in an efficient way. The actual dialogue always has to follow some predefined path. Such models have particularly been used for information dialogues and for command-and-control applications. However, for other kinds of dialogues their usefulness is less straightforward. One class of dialogues for which this is the case are so-called *task-oriented dialogues*; that is, dialogues about a task that is carried out during the interaction. Typically, in such dialogues, the system and the user need to cooperate to achieve a certain goal, generally that of solving a problem. An example would be a voice interface to a VCR, which helps the user in programming a program. For such applications, the user is assumed to have insufficient knowledge to solve the problem alone, while the system has complete knowledge of the task. Prime examples of plan-based dialogue systems are the *circuit fix-it shop*, described in detail in Smith & Hipp (1994), and the Trains/Trips[26] systems developed in Rochester (see e.g., Allen et al., 1995, 2000, Traum et al. 1996). Both approaches are firmly based on the planning approaches known from ('old') artificial intelligence (AI). The circuit fix-it shop helps users in repairing electronic circuits, the Trains/Trips systems help users with the construction of a plan (typically in a logistical setting). Note that these are tasks for which a slot-filling metaphor seems inappropriate.

Grosz & Sidner (1986) have argued that in task-oriented dialogues the structure of the task co-determines the structure of the dialogue. In general, a task typically consists of a sequence of sub-tasks. Consequently, a task-oriented dialogue consist of a sequence of sub-dialogues, each addressing a specific sub-task. Consider the task of programming a VCR. To perform this task, one has to perform a sequence of subtasks (insert a video tape, select the channel which broadcasts the relevant program, enter the start and end time, etc.). A voice interface which helps the user programming his or her VCR has to address each of these subtasks, in the correct order, to complete the main task successfully. During the task, both the system and the user have various goals, which are aimed at achieving certain states or which trigger certain actions. The actions are a complicating factor here, because they typically lead to a change of the 'state of the world'. For example, initially there may be no tape in the VCR, while after completion of the first subtask a tape has been inserted. This means that the system has to be able to track changes in the outside world and take

[25] This terminology derives from Doddington et al. 1998: 'sheep' are people who are generally well recognized by automatic speech recognizers, while for 'goats' it is much more difficult to be recognized.

[26] `http://www.cs.rochester.edu/research/trains/`, checked on 28/08/2000.

the repercussions of the changes for the ongoing dialogue into account. (If the system were not capable of detecting changes in the outside it world, it would continue instructing the user to insert a video tape.) In addition, the system also has to have knowledge about the competence and knowledge of the user, since these have a clear influence on how the task at hand should be tackled.

The *circuit fix-it shop*, developed and described by Smith and Hipp, provides an interesting view on how these different issues can be integrated in a single voice interface. One of the main insights of Smith and Hipp is that the interface should be able to *reason* about the task, the application and the user in an integrated fashion. Accordingly, the core of their system is an automated reasoner, a *theorem prover*, that is: a module that is capable of deriving logical conclusions from a set of premisses. The theorem prover is used primarily to determine whether a certain (sub-)goal has successfully been completed. The approach is as follows: completeness for a subgoal is defined by a theorem. The system now determines whether the subgoal has been accomplished by automatically trying to find a proof of the theorem, using the axioms in its knowledge base (axioms are logical expressions which are 'assumed' to be true and thus do not require a proof themselves) and the laws of logic.

Consider the following (rather simplified) situation: the theorem is "the VCR has been successfully programmed" (this proposition is represented as p). Suppose that the system's knowledge base contains the following two axioms: "if the red light is on, all pieces of information have been entered" (represented as $a \rightarrow b$) and "if all pieces of information have been entered, the VCR has been successfully programmed" ($b \rightarrow p$). On the basis of these two axioms, the system can infer $a \rightarrow p$ ("if the the red light is on, the VCR has been successfully programmed"), but not the theorem p itself. Of course, it would be able to prove the theorem if the system had access to the information that "the red light is on". In other words, the theorem p could be proven to be true if the axiom a were not missing from the knowledge base. The key-insight of Smith & Hipp is that missing axioms require interaction with the user (they dub this the *missing axiom theory*); the system should just ask whether a is true or not (system: "Is the red light on?"). A positive answer from the user is taken to imply that the system may add a as an axiom to its knowledge base and subsequently can complete its proof of p. Notice that this approach requires an "interruptible theorem prover"; a theorem prover which can suspend its proof construction when it encounters a missing axiom and can wait for input from the user.

The flexibility of the interruptible theorem prover is beneficial for other things as well, for instance for the management of subgoals. How should a new subgoal be selected? Smith & Hipp propose to choose the goal which has the highest probability to lead to success given the current state of the dialogue. Now, it may well be that during the interaction another subgoal suddenly seems more promising (for instance, because the user takes the initiative and provides important information regarding a different goal than that which the theorem prover is working on). In that case, the theorem prover should be able to abruptly switch to that subgoal. In a similar vein, Smith & Hipp also keep track of the competence and knowledge of the user in a user-model. Interestingly, information about the user is also stored in the form of axioms, which provides a seamless integration of information about the user in the proof under construction.

While the circuit fix it shop is primarily a practical approach, the Trains/Trips projects carried out at the University of Rochester are mainly a long term research vehicle. Various demo-systems have been developed, all concerned with interactive plan building; that is, system and user cooperate on the construction of a (logistical) plan. The various Trains demonstrators are concerned with finding efficient routes for trains in the north-east part of the United States. This is a ”very simple task” (Allen *et al.* 2000), typically involving three trains. Based on the experience with Trains, a new dialogue architecture was developed in the second part of the eighties (not unlike the communicator architecture discussed in section 4.2), resulting in TRIPS (The Rochester Interactive Planning System). With TRIPS various demo systems have been developed for tasks like planning the evacuation of a group of people from an island threatened by an impending hurricane or the coordination of emergency vehicles in response to a simulated 911 call.

Trains/Trips continues in the well-established AI tradition of formalizing speech acts using traditional planning mechanisms. The notion of a *speech act* is due to Austin (1962), and further developed by Searle (1969). Speech act theory essentially claims that utterances should be treated as *actions*. Thus by uttering a sentence a speaker performs an action (e.g., inform, request, threaten, commit, demand, etc.). Cohen and Perrault (1979) and Allen and Perrault (1980) show that speech acts can be formalized by specifying necessary and sufficient conditions (felicity conditions in the terminology of Searle 1969) for the performance of a specific speech acts which, when executed, have certain effects. The conditions and effects are primarily defined in terms of the beliefs, desires and intentions of speakers and hearers. How utterances relate to the beliefs, desires and intentions is described in the BDI model of Bratman *et al.* (1988), see Figure 7.

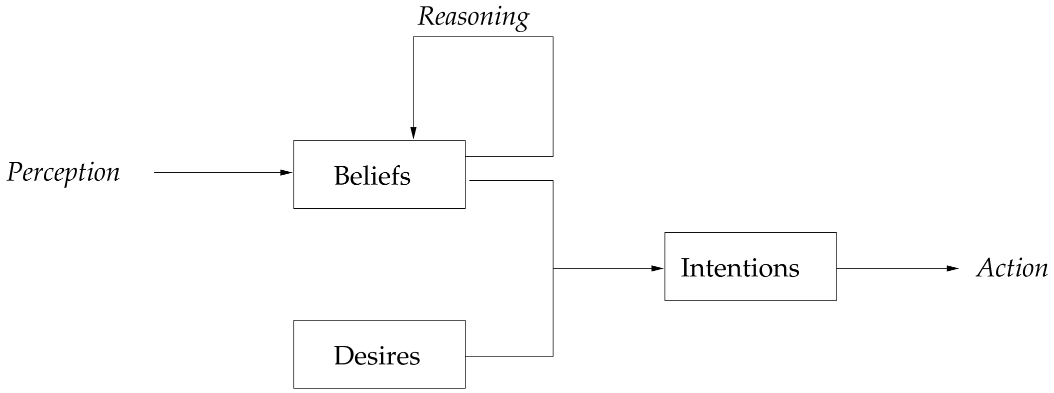

Figure 7: The Beliefs, Desires and Intentions (BDI) model

According to the BDI model, an agent has beliefs about the current state of the world, and in addition the agent has certain desires about how it would like the state of the world to be (usually, this set of desires is predetermined). On the basis of its beliefs and desires, the agent selects a goal to achieve; an intention. Typically, the agent then constructs a plan (consisting of a sequence of actions) which it believes will result in achieving its goal. Once a plan has been constructed, the agent can initiate the execution of the plan by executing the initial actions. Typically, these change the state in the world. As the agent makes observations about the changing world, this will generally lead to a modified set of beliefs, called plan motoring in AI. Given these new beliefs, it may well be that the agent no longer believes that its current plan is the most effective, and consequently it may repair its plan or even construct a new one.

This is a general action planning mechanism, but it can also be used to construct a conversational agent (as required for a voice interface). In that particular context, the beliefs and desires can typically be concerned with the beliefs and desires of the user (for instance, the system can believe that the user does not know how to program a VCR but has the intention to do so). On the basis of the current (communicative) beliefs and desires, the system may select communicative goals and construct a plan on the basis of these goals. The execution of this plan will typically consist of a sequence of speech acts, conveyed to the user. The execution of these speech acts and, more importantly, the reactions of the user will lead to a new state of the world. On the basis of these the system can update its set of beliefs (for instance, it may now believe that the user is at least partially capable of programming its VCR or even no longer has the intention to program it) and, if necessary, change its plan accordingly. The core of the TRAINS/TRIPS systems is a conversational agent of this form.

Advantages and disadvantages
The circuit fix-it shop provides an integration of many theoretical results (primarily from 80s AI), and is arguably one of the first full-fledged spoken dialogue systems (completed by 1991). Smith & Hipp (1994) is still the only comprehensive description of how a spoken dialogue system can be constructed, including details on the various algorithms, the implementation and the evaluation. Many of the central aspects of the system are still relevant today (interruptible theorem proving, user modelling, variable initiative dialogue). Even though both language and speech technology have matured considerably in the past decade (the vocabulary of the circuit fix-it shop consisted of only 125 words), the book is still a worthwhile read. It is interesting to note that in recent years there is an increased interest in 'computational logic'. In this field special attention is paid to the trade off between expressive power and computational complexity (see for instance Areces 2000). It would be highly interesting to investigate the usefulness of 'computational logic' for more advanced voice interfaces in the spirit of the missing axiom theory.

The TRAINS/TRIPS system is probably the largest (in terms of effort) and most ambitious project in the area of conversational interfaces. It has been running for at least a decade and is still very much in development. It is firmly based in the traditional planning approach, and has shown the usefulness of such techniques for conversational agents. It should be stressed that many other aspects of voice interfaces receive a detailed treatment as well; for instance, in the framework of TRAINS/TRIPS seminal work on computational models of information grounding was carried out (Traum 1994). However, given the large scale and the varying focus of attention it is difficult to obtain a general idea of the performance and general usefulness of this approach.

In general, the goal directed approach to dialogue has a number of potential advantages over approaches such as the finite state approach or the approach based on the slot-filling metaphor described above. First of all, it is difficult to see how these latter approaches could deal with the task-oriented dialogues. Conversely, the goal-directed approach described in this section could well be applied to information dialogues. Moreover, the goal-directed approach is more principled than the approaches described earlier, in that they are more (Trains) or less (circuit fix-it shop) based on a general theory of communication. Arguably this makes them less domain- and application-dependent and gives them better prospects of obtaining ”natural”, human-like communicative abilities. Finally, there is no doubt that the BDI approach to conversational agents has led to many insights on how humans communicate.

However, the traditional AI planning approach in general has its problems, and many of these carry over to the approaches described here. One of the main problems of planning is that, in its general form, it is computationally very expensive (see e.g., Kautz 1991). In addition, given that speech acts are modelled as actions with certain preconditions and effects, a number of notorious AI problems manifest themselves and should be dealt with. The most important of these is probably the *frame problem* (McCarthy 1980); how to specify which things in the world remain *unchanged* after a certain action has been executed?

Another general problem with the tradition BDI approach is that the notions of belief and intention lack proper formalizations. This makes it very difficult to model reasoning processes, or to maintain a proper balance between beliefs and intentions. Consequently, the link between beliefs and intentions and the associated actions is purely operational. Rules are defined which *state* that if an agent beliefs x and has intention y, than action z should be performed. There are no explicit, independently motivated *rationality principles* which explain why an agent performs certain acts. These issues will be described now.

6.2 Rational conversational agents

[begin digression] Even though the traditional plan-based approach to conversation comes much closer to the human communication process than the structural approaches described in section 5, they leave certain aspects of human communication out of consideration, most notably the (alleged) rationality of human behavior. Of all the computational models of dialogue discussed here, the rational approach to conversational agency is the most explicit in its aim to mimic human (intelligent) communicative behavior (cf. Turing 1950). Sadek and de Mori (1998:538): "a system capable of carrying on an "intelligent dialogue" has to be an *intelligent system*, in which the communication ability is not primitive, but is grounded on a more general competence that characterizes *rational behavior*".

The rational agency approach can be understood as a reformulation in a formal (logical) framework of the plan-based approach. The foundation of this approach is formed by the seminal work of Cohen & Levesque (1990) which aims at specifying "the 'rational balance' needed among the beliefs, goals, plans, intentions, commitments, and actions of autonomous agents" (p. 33). Sadek has provided a number of modifications and extensions of this 'rational balance' (see e.g., Sadek 1994[27]) and is one of the main advocates of the approach for voice interfaces. His views are concisely summarized in Sadek and de Mori (1998). The approach consists of a number of ingredients: (*i.*) since we want to reason about beliefs, desires/goals and intentions, these notions need to be formalized in logic, and (*ii.*) on the basis of these notions, a 'rational unit' is constructed; this is essentially a method to determine which actions should be undertaken.

First, the notions beliefs, goals and intentions need to be formalized. If an agent i beliefs a proposition p, this is denotated as $B(i,p)$. The logical behavior of the B operator is governed by a number of axioms (which have a long history in logic, see e.g., Montague 1960, and have been discussed for belief by Hintikka 1962):

$$\begin{array}{ll}
B(i,\varphi) \wedge B(i,\varphi \rightarrow \psi) \rightarrow B(i,\psi) & \text{(K)} \\
B(i,\varphi) \rightarrow B(i,B(i,\varphi)) & \text{(4)} \\
\neg B(i,\varphi) \rightarrow B(i,\neg B(i,\varphi) & \text{(5)} \\
B(i,\varphi) \rightarrow \neg B(i,\neg\varphi) & \text{(D)}
\end{array}$$

The first axiom essentially says that beliefs are closed under consequence: if an agent i be-

[27] The main difference lies in Sadek's use of his *observation principle*, which accounts for a distinction between what an agent observes from another agent and the action the latter has really performed.

lieves that φ and that $\varphi \rightarrow \psi$, then i also believes ψ. The second and third axioms are concerned with positive and negative introspection (if an agent i believes that φ is true, then i believes that he believes that φ is true (4), and if an agent i does not believe that φ is true, then i believes that he does not believes that φ is true). The last axiom states that beliefs should be consistent (thus, if i believes that φ, then it can not be the case that i also believes the negation of φ). Moreover, the following "necessitation" rule (N) applies (which states that an agent believes everything that is always true):

necessitation rule (N)
if φ is always true, then so is $B(i, \varphi)$

The resulting logic of belief is known as NKD45 (or weak S5).[28]

In a similar vein goals can be formalized; the notation $G(i, \varphi)$ is used to represent the fact that an agent i has a goal φ (that is: wants to be in state where φ is true). At this point, it is a good exercise for the reader to consider the axioms for belief given above, and determine which of those are intuitively applicable to goals. Some reflection will show that introspection (and certainly negative introspection) is not a useful property of goals.[29] Probably, all we want is to state that goals are closed under consequence and that goals are consistent:

$G(i, \varphi) \wedge G(i, \varphi \rightarrow \psi) \rightarrow G(i, \psi)$ (K)
$G(i, \varphi) \rightarrow \neg G(i, \neg\varphi)$ (4)

Moreover, the connections between beliefs and goals needs to be modelled. For example, Cohen & Levesque's "realism constraint" essentially states that an agent cannot have a goal which the agent believes to be false.

realism constraint
$B(i, \varphi) \rightarrow G(i, \varphi)$

In addition, if φ is a goal of an agent, then so are the expected consequences of this goal.

expected consequences constraint
$G(i, \varphi) \wedge B(i, \varphi \rightarrow \psi) \rightarrow G(i, \psi)$

All these rules are simple and intuitive; the basis of rational agency is elegant enough and largely uncontroversial. Unfortunately, things soon become more complex. One thing that is missing, according to Cohen & Levesque (1990) is a notion of commitment. There is no guarantee that an agent will not easily give up a goal. To compensate for this, Cohen & Levesque introduce a notion of a persistent goal (a so-called p-goal, abbreviated as PG). An agent's goal is persistent if the agent will not give up the goal until it has been satisfied or until the agents thinks that it will never be true.[30]

$$PG(i, \varphi) \text{ if and only if } G(i, (\text{Future } \varphi)) \wedge \quad (1)$$
$$B(i, \neg\varphi) \wedge \quad (2)$$

[28] Interestingly, Hintikka (1962) argued that belief is best modelled by the logic NKT4 (also known as S4). This logic omits the axioms 5 and D and adds the T (for truth) axiom which states that if an agent i beliefs φ, then φ is true ($B(i, \varphi) \rightarrow \varphi$).

[29] If you don't have a particular goal, then you typically also do not have the goal of not making this your goal.

[30] Cohen & Levesque use additional operators Future φ, Necessary φ and Before $\varphi\psi$, all with precise definitions which guarantee that φ will be true at some future point of time, that φ is necessary and that φ is true before ψ is true respectively.

$$[\text{Before } ((B(i,\varphi) \vee B(i, (\text{Necessary } \neg\varphi)))\neg G(i, (\text{Future } \varphi)))] \qquad (3)$$

Thus, an agent has a *persistant* goal φ if (1) the agent has a goal to the effect that φ will be true in the future, (2) the agent believes that φ is not true now, and (3) the agent will not discard the goal φ before this goal has been obtained or is necessarily unobtainable. Intentions can now be defined as persistant goals.[31]

The resulting formalized notions of belief, goal and intention serve as input for what might be called "the rational unit" of a voice interface (i.e., a dialogue manager based on rationality). The rational unit has a number of communicative actions to its disposal, each associated with preconditions (or "feasibility preconditions" in Sadek's terminology) and effects (dubbed "rational effects" by Sadek). For example, one communicative act an agent i may perform is inform an agent j of φ. The precondition is that i believes that φ is true and i does not believe that j already believes φ (formally: $B(i,\varphi) \wedge \neg B(i, B(j,\varphi))$). If the preconditions are satisfied and the inform action is carried out, the effect will be that j believes φ.

The rational unit is centered around two rationality principles. The first one states that if an agent has the intention to achieve a certain goal, then the agent will select an act whose effect corresponds to the goal. Put differently, the agent should select actions in accordance with its goals. The second principle states that once an agent has the intention of performing some action, it also adopts the intention of making the preconditions of this action true. Notice that these two principles, taken together, define a planning algorithm that deduces plans of actions by inferring chains of intentions. For this, as for the circuit fix-it shop, an automatic theorem prover is used, albeit in a rather different way. The system does not ask the user for missing axioms, but only asks the user if the preconditions of an asking action are met and the effects of this asking action (i.e., the answer) contributes to achieving the current goal of the system.

Above, it was stated that the effect of the inform action is that the hearer comes to believe the proposition about which he or she was informed. This is not the case in every form of communication (we typically do not believe everything we are told), but only of *cooperative* communication. Roughly, cooperativity can be obtained if an agent (say the system) has the intention of helping another agent (say the user) obtain his or her intentions.

The rational dialogue manager outlined here has been implemented and forms the core of ARTIMIS[32], which itself is the 'rational core' of AGS, a directory of voice servers hosted by France Télécom (applied to the areas of jobs and weather). The rational dialogue agent manages the dialogue by drawing inferences from the representations of the user's input and the axioms for rational behavior outlined above. For this purpose, a special theorem prover was developed for the first-order model logic described above (see Bretier & Sadek 1996). To get a feeling of how this works consider the following example. Assume that after processing the most recent user input, the system understands that the user wants to know whether p (say, for the sake of concreteness, whether there will be an English detective on channel 1 tonight). On the basis of this, the system derives that it is the intention of the user to know if p, and consequently, that the user does not already believe that p or that

[31] Cohen & Levesque argue that intentions are (even more) complex than persistant goals in that intentions should not be achieved accidentally or unknowingly. Therefore, Cohen & Levesque define that an agent i has the intention of performing an action α if and only if i first has the persistant goal of believing he is about to perform α and then doing it.

[32] Short for *Agent Rationnel à base d'une Théorie formelle de l'Interaction mise en œvre par un Moteur d'Infrence Syntaxique*

$\neg p$. The cooperation principles imply that the system takes over this intention. Based on the first rationality principles, the system adopt the intention of informing the user that p or of informing him/her that $\neg p$. Based on the second rationality principle, the system determines which of these two intentions is currently feasible (i.e., of which the preconditions are met). If the system believes that p, this means that the preconditions of the first action are met (the system, but not the user, believes p). The system then selects this action, sends it to the natural language generation module ("Yes, there is an English detective on channel 1 tonight.") and updates its knowledge base. One effect of carrying out this action is that the user now believes that p.

Advantages and disadvantages

Since the method outlined here is explicitly aimed at a logical reconstruction of the beliefs, desires and intentions model, many of the advantages and disadvantages of that model carry over to the rational approach. On the positive side, of all the computational models of dialogue discussed here, the rational agency model comes closest to modelling human communicative behavior, and as such it holds the promise of being able to communicate in an efficient and cooperative way. Moreover, the theory is, in principle, completely domain and language independent. For building a new application, all that needs to be done is axiomatize the knowledge relevant for the new domain (this is not always a trivial matter, see below). Naturally, the rationality and cooperativity principles remain the same.

A general criticism that can be levelled against the approach of Cohen & Levesque and of Sadek's modifications of it, is that the logic they use is very complex and powerful; it combines classical first-order logic, with epistemic logic (for reasoning about beliefs), temporal logic (for reasoning about time) and dynamic logic (for reasoning about actions). This makes it difficult to determine which parts of the logic serve which purposes. Also the process of updating a realistic set of beliefs is computationally extremely difficult (see e.g., Pulman 1996 for a good discussion of this issue). This can easily be illustrated with an example. Suppose that the system's current belief set $\Delta = \{a, a \rightarrow b\}$. The system now observed that in the world $\neg b$ is the case. Obviously, this belief cannot simply be added to the belief set Δ as it would cause an inconsistency. This means that one of the elements of Δ should be removed (this process is known as *belief revision*). Which one? To solve this problem it is generally assumed that there is some priority ordering on proposition in the belief set. Notice that this requires an additional mechanism. Notice also that a theory of belief revision requires a method for consistency checking (without such a procedure inconsistent beliefs cannot be detected). This is typically done using the notion of entailment or logical consequence. We can say that a belief set Δ is inconsistent if the contradiction (i.e., a formula which is always false, usually denoted as $\perp$) is a logical consequence of Δ. The notion of logical consequence has two problems: (*i*) if the logic is expressive enough, the notion of logical consequence may become undecidable. This is the case already for standard first-order predicate logic. And (*ii*), it may lead to the problem of *logical omniscience*. This is the problem that an agent believes everything that is true (which is typically not the case in real life; there are many true propositions about which we do not have any beliefs). A final problem is that this method requires a formalization of all the relevant knowledge in the domain, which is generally a very hard task. **[end digression]**

7 Error spotting and handling

One of the main problems of current spoken dialogue systems is that speech recognition is not perfect: errors will occur. The computational models of dialogue described so far do not explicitly address this issue. The reason is that these models implicitly assume that dealing with communication problems is just one part of dialogue and as such requires no special attention. Nevertheless, error spotting and handling is one of the main stumbling blocks for current voice interfaces, and consequently, in practice "(...) the lion's share of dialogue management intelligence in the present generation of [voice interface] systems is mainly needed to cope with recognition errors" (den Os, et al., 1999:1528).

Unfortunately, there are no reliable methods (yet) to decide whether the speech recognition engine made an error or not. For instance, one attempted solution for the detection of errors is the use of acoustic confidence scores. Recognition systems use these to decide whether they should 'believe' the recognized string. However, there is not a simple one-to-one relation between low confidence scores and (recognition) errors, nor between high confidence scores and correct recognitions (see, e.g., Bouwman et al., 1999). Other research has shown that prosodic cues and lexical information from the recognized strings — in addition to acoustic confidence measures — may also help to distinguish erroneous utterances from correct ones (e.g., Hirschberg et al., 1999), but, again, these do not completely eliminate misunderstandings.[33] It should be noted that both these strategies are only concerned with recognition errors, whereas communication problems may also be due to other factors, e.g., because the system may make wrong default assumptions. For instance, the train timetable information systems discussed below always assume that the user wants to travel *today*, which is not necessarily the case.[34] In conclusion, a voice interface cannot fully rely on confidence scores and other prosodic or lexical features to decide whether or not it can believe a recognized utterance. This means that current voice interfaces are forced to verify the utterances they recognized.

Consider the following (translated) dialogue ([ppn1-Reuver]) from the corpus of Weegels (2000). Here U is a human user who interacts with a Dutch spoken information system S that provides train timetable information.

S_1: Good morning, this is the talking computer. From which station to which station do you want to travel?
U_1: From Eindhoven to Reuver.
S_2: On what day do you want to travel from Eindhoven to Winsum?
U_2: I want to travel at 8 ′o clock from Eindhoven to Reuver.
S_3: On what day do you want to travel from Eindhoven to Utrecht CS at 8 ′o clock in the morning?
U_3: No, in the evening at 8 ′o clock I want to travel from Eindhoven to Reuver.
S_4: From which station do you want to travel on Saturday July 18 at 8 ′o clock in the morning to Utrecht central station?
U_4: I want to travel tonight at 8 ′o clock from Eindhoven to Reuver.

[33] Conversely, in some cases, the recognizer might tolerate misrecognitions without causing communication errors, i.e., when the recognized string is conceptually identical to the intended meaning of the user's utterance (e.g., "yeah" vs. "yes") (Taylor et al., 1998).

[34] For a more exhaustive analysis of potential sources of errors, see Dybkjær et al. (1998), who report on a Wizard of Oz experiment.

S_5: So you want to travel from Eindhoven to Zutphen today at 20.00?
U_6: No, I want to travel from Eindhoven to Reuver today at 20.00.
S_6: From which station to which station do you want to travel?
etc.

This dialogue is certainly suboptimal by all conceivable standards, even though the user ultimately receives the desired information later on in the actual dialogue. It is difficult to generalize over different voice interfaces, but the example illustrates some of the key aspects of current practice in voice interfaces.

The dialogue system under consideration employs both explicit and implicit verification questions to find out whether it has understood the user correctly.[35] Examples of implicit verification are S_2, S_3 and S_4. An implicit verification question serves *two* purposes: it attempts to verify whether the preceding user utterance was correctly understood, *and* it proceeds with the conversation by immediately asking a follow-up question. The main advantage of implicit verification is that its combination of purposes is efficient; verification does not have to slow down the dialogue. The downside of this strategy is that when the system makes an error, users become rather confused (see, e.g., Weegels, 2000). Correcting an implicit verification amounts to denying a presupposition, which is known to be difficult for speakers. This is most clear for question S_4. To answer it the user both has to supply the requested information and correct the system's assumption. An alternative for implicit verification that is often employed is explicit verification, of which S_5 is a typical example. This question is solely aimed at verifying that the system's current assumptions are correct. Of course, this requires extra turns, which users may find annoying. However, the advantage over implicit verification is that it is generally easier for the system to deduce whether the verified information is indeed correct. Unfortunately, this is not always as simple as one might think, for even though explicit verification questions typically are of the form of a yes/no question, it turns out that users do not always answer with a simple "yes" or "no" to confirm or disconfirm the system's assumptions. In other words, for the detection of problems following explicit verifications, the system cannot rely on the mere presence of a "yes" or a "no" (see, e.g., Hockey, Rossen-Knill, Spejewski, Stone and Isard, 1997). For instance, sometimes users confirm verified information by simply repeating it, or disconfirm it by immediately correcting it. In sum, neither explicit nor implicit verification is by itself a satisfactory solution for dealing with the uncertainties in spoken human-machine interaction. It would be good practice to use a strategy that combines the advantages of both verification strategies, while simultaneously minimizing the disadvantages. Thus, it would be a better strategy to start with implicit verification and immediately change to explicit verification when communication problems arise. This only works if the system has some reliable and automatic strategy to determine whether the communication is going well or not based on the user's input.

In recent years this has been an active field of research, and it appears that user's signal problems in a number of different ways. For example, Krahmer et al. (1999, 2001a) have shown that user's reactions to problems are linguistically marked in various ways: post-error utterances contain more words, more syntactically marked constructions, more repetitions, less new information etc. In addition, user's reactions to problems are prosodically

[35] Even though explicit and implicit verification are the most common verification strategies, this is not to say that they are the only ones. For instance, some systems do not verify immediately, but only when they think they have collected all the relevant pieces of information (compare S_5 above).

different from non-problem signalling utterances in that they are longer, loader, slower, and higher (Swerts et al. 2000, Krahmer et al. 2001b).

The question is whether such features can be detected automatically. Recent work of van den Bosch et al. (2001) suggest that this is indeed the case (see also Walker et al. 2000). Van den Bosch et al. describe a number of machine learning experiments, performed with instance-based learning (Aha et al. 1991) as well as with RIPPER (Cohen, 1996), on a variety of features available in the vast majority of voice interfaces. The best results were obtained with the types of the six most recent system questions and the lexical information from the two most recent word graphs, corresponding to the two most recent user answers. The underlying assumption is that various of the linguistic features found in the aforementioned descriptive studies have correlates in the word graph. On the basis of these features, RIPPER was able to detect communication problems with a 91% accuracy. In the end, we believe that the best results for on-line error detection will be obtained by a combination of factors: the history of system questions, lexical information derived from the word graph, but also acoustic confidence scores and prosodic information.

Here are two examples of how an on-line, quantitive error detection method along the lines advocated here, may be used by the dialogue manager to adapt its strategy to the current state of the dialogue. First, in the introduction, it was noted that neither implicit nor explicit verification is by itself a satisfactory solution for dealing with the uncertainties in human-machine dialogue. An attractive compromise would be to use implicit verification when the user sends 'go on' signals, switch to explicit verification when errors are detected (thus, for instance, it would have been better to pose S_3 of the example dialogue in section 2 in the form of an explicit verification) and back again when the dialogue is on the right track. In this way, the dialogue manager is capable of adapting to the current state of affairs (cf. also Veldhuijzen van Zanten, 1999, Litman and Pan, 1999). A second example situation in which it might pay off to look at positive and negative cues is the following. Levow (1998) found that the probability of experiencing a recognition error after a correct recognition is 16%, but immediately after an incorrect recognition it is 44%. This increase is probably caused by the fact that speakers use hyperarticulate speech when they notice that the system had a problem recognizing their previous utterance. One can imagine a system using two recognizers, one trained on normal speech and one on hyperarticulate speech. If post-processing would reveal that the current user utterance is a likely indicator of problems, then the system could decide to focus on the recognition results delivered by the engine trained on hyperarticulate speech. Whether such a strategy is feasible given the current state of technology (and whether it is at all possible to develop an efficient recognizer tuned for hyperarticulate speech) is still an open question. In any case, such applications trade on the assumption that errors can be spotted automatically and accurately.

How does all this relate to current practice in voice interface design? Some current dialogue systems use a combination of explicit and implicit verification, where the choice of verification strategy is determined by acoustic confidence scores (e.g., Sturm et al., 1999). Given that currently used acoustic confidence scores are not fully reliable, it seems worthwhile to employ user feedback to verification utterances as an additional source of information. It is also relatively common practice to *backtrack* if the user disconfirms verified information. An example of this is S_6 in the example dialogue discussed in the introduction. While this is a reasonable strategy in general (due to the hyperarticulation effects it is often better to just start anew rather than repeatedly try to solve errors), it is also a pity that the system in this example had finally managed to collect all but one of the relevant pieces of information and then was forced to throw away the results. However, using the combined findings just described, an alternative suggests itself: the speaker uses various cues to signal a communication problem, and, moreover, the word "Reuver" is typically associated with a narrow focussed pitch accent. This would provide the dialogue manager with more fine grained information and makes it possible to decide upon a more suitable follow-up question (one specifically focussing on the arrival station). Another common strategy that current voice interfaces often employ is repeating the question when the user failed to provide an answer. The following excerpt from one of the dialogues we studied is an example:

S_1: When do you want to travel?
U_1: On the first day of Christmas.
S_2: What time do you want to travel on July 12?
U_2: (silent)
S_3: Sorry, I did not understand you. What time do you want to travel
on July 12?
etc.

Here, as above, it seems that a better choice would have been to switch from implicit to explicit verification after U_2.

In sum, it seems to be beneficial to pay attention to the cues users actually employ when they are confronted with communication problems. Paying attention to these cues paves the way for principled decisions about follow-up actions in the dialogue. In particular, paying attention to combinations of cues will enable a substantial improvement of the somewhat crude techniques which form current practice (such as backtracking after a disconfirmation, simply repeating the question when the user fails to provide an answer or sticking to implicit verification questions when the user clearly has difficulty answering these).

Part II: The art of voice interfaces

In the prologue it was argued that developing voice interfaces is both a science and an art. So far, we have concentrated on the science side; architectures for voice interfaces have been discussed, a number of computational models of dialogue have been reviewed, and we paid special attention to dealing with communication problems. All these issues are essential for developing a voice interface, but making principled decisions regarding architecture, dialogue model etc. is not sufficient for developing good voice interfaces. It is also important to design the voice interface in a principled way, and this brings us to the *art* of voice interfaces. In this part of the report we will pay attention to the design and development of voice interfaces. In general, there has been, and still is, a lot of interest in designing user interfaces and as we shall see many of the principles of general user interface design are applicable to voice interfaces as well. In this part we focus on the voice interface design and development process (section 8) and discuss some of the more important do's and dont's in the form of a collection of guidelines in section 9.

8 Designing and Developing Voice Interfaces

8.1 What is a usable voice interface?

Probably the main characteristic of a 'good' voice interface is that is *usable*.[36] Usability is a widely discussed concept in the field of interfaces, and various operationalizations have been proposed. One is from Nielsen (1993:26): Nielsen states that usability is a multidimensional concept comprising *learnability*, *efficiency*, *memorability*, *errors* and *satisfaction*, and describes ways in which each of these can be measured. For a clear and informative discussion of these concepts we refer to Nielsen (1993), here we want to discuss what they mean in the context of a voice interface.

First, a voice interface should be *easy to learn*. In a way, this property is easier to obtain for voice interfaces than for other interfaces, since a voice interface builds on an interaction style which come natural to most users: speaking. However, given the limitations of current speech recognizers, the number of possible inputs is limited. This implies that users must learn which inputs are possible in a certain context and which are not. For instance, in the case of voice control, the user must learn which keywords will be recognized and which will not. The easiest way to determine learnability is by measuring the average time which naive users (i.e., users who have no prior experience with this voice interface or even with voice interfaces in general) need to acquire a certain level of proficiency in using it.

A voice interface should also be *efficient to use*. This property is related to previous one in the following way: typically, new users have an initially steep learning curve which gradually flattens as they grow more experienced. Efficiency is now defined as the level of performance in this latter part of the learning curve. Of course, it is very difficult to decide for a given user where he or she is currently located in the learning curve. Therefore, it is usually assumed that users are experienced when they have interacted with the system for a specified amount of time. One way to measure efficiency is to formulate some typical tasks and measure the time it takes users with the required expertise level to finish these tasks.

Memorability is primarily relevant for casual users, i.e., users who interact with a system occasionally. This is not generally the case for voice control interfaces, but it is certainly characteristic of potential users of information services. Memorability is typically measured with a memory test; after having interacted with the system, users are asked to explain the effect of certain inputs and/or to explain which vocal actions are required to perform certain tasks. Another method is to measure the time users need to carry out certain tasks, when they have not been using a particular voice interface for some period of time.

The fourth dimension is concerned with *errors*. In general, users should make few errors and no catastrophic ones. It is worth stressing that in the usual definition of usability this really refers to *user*-errors, defined as any action from the user which does not directly contribute to accomplishing the desired goal (for instance: the user may ask the wrong kind of question or issue a command which is not applicable in the current context). Naturally, in the case of voice interfaces it is highly likely that the *system* will be responsible for the majority of errors. From a usability perspective this is immaterial: it seems a reasonable hypothesis that the number of (speech recognition and/or interpretation) errors is inversely proportional to the perceived usability of the voice interface. Naturally, the number and frequency of errors (both from the user and from the system) are easily determined, one can simply count the errors which occurred when the earlier usability attributes are measured.

[36] In general, usability is only one aspect of a variety of factors which determine the acceptability of a system. Here we will have little to say about other relevant factors such as cost, reliability, compatibility, utility etc.

1. Know the user
   (including task analysis, functional analysis)
2. Competitive analysis
3. Setting usability goals
4. Parallel design
5. Participatory design
6. Coordinated design of the total interface
7. Apply guidelines and heuristic evaluation
8. Prototyping
9. Empirical testing
10. Iterative design
11. Collect feedback from field use

Figure 8: The 11 stages of Nielsen's (1993) Usability Engineering Lifecycle

The fifth and final metric is *subjective satisfaction*: do users find the voice interface pleasant to use. There are various ways to measure subjective satisfaction, the most common is by letting users fill in a questionairre after completing the interaction. Typically, these questionairres are very short. Often, a list of statements is used (e.g. "I found this system very pleasant to use"), and users are asked to rate their degree of agreement using a so-called *Likert scale*. Such a scale may have a range from 5 to 10. In the 5 scale case, 1 = strongly disagree, 2 = partly disagree, 3 = neither agree nor disagree, 4 = partly agree and 5 = strongly agree. An alternative method is to use psychophysical measures such as heart rate and blood pressure. Such measures are more objective and reliable, but they are impopular for usability measures since they generally require 'intimidating experimental conditions' and lead to more tension in subjects. Swerts & Krahmer (2000) have suggested that also the users speech signal may be used to estimate satisfaction; if the interaction is not running smoothly subjects tend to modify their speech by making it overall higher, loader, slower etc.

8.2 The usability engineering lifecycle applied to voice interfaces

Defining when a voice interface is usable is one thing, developing one is quite another. It is by now received wisdom that usability design is an iterative process which should be integrated in the general development process. Several usability engineering lifecycles have been described (see e.g., Nielsen 1993:71ff and Shneiderman 1998:104ff). All these methods should be used throughout the more general software lifecycle, about which we will have little to say here. In this section, we briefly describe the method described by Nielsen in the context of voice interfaces.

Nielsen's usability engineering lifecycle comprises 11 stages (see figure 8). The first step is to *know the user*. Which people will use the voice interface and in which context? Obviously this has consequences for the kind of speech recognizer that should be used, but also on the kind of interaction. As part of this phase, a task analysis and a functional analysis should be carried out. In a task analysis the users' overall goals should be studied as well as the way they currently carry out the task. The result will include a list of actions users want to accomplish with the system and the information they need to be able to carry out these actions. One should not only investigate how users currently perform their task, but also *why* they do it the way they do. This is done in a functional analysis. For example, when looking how users operate a TV it will soon become clear that they typically do this by pressing buttons on the remote control. A naive voice interface might require ludicrous commands like "press the volume button on the remote control". A functional analysis would reveal that what the user really wants is to modify the volume and the use of the remote control is just a means to achieve this.

The second stage which Nielsen describes is an opportunistic one: if you want to make a voice interface it is expedient to look at other voice interfaces that have been developed. The reason is straightforward: these other voice interfaces have been fully developed and one can learn a lot from what they did right and what they got wrong. One could even observe users interacting with these systems and analyse the problems that were encountered.

As we have seen in section 8.1, there are various, measurable attributes which make a voice interface usable. It is important to set usability goals in an early stage of development (stage three), so that it can later be measured to what extent the goals have been met. This also makes it possible to put more emphasis on certain attributes. For example, trying to minimize errors is crucial for voice interfaces, but difficult to achieve given the current state of the art. This puts extra weight on other usability features.

Then, with stage four (parallel design) the actual design starts. Nielsen (1993:85) states that it is often a good idea to start the design with a parallel design process: several people work out preliminary designs in a parallel fashion. In this way, one can explore various strategies before settling on one single approach. Usually, this can a brief phase (from a couple of hours to one or two days). Using this process, one can generate various rough drafts of the basic design idea and select the most promising one(s) for further development.

Parallel design is particularly important for new systems, where little knowledge is available concerning which interface works best. For voice interfaces this may not be the most obvious start. What is important for voice interfaces in the early stages of design and development are a number of basic decisions. Will we use a speaker independent or a speaker dependent recognizer? What dialogue model is the most suitable for the current task?

Stage five of Nielsen's usability engineering lifecycle is concerned with participatory design. It is important to know what future users want of the voice interface that is being developed. The best way to find out is by asking them as early as possible in the design and development. These users need not be able to come up with design solutions for the current system (since as Norman 1988 put it: *users are not designers*), but they generally are very good at responding to first ideas. They typically can ask questions about practical usage which designers would never consider (Norman 1988: *designers are not users*).

The sixth stage is concerned with coordinating the total interface in order to achieve consistency. Consistency is one of the central properties of usable interfaces (see below). An inconsistent user interface is difficult to learn, difficult to memorize, prone to error and probably not very satisfactory. The best way to achieve consistency is by having some central person or group of persons who can coordinate the various aspects of the interface.

The seventh stage is concerned with applying guidelines and heuristic evaluation. Essentially, guidelines summarize basic user interface design principles which should be followed. Guidelines come in two flavors: on the one hand there are small sets of *general guidelines* which are applicable to all user interfaces, on the other hand there typically are larger collections of *application specific guidelines* (for instance guidelines for the development of voice interfaces). In many cases, these specific guidelines fill in the details left unspecified by the general guidelines. Guidelines are an important checklist for the developer (did I apply all the guidelines correctly?), but they are also typically used as a background for a heuristic evaluation. We come back to both kinds of guidelines and to heuristic evaluation in section 9.

It is generally not a good idea to start the actual, full scale implementation on the basis of initial user interface designs. Usually, it is better to develop prototypes on the basis of the early designs, and test these prototype with users. Prototypes allow designers to test their ideas in an early stage and modify these ideas based on the test results. Making prototypes is generally cheaper and faster than developing a complete system. It can thus be done various times, until a good and stable design of the interface has been achieved. For voice interfaces there are various tools which may be used for prototyping purposes, such as SpeechMania and the CSLU toolkit (see section 5). An alternative which is used very often for the development of voice interfaces is the *Wizard of Oz* paradigm (see e.g., Gibbon et al. 1997) idea. This is basically a simulation of a voice interface, where users are led to believe they are interacting with a voice interface, while in fact the voice interface is simulated by a person (the wizard) in a different room. Given an input from the user, the wizard responds in an appropriate way. In this way, voice interaction styles and dialogue strategies can be tested without the need of developing a full interface. Usually it is a good idea to develop a Wizard of Oz system which integrates part of the voice interface (e.g., speech recognition and synthesis) while other parts are performed by a wizard. Such systems are typically referred to as *bionic wizards*. These have a number of advantages over complete wizards: (1) it makes the task of the wizard generally easier, (2) they are more realistic and thus more 'believable' for subjects (they are faster), and (3) speech recognition errors need not be simulated: they are simply there. Wizard of Oz experiments are also used to find out *how* subject will communicate with a given system. For instance, it will give crucial information concerning which words/commands/phrases users may use when they are interacting with the voice interface. This data is used to train the speech recognizer, but it also has important consequences for how the system should express itself. In addition, it provides information on how users interpret utterances from the system: do they understand what the system is saying and do they respond in the way the designer expected?

The ninth stage concerns *evaluation* of the interface. This is a logical follow up on the eight stage: the purpose of developing prototypes is that they allow one to evaluate the design ideas. Evaluation is something that typically needs to be repeated a number of times, also when the interface is nearing its final version. We return to the issue of evaluation at the end of section 9, when we briefly discuss heuristic evaluation, and in section 10.

The tenth stage is arguably the crucial one: *iterative design*. On the basis of the usability problems encountered during the evaluation of version n of the interface, one can produce version n+1. According to Nielsen in each consecutive iteration all of the preceding nine stages should be reconsidered. However, for some of the stages this is more important than for others. It seems likely for instance that the user profile will not change as a consequence of evaluating the last version of the system. Iterative design is of great importance for voice interfaces since experience shows that is generally hard to predict how individual subjects will react to prompts and questions from the system.

The eleventh and final stage of the usability engineering lifecycle starts once the system has been released. It is important to continue gathering usability data, since these may have a strong impact for designing and developing new versions or new, future systems.

Discussion
Nielsen's Usability Engineering Lifecycle is a general work method which aims at producing more usable systems. It is important to stress that it is not always necessary to follow each step; some steps are more essential than others. Usability experts have rated the various stages (Nielsen 1992) and the three highest rates were assigned to, in chronological order:

- task analysis of the user's current task
- empirical tests with real users
- iterative design

As said above, there are various other methods for usability engineering, and these stages are certainly important parts of these methods as well. In general, it is probably not so important for the final result which usability engineering method was used. The important point is that one should not start the actual development too early, which in general can save a lot of time and money. In this way, the chances are minimized that unnecessary functionality is developed or that the whole system needs to be modified due to usability problems.

9 Guidelines for voice interfaces

There are two variants of guidelines: small collections of general guidelines which apply to any kind of interface and larger collections of specific guidelines tailored to a certain kind of interface (e.g., a voice interface). Typically, many of these second kind of guidelines are specific instances of guidelines of the first kind. Various usability researchers have proposed partially overlapping sets of general guidelines (e.g., Nielsen's 1993 eleven usability heuristics for usable interfaces, Norman 1988's four principles of good design and Shneiderman's eight golden rules of interface design). Similarly, the literature on voice interface contains various partially overlapping collections of guidelines specific to this kind of application (Cosky et al. 1995, Fraser 1994, Lea 1994, Leiser 1993, Hapeshi 1993, Karat et al. 1999, Gardner-Bonneau 1999).

Many heuristic rules or guidelines can be understood as ways of helping the user in forming a good *conceptual model* of the system. Consider the general picture (after Norman, 1986) in Figure 9. The user's conceptual model captures how the user thinks a particular system operates; it allows the user to mentally 'simulate' operating the system. A good conceptual model allows a user to predict what the effects of his or her actions are. Users build such a mental model largely through experience. Typically, designers and developers also have a model of how the system operates, and in the ideal world the design model and the user's conceptual model coincide. This ideal is never achieved, since designers and users typically do not interact directly. The user only interacts with the system. Therefore it is important that the system helps the user in building a good conceptual model. This is already difficult for general user interfaces but even more so for voice interfaces, since there usually is no clear system image. Much of the functionality is not visible for the user, thus the "system image" of a voice interface often does not help the user much in building a mental model of how the system works. Multimodal systems with speech form one potential solution to this problem.

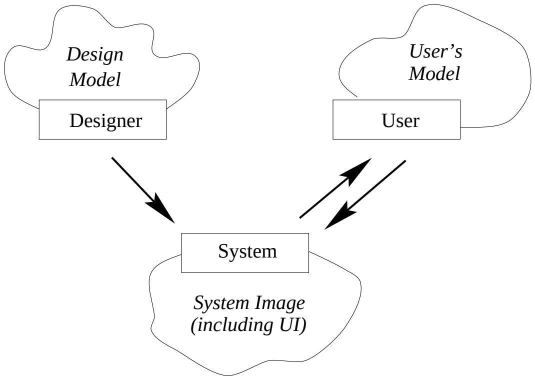

Figure 9: On conceptual models

In this section, we discuss Shneiderman's eight golden rules for general user interfaces and supplement them with specific guidelines for voice interfaces selected from the aforementioned references, where this is relevant.

1. Strive for consistency

An interface should be consistent. This is usually a very difficult rule to obey, for one thing because there are many forms of consistency: consistent sequences of actions should be required in similar situations, identical terminology should be used in prompts, etc. There are various reasons why consistency of an interface is important. Consistency is essential for usability: inconsistencies are a main source of error and a consistent interface is easier to learn. In general, a consistent system makes it easier for the user to form a good conceptual model of the system; the actions of a consistent system are easier to predict. Grudin (1989) notes that sometimes it is better *not* to be consistent. An inconsistent, non-standard reaction will alert the user and attract attention. This can for instance be useful in the case of problems.

The literature on voice interfaces contains the following specific consistency related guidelines:

1. *Balance consistency with adaptivity.* (Leiser 1993:287): be consistent, but also enable the possibility to use more reduced dialogues.

2. *As much as possible, assign the same machine response to the same input in all contexts*. (Lea 1994:26)

2. Enable frequent users to use shortcuts

One of the difficulties of user interface design is that the population of potential users is far from homogeneous. One distinction which has received a lot of attention is that between *novice* and *expert* users. Novice users typically need support from the system in performing the tasks, while expert users want to be able to carry out their tasks in a quick and straightforward manner. One way to accommodate both user-profiles in a single interface is by providing shortcuts for frequent/expert users. Nielsen (1993:41) calls such shortcuts *accelerators*: "UI elements that allow the user to perform frequent tasks quickly, even though the same tasks can also be performed in a more general, and, possibly slower, way." Various ways to introduce shortcuts/accelerators in voice interfaces have been proposed. For command & control applications, it is possible, for instance, to define *power-commands*, i.e., commands which capture various functions in a single keyword. Voice interfaces can accommodate expert users by allowing for mixed initiative dialogue. In this way, expert users can take the initiative and, for instance, provide the system with more information than it requested. This can lead to a substantial speed-up of the dialogue.

3. Offer informative feedback

The third golden rule basically states that for every user action, there should be an appropriate system feedback (compare Norman's principle of feedback 1988:27). In that way, the user can perceive what action was performed and what result was accomplished. This is especially important for voice interfaces, because (1) voice interfaces often do not have a clear system image and (2) given the limitations of speech recognition there is always the possibility that the user is misunderstood.

The golden rule states that feedback should be *informative*. But what is informative? In some cases, it is enough if the system just performs the action, even if the possibility exists that it is the wrong one. For instance, if the user in a command & control interface to a VCR requests to play the tape, the system provides enough feedback by just starting to play the tape. This holds in particular for 'cheap' actions which can easily be undone. In general, it is sufficient to provide precisely enough feedback for the user to determine which action was carried out. There is one exception to this rule of thumb: in the case of errors it is usually expedient to provide *over-informative* feedback. This allows users to determine what went wrong and makes it easier for them to infer what possible remedies there are.

Not only feedback is important, also the questions (prompts) that a voice interface produces need to be designed carefully. The literature on voice interfaces contains the following specific guidelines for prompts.

1. *Keep prompts as brief but explicit as possible without being terse.* (Fraser 1994:137; Lea 1994:15)

2. *Keep prompts as simple as possible.* (Fraser 1994:137)
   Wordy prompt (system: 'I heard you saying ...') lead to confusion. (Lea 1994:15)

3. *Use a consistent linguistic style for prompts.* (Fraser 1994:137)

4. *Wherever technically possible, allow users to interrupt the prompt.* (Fraser 1994:137; Lea 1994:30)

5. *Where prompt interruption is not possible, ensure that either the recognizer starts listening the instant the prompt stops playing, or use some audible signal to indicate when speech may begin.* (Fraser 1994:137)

6. *If prompts are canned, either use a single speaker or, if more than one is used, ensure that each speaker serves an intuitively distinct function.* (Fraser 1994:137)

7. *Different voices can be used to convey the equivalent of different 'active' windows. Use such distinctions carefully.* (Lea 1994:31)

8. *Try to avoid mode switching (such as voice output, then key-board input etc.).* (Lea 1994:16,27) (Compare Reeves & Nass' (1996) rule of matched modality: *If an interface accepts only text input, perhaps it should produce only text output. If the user can respond with voice, then a voice-based interface might work better.*

9. *Do not expect instructions presented to the user at the start of a dialogue to be remembered in subsequent turns.* (Fraser, 137; Lea 16)

10. *Wherever possible, re-prompting after errors of user input should provide extra guidance to help the user behave in the desired fashion.* (Fraser 1994:137)
    Repeat the attempt containing an error one time, so that the user can recognize the error, and do something about it. If the user makes the same error again, rephrase the prompt in such a way that misrecognition is unlikely. (Lea 1994:32)

11. *Machine prompts should guide the naive user, while experienced users should be able to initiate actions (including jumps ahead) without restrictions to prompted choices.* (Lea 1994:30) (see also rule 2)

12. *Ensure that each prompt (except the last) finishes with an explicit question or command, thereby moving to the next stage in the discourse. Proceed with the discourse rather than stepping back to verify the past.* (Fraser 1994:137; Lea 1994:15, 31)

The last statement of the last rule is not uncontroversial and is worth to examine in some more detail. It has been said many times that speech recognition errors may always arise. Hence current voice interfaces are in a constant need of verification. There are essentially two kinds of verification: (1) *explicit* verification, where the system's question is only concerned with verifying whether it understood the user correctly ("Did you say channel one?") and (2) *implicit* verification, where the system's question performs two tasks: verifying and asking a follow up question ("Which program do you want to record on channel one?"). According to the last guideline, implicit verifications should be preferred. However, as we have seen in section 7, users have a lot of difficulty with implicit verification questions when there are problems. It has also been found that even though explicit verification requires extra turns, it does not lengthen the overall duration of the dialogue. In sum, a better rule might be: *use implicit verification if the dialogue runs smoothly, and explicit verification in the case of problems.*

4. Design dialogs to yield closure

The fourth golden rule states that sequences of actions should be organized into groups with a beginning, middle and end. This is important because it structures the interaction, and thus makes it easier to learn and remember. In addition it is beneficial for the user's conceptual model of the interaction. A closely related guideline is Nielsen's (1993:115ff) "simple and natural dialogue"

The literature on voice interfaces contains the following more specific guidelines.

1. *Use progressive disclosure of information; minimize the need to read lengthy manuals.* (Lea 1994:28)

2. *Make frequent things be done with easier, shorter inputs, while infrequent or critical things require more complicated inputs.* (Lea 1994:28)

3. *Help the user to know where he or she is in a discourse, and what to do next.* (Lea 1994:28)

4. *Do not not force interaction.* (Leiser 1993:286): input from the user should not be demanded, he or she should simply be made aware of the features and activities which are available on a take it or leave it basis.

5. *Know the user. Identify novice, intermittent, and expert users, and their features.* (Lea 1994:23)

6. *Use natural speech output (not synthesis) for speech prompts to avoid focus on the quality of the machine voice, and to prevent the common tendency of human users to mimic aspects of prompting voice.* (Lea 1994:25)[37]

7. *Support interruption and recovery.* (Leiser 1993:286): use the normal 'manners' for interrupting the user in his current activities. I.e., only interrupt in 'critical/urgent' situations, and/or justify the interruption. Also reassure the user that the system is robust against sudden interruptions (e.g., by using synthesized speech (less social urgency), and by building up confidence by simply *being* robust).

8. *Mark interaction context.* (Leiser 1993:287): use visual and/or auditory cues to provide context marking, also emphasize switches from one application to another.

9. *Allow inputs which perform many steps, or which go from one point in a discourse to any other point ('jumps').* (Lea 1994:30)

10. *Include best-choice default actions for when user inputs are not provided as allowed.* (Lea 1994:30)

11. *Structure tasks into small pieces, with frequent closure points, so the user doesn't have too much to remember at any point, and so that disruptions are minimized.* (Lea 1994:28)

5. Offer error prevention and simple error handling

The motivation for this rule is straightforward: an absolute minimum of errors is a *sine qua non* for usability. Just like golden rule 3, this rule is of special importance for voice interfaces. The literature on voice interfaces contains the following specific guidelines.

1. *Provide a visible or discernable (audible, interpretable) response for every spoken input (including rejected ones), so the user knows that the machine received the input, and what interpretation is assigned to the input.* (Lea 1994:31)

2. *Provide feedback after each input, to confirm recognition and signal moving ahead in the discourse.* (Lea 1994:31)

3. *Sparingly use unexpected machine outputs or contrasts to the normal condition (e.g, on-screen highlighting, blinking of icon, spoken alert messages). Reserve them for major events.* (Lea 1994:31)

4. *For potentially 'dangerous', expensive actions include a confirmation step, in which the user is asked if he or she really wants to have that action performed. Provide explicit explanations of what will happen. Do* not *use these kind of confirmation steps for 'harmless', inexpensive actions; they slow down the progress and negatively remind the user of the possibility of errors.* (Lea 1994:31) (see also rule 2)

[37] Interestingly, Leiser (1993) argues that the social pressure of the user to converse with the system may be lower if the speech is synthesized.

5. *To prevent errors, make erroneous choices unavailable when the context of the input says the action is inappropriate (compare the 'dimming' of unavailable menu options or visual icons).* (Lea 1994:31)

6. *Use phonetically distinct words or phrases for allowed inputs.* (Lea 1994:31)

7. *When errors occur, have the system 'take the blame' for the error ("I didn't understand that input") or assign no blame. Do not blame the user by phrases such as "Illegal input". Focus on* recovering *the error.* (Lea 1994:32)

8. *Permit user reversal of actions, with 'undo' commands.* (Lea 1994:32) (see also rule 6)

9. *Allow user correction of local errors, without entering the entire command again.* (Lea 1994:32)

6. Permit easy reversal of actions

As much as possible, actions should be reversible (see also discussion on rule 3). This relieves anxiety on the user's behalf, since he or she knows that any errors which may occur can easily be undone. It also encourages exploration of unfamiliar options, which is important for building up a more complete and accurate conceptual model.

7. Support internal locus of control

In general, users do not like to be controlled by a system; users should be the *initiators* of actions, not the *responders*. Gaines (1981) formulated this rule as the *avoid acausality* principle. Again, an exception should be made in the in the case of errors, then the system should take the initiative in providing a solution.

8. Reduce short-term memory load

This final rule is concerned with limitations of human information processing in short-term memory. The seminal work of Miller (1956) indicates that humans can remember "seven-plus or minus two chunks" of information. This implies that sequences of actions, keywords etc. should be minimized. This golden rule poses an interesting paradox for voice interfaces. On the one hand, the use of spoken natural language input frees the user from learning and remembering a specific interaction style. However, due to the limitations of current speech technology, users can generally only use a limited vocabulary. Thus, they still have to learn and remember which are the allowed inputs in a given state. Unfortunately, it is difficult for a voice interface to provide the user with all the relevant options in a given context (there is, for instance, no voice counterpart to a pull-down menu with options).[38]

Discussion

Sets of guidelines such as the 8 golden rules and their refinements for voice interfaces are useful for two reasons. They can be used as a *checklist* when developing a new (voice) interface, and they can serve as the basis of a *heuristic evaluation* (see Nielsen 1993:155ff).[39] A heuristic evaluation is typically performed by a small group of usability experts (± 3-5).

[38] In Cosky *et. al.* (1995) a general so-called *Question + Options* approach is discussed. Each system question is followed by a short pause, which allows the experienced user to immediately give the desired command. After this short pause, a menu is given, from which the novice user can make his or her choice.

[39] See also `http://www.useit.com/papers/heuristic` (checked on 05/02/01).

They evaluate a (paper) prototype with respect to a list of usability principles (the "heuristics"). A main advantage of this method is that it is quick and easy, but still may provide very useful information.

However, collections of guidelines such as the eight golden rules also have their limitations. One general problem is that most guidelines are easier to formulate than to carry out. For example, defining a good 'go back' option is not a straightforward matter. Go back to where? Similarly, dealing with errors is very difficult, certainly in the case of voice interfaces (see section 7).

In general, Shneiderman's golden rules are a very mixed bag: some rules are truly foundational, others (6 and 7, for instance) seem less basic. A related question is: why *eight*? Why not: "provide a good conceptual model" (Norman 1988), "provide a help and documentation" (Nielsen 1993), "make things visible" (Norman 1988). Surely these are also important principles. There are also guidelines which are specific for voice interfaces that do not fit well with the eight golden rules, such as:

1. *When speech is involved, keep the added value of speech in mind.* (Lea 1994:15)

2. *Where possible, use human-human dialogue data to build an understanding of the domain and its component tasks.* (Fraser 1994:128)

3. *In the absence of simulation data, use human-human dialogue data to create vocabularies, language models, and dialogue automata, augmented where necessary by careful use of linguistic intuitions.* (Fraser 1994:128)

4. *Conduct Wizard of Oz simulations to determine the effect of human-computer factors for a specific task of application domain.* (Fraser 1994:138)

In sum, any set of guidelines is somewhat arbitrary if there is no underlying theory.

In this respect it is perhaps interesting to briefly refer to the work of Paul Grice. Grice was a philosopher of language who in 1967 formulated one basic principle and four rules for natural communication. The principle is the cooperativity principle:[40]

> **cooperativity principle**
> Make your contribution such as is required given the current state of the dialogue.

The assumption is that this principle can never be violated by people who engage in a conversation. The four rules, known as the *Gricean maxims*, are the following

maxim of quality
Be consistent

maxim of quantity
Be as informative as necessary

maxim of relevance
Be relevant

[40] Interestingly, Norman and Lewis (1986) formulate a comparable principle when discussing error-handling in UIs.

maxim of manner

Be perspicuous

1. avoid obscurity
2. avoid ambiguity
3. be brief
4. be orderly

There are obvious relations between these maxims and the golden rules. The maxim of quality is golden rule 1. The maxim of quantity is golden rule 3. The maxim of relevance is related to golden rule 4. Finally, the maxim of manner is closely connected with rules 3 and 5. More interestingly, Grice explicitly allows for the possibility of violating the maxims, and this gives rise to so-called "conversational implicatures". For example, the following sentence clearly violates the maxim of manner: *Miss Singer produced a series of sounds corresponding closely to the score of an aria from Rigoletto*. The conversational implicature is obvious: the person who wrote this sentence did not like the singing of Miss Singer. As we have seen, many of the golden rules can also be violated for good reasons; being inconsistent attracts the user's attention, being over-informative provides clues for error-recovery, etc. The connection between UI guidelines and the Gricean maxims has been noted by various researchers (e.g., Baber 1993, Reeves and Nass 1996, and —in the context of voice interfaces— Bernsen et al. 1998). These parallels have led people to speculate on the relations between human-human and human-computer interaction. An interesting reference in this respect is Nickerson (1977), who wrote the following about "conversational interfaces": "Satisfying and effective interactions may resemble conversations in some respects; they may differ from them markedly in others." A somewhat opposite position is taken by Reeves and Nass (1986) who basically claim that there are no differences between the two kinds of interactions. Their central tenet is that humans treat computers as if they are *social actors*. Even though their work is not uncontroversial, it contains many insights which are highly relevant for the developers of voice interfaces.

10 Evaluation

Evaluation is an essential part of usability engineering, and there are many studies on how usability evaluations should be performed (see e.g., Nielsen 1993). However, in the case of voice interfaces, evaluation is one of the main open research issues. Usually, three types of evaluation for voice interfaces are considered relevant (see Hirschmann and Thompson 1996, Gibbon et al. 1997, Bernsen et al. 1998).

1. *performance evaluation*, measurements of the performance of the system ("black box tests") and of its components ("glass box tests") in terms of a set of quantitative parameters,

2. *diagnostic evaluation*, which is essentially Nielsen's heuristic evaluation, and

3. *adequacy evaluation*, study of how well the system and its components fit their purpose and meet the actual user needs.

Here we focus on performance evaluation. Usually, this is done by applying a number of objective and subjective tests to the voice interface. One objective metric which focusses on the complete system is task completion/transaction success, other objective metrics assess the performance of components, e.g., speech recognition rate. A typical subjective measure is user satisfaction.

Using a battery of diverse metrics has a number of disadvantages. For instance, it can happen that different metrics contradict each other. Danieli and Gerbino (1995) (see also Walker et al. 2001) compared two train time table information systems, and found that one had a higher transaction success, while the dialogues with the other were approximately half as long. Which system is better? In general, we want to know how various interacting factors influence the way in which users perceive the performance of a system. Arguably, this also makes it easier to generalize findings across different versions of one system and across different systems.

Walker et al. (1997, 2001) have developed a general framework for evaluating voice interfaces that intends to overcome the aforementioned disadvantages: PARADISE (short for PARAdigm for DIalogue System Evaluation). PARADISE explicitly aims "to develop predictive models of the usability of a system as a function of a range of system properties" and "to make generalizations across systems about which properties of the system impact usability". This is done by deriving a combined performance metric for a dialogue system as a weighted linear combination of a task-based success measure and dialogue costs. The underlying idea is that voice interfaces attempt to maximize user satisfaction, which they can do by maximizing the task success while at the same time minimizing the costs. The framework works as follows: a dialogue corpus is collected under experimental conditions and subjects are asked to subjectively rate their satisfaction on a per-dialogue basis. Then a number of simple metrics that can be directly measured from the system's log-files are used to predict the values for user satisfaction. The measures used by Walker et al. (2001) are the following:

- *Dialogue efficiency metrics*
  - elapsed time, system turns, user turns
- *Dialogue quality metrics*
  - mean recognition scores, timeouts, rejections, helps, cancels, bargeins (raw)
  - timeout%, rejection %, help%, cancel%, bargein% (normalized)
- *Task success metrics*
  - task completion
- *User satisfaction*
  - sum of TTS performance, ASR performance, task ease, interaction pace, user expertise, system response, expected behavior, comparable interfaces, future use

The first three metrics are determined automatically from the corpus, the last metric is based on a user satisfaction survey with questions like "Was the system easy to understand?" (TTS performance), "Did the system work they way you expected him to in this conversation?" (expected behavior). The decision to use both raw and normalized dialogue quality metrics is based on the intention of Walker et al. 2001 to study the generalizabilty across systems. Obviously, the raw data are very much task and domain specific and hence less likely to generalize across systems than the normalized features. ‘Timeout’ refers to the number of prompts to which the user did not respond within the expected time frame, ‘rejects’ are cases in which the system’s confidence score in understanding is so low it repeats the question, ‘helps’ and ‘cancels’ refers to the number of cases in which the system believes the user said ″help″ or ″cancel″ respectively, and ‘bargeins’ records the number of times a user interrupts the system.

Walker et al. (2001) describe how they applied the framework to three different systems, and showed that their framework indeed allows for generalizations across systems and user populations. To do so, they first train their models (one for each system they studied) to predict user satisfaction for the various systems using multivariate linear regression, and then test these models across different systems to see how well they generalize. The results show that, in general, the models can predict about 50% of the variance in user satisfaction for the training data and on average a little lower on the test data. In addition they find that some models generalize better than others, partly due to differences in size of the test set. In addition, they performed a study of which features were the best (significant) predictors of user satisfaction and, somewhat unsurprising, the two main factors are recognition performance and task completion. Put differently, when the recognition performance drops or the task is not completed, users are less satisfied with the interaction. However, other factors also play a role, in particular rejects%, helps% and bargeins%.

In general, it seems fair to say that Walker and co-workers make an interesting contribution to the development of general, predicative models for the usability of voice interfaces. At the same time, the results show that there must be various additional features which co-determine user satisfaction, so more work is needed to develop additional metrics which capture these features. The work by Walker et al. has two interesting additional uses. First, notice that the idea of using a weighed linear combination of features also lies at the heart of the stochastic dialogue manager discussed in section 5.3. In fact, PARADISE can be seen as one way of determining the objective cost function. Second, it would be highly interesting to use PARADISE on-line, i.e., to make on-line predictions of user-satisfaction (see Walker et al. 2000). In this way, the system can adapt its strategy to the user. This would generalize the error-recovery strategy discussed in section 7.

11 FURTHER READING

There is no single reference which captures everything described in this report. Concerning the science of voice interfaces, one has to consult the individual articles mentioned in Part I. Cole et al. (1996) and Jurafsky & Martin (2000) provide fairly accessible overviews of the state of the art in speech and language technology, though both have little to say about computational models of dialogue.

Recently, various books have appeared which are primarily concerned with the art of building voice interfaces. The two handbooks which have been developed within the Eagles project,[41] Gibbon et al. (1997, 2000) are essential reading. They contain very detailed descriptions of the kind of technology required (kind of microphone etc.), Wizard of Oz experiments, data collection, etc. Baber & Noyes (1993) and Gardner-Bonneau (1999) are two collections of usability studies carried out with and for voice interfaces. Both contain a number of interesting studies, and are worth looking into. Bernsen et al. (1998) describes the entire process of designing and developing voice interfaces, based on experiences with the

[41] See `http://coral.lili.uni-bielefeld.de/EAGLES/` (checked on 05/02/01).

Danish Dialogue Project. The book is "firmly applications-oriented" and has very little to say about dialogue models.

A final, useful source for further information is the website of DISC project, focussed on *Spoken Dialogue Systems and Components: Best Practice in Development and Evaluation*. One of the main results of this project is a website, which contains a number of useful links to tools which are relevant for the development of spoken dialogue systems.[42] Also of interest is the list of operational Spoken Dialogue Systems (compiled in summer 2000).[43]

REFERENCES

Aha, D., Kibler, D. & Albert, M. (1991). Instance-based learning techniques. *Machine Learning*, *6*: 37-66

Ahn, R., R.J. Beun, T. Borghuis, H. Bunt, C. van Overveld, 1995. The DenK Architecture: A Fundamental Approach to User-Interfaces. *Artificial Intelligence Review* **8** (3):431-445.

Allen, J., D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, A. Stent 2000, An architecture for a generic dialogue shell, *Natural Language Engineering*, in press.

Allen, J., L. Schubert, G. Ferguson, P. Heeman, C. Hwang, T. Kato, M. Light, N. Martin, B. Miller, M. Poesio and D. Traum, 1995. The TRAINS project: A case study in building a conversational planning agent, *Journal of Experimental and Theoretical AI*, **7**, 7-48.

Allen, J. and C. Perrault, 1980. Analysing intention in utterances, *Artificial Intelligence*, **15**(3):143-178.

Areces, C., *Logic Engineering*, Ph.D. dissertation, ILLC, University of Amsterdam, 2000.

Aust, H., M. Oerder, F. Seide and V. Steinbiss, 1995. The Philips Automatic Train Timetable Information System. *Speech Communication* **17**:249-262.

Austin, J., 1962. *How to do things with words.* Oxford: Clarendon Press.

Baber, C. 1993. Developing interactive speech technology. In: Baber & Noyes (1993).

Baber, C. and J. Noyes (eds.), 1993. *Interactive Speech Technology: Human factors issues in the application of speech input/output to computers*, Taylor & Francis, London.

Bernsen, N., H. Dybkjær and L. Dybkjær, 1998. *Designing Interactive Speech Systems: From First Ideas to User Testing*, Springer Verlag, Berlin.

Black, A. and Taylor, P., 1997. Festival Speech Synthesis System, Human Communication Research Centre, Technical Report, HCRC/TR-83.[44]

van den Bosch, A., Krahmer, E. & Swerts, M. (2001), Detecting problematic turns in human-machine interactions: Rule-induction versus memory-based learning approaches. In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, Toulouse.

[42] See `http://www.disc2.dk/tools/` (checked on 05/02/2001)

[43] See `http://www.disc2.dk/tools/opSDLSs.php` (checked on 05/02/2001).

[44] Downloadable from `http://www.cstr.ed.ac.uk/projects/festival/papers.html`. (checked on 29/11/2000).

Bouwman, A., Sturm, J. & Boves, L. (1999). Incorporating confidence measures in the Dutch train timetable information system developed in the Arise project. *Proceedings International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (pp. 493-496). Phoenix, AZ, Vol. 1.

Bratman, M., D. Israel and M. Pollack, 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence* **4**:349-355.

Bretier, P. and D. Sadek, 1996. A Rational Agent as the Kernel of a Cooperative Spoken Dialogue System: Implementing a Logical Theory of Interaction. In: *Intelligent Agents III*, J. Müller, M. Wooldridge and N. Jennings (eds.), Springer Verlag.

Chu-Caroll, J and M. Brown, 1997. Tracking Initiative in Collaborative Dialogue Interaction, *Proceedings of the 35th Annual Meeting of the Association for Computional Linguistics (ACl-EACL'97)*, pp. 262-270.

Cohen, P. and H. Levesque, 1990. Rational Interaction as the Basis for Communication. In: *Intentions in Communication*, P. Cohen, J. Morgan and M. Pollack (eds.), MIT Press, Cambridge, MA, 221-255.

Cohen, P. and C. Perrault, 1979. Elements of a plan-based theory of speech acts, *Cognitive Science* **3**(3):177-212.

Cohen, W. (1996). Learning trees and rules with set-valued features. *Proceedings 13th National Conference on Artificial Intelligence (AAAI)*.

Colby, K., 1999. Human-Computer Conversation in a Cognitive Therapy Program. In: *Machine Conversations*, Y. Wilks (ed.), Kluwer Academic Publishers.

Cole, R., J. Mariani, H. Uszkoreit, A. Zaenen and V. Zue (eds.), 1996. *Survey of the State of the Art in Human Language Technology*, Cambridge University Press, Cambridge, UK.

Cosky, M., et al., 1995. Talking to Machines Today and Tomorrow: Designing for the User. *AT&T Technical Journal*, pp. 81-90.

Danieli, M. and E. Gerbino, 1995. Metrics for evaluating dialogue strategies in a spoken language system. In: *Proceedings of the 1995 AAAI spring symposium on Empirical Methods in Discourse Interpretation and Generation*, 34-39.

Doddington, G. et al., 1998. Sheep[, goats, lambs and wolves: A statistical analysis of speaker performance in the NIST 1998 speaker recognition evaluation. In: *Proceedings ICSLP-98*, Sydney.

Dybkjær, L., Bernsen, N. & Dybkjær, H. (1998). A methodology for diagnostic evaluation of spoken human-machine dialogue. *International Journal Human-Computer Studies*, *48*:605-625.

Fraser, N., 1994. Interactive dialogue. In: EAGLES Spoken Language Systems (draft), ESPRIT.

Gaines, B. 1981. The technology of interaction: dialogue programming rules, *International Journal of Man-Machine Studies* 14:133-150.

Gardner-Bonneau, D., 1999. Guidelines for Speech-enabled IVR Application Design. In: *Human Factors and Interactive Systems*, D. Gardner-Bonneau (ed.), Kluwer Academic Publishers.

Gibbon, D., R. Moore and R. Winski (eds.), 1997. Handbook of Standards and Resources for Spoken Language Systems. New York, Mouton de Gruyter.

Gibbon, D., I. Mertins and R. Moore (eds.), 2000. Handbook of Multimodal and Spoken Dialogue Systems. Kluwer Academic Publishers.

Goldschen, A and D. Loehr, 1999. The Role of the DARPA Communicator Architecture as a Human-Computer Interface for Distributed Simulations. In: *Simulation Interoperability Standards Organization (SISO) Spring Simulation Interoperability Workshop*, Orlando, Florida, March 14-19[45]

Grudin, J. 1989. The case against user interface consistency. *Communications of the ACM*, 32(10),1164-1173.

Grosz, B. and C. Sidner, 1986. Attentions, intentions and the structure of discourse, *Computational Linguistics* **12**(3):175-204.

Hapeshi, K., 1993. Design guidelines for using speech in interactive multimedia systems. In: *Interactive Speech Technology: Human Factors Issues in the Application of Speech Input/Output to Computers*, C. Baber and J. Noyes (eds.), Taylor & Francis, London.

Hirschman, L. and H.S. Thompson, 1996. Overview of evaluation in speech and natural language processing. In: Cole et al. 1996.

Hirschberg, J., Litman, D. and Swerts, M., 1999. Prosodic cues to recognition errors, in: *Proc. of the 1999 international workshop on Automatic Speech Recognition and Understanding (ASRU)*, Keystone, CO, December 1999.

Hintikka, J., 1962. *Knowledge and Belief* Cornell University Press, Ithaca.

Hockey, B., Rossen-Knill, D., Spejewski, B., Stone, M. & Isard, S. (1997). Can you predict answers to y/n questions? Yes, no and stuff. *Proceedings of Eurospeech'97* (pp. 2267-2270). ESCA, Rhodes, Greece.

van der Hoeven, G., Andernach, J., van der Burgt, S., Kruijff, G.-J., Nijholt, A., Schaake, J., and de Jong, F., 1995. SCHISMA: A natural language accessible theatre information and booking system. In: *First International Workshop on Applications of Natural Language to Data Bases (NLDB'95*, Versailles, France.

Hulstijn, J., 2000. *Dialogue Models for Inquiry and Transaction*, Ph.D. dissertation, University of Twente.

Jurafsky, D. and J. Martin, 2000. Speech and Language Processing: An introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Prentice-Hall.

[45] `http://fofaca.mitre.org/doc.html`; checked on 28/08/2000.

Jelinek, F., 1997. *Statistical Techniques for Speech Recognition*, MIT Press, Cambridge, MA.

Karat, J., J. Lai, C. Danis, and C. Wolf, 1999. Speech User Interface Evaluation. In: *Human Factors and Interactive Systems*, D. Gardner-Bonneau (ed.), Kluwer Academic Publishers.

Kautz, H. 1991. A formal theory of plan recognitions and its implementation. In: Allen, J., Kautz, H., Pelavin, R and Tenenberg, J (eds.), *Reasoning about plans*, Morgan Kaufman, 69-125.

Kearns, M. and Singh, S., 1998. Finite-sample comvergence rates for Q-learning and indirect algorithms. In: *Proc. Neural Information Processing Systems*, Denver, CO.

Klabbers, E., 2000. *Segmental and prosodic improvements to speech generation*, PhD dissertation, Einhoven University of Technology.

Krahmer, E., Swerts, M., Theune, M., Weegels, M., 1999. Error Spotting in Human- Machine Interaction. *Proceedings Eurospeech'99*, September 1999, Budapest

Krahmer, E., Swerts, M., Theune, M., Weegels, M., 2001a. Error Detection in Spoken Human-Machine Interaction. *International Journal of Speech Technology*, to appear.

Krahmer, E., Swerts, M., Theune, M., Weegels, M., 2001b. The Dual of Denial: Two uses of disconfirmations in dialogue and their prosodic correlates. *Speech Communication*, to appear.

Larson, J., 1992. *Interactive software: tools for building interactive user-interfaces*, Prentice Hall, New Jersey, USA.

Langley, P., C. Thompson, R. Elio and A. Haddadi, 1999. An Adaptive Conversational Interface for Destination Advice. In: *Proceedings of the Thrid International Workshop on Cooperative Information Agents*, Uppsala, Sweden. Springer Verlag.

Lea, W., 1994. Developing usable voice interfaces. *Journal of the American Voice Input/Output Society*, vol. 16.

Leiser, R., 1993. Driver-vehicle interface: dialogue design for voice input. In: *Driving future vehicles*, A. Parkes & S. Franzen (eds.), Taylor & Francis, pp. 275-293.

Levin, E., R. Pieraccini and W. Eckert, 2000. A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies. *IEEE Transactions on Speech and Audio Processing*, **8**(1):11-23.

Levow, G.A. (1998), Characterizing and Recognizing Spoken Corrections in Human-Computer Dialogue. *Proceedings of the 36th Annual Meeting of Association for Computational Linguistics and the 17th International Conference on Computational Linguistics (COLING-ACL)* (pp. 736-742). August 10-14, Montreal, Canada.

Litman, D. & Pan, S. (1999). Empirically evaluating an adaptable spoken dialogue system. *Proceedings of the 7th International Conference on User Modelling (UM)*

Litman, D., Hirschberg, J., and Swerts, M. 2000a. Predicting Automatic Speech Recognition Performance Using Prosodic Cues, in: *Proc. of the First North-American Chapter of the Association for Computational Linguistics*, April 29-May 4, Seattle, Washington.

Litman, D., M. Kearns, S. Singh. M. Walker 2000b. Automatic Optimization of Dialogue Management, in: *Proceedings of the 18th International Conference on Computional Linguistics*, Saarbrücken, Germany.

Massaro, D., Cohen, M., Beskow, J., Cole, R., 2000. Developing and evaluating conversational agents. In: *Embodied Conversational Agents*, J. Cassell, J. Sullivan, S. Prevost, E. Churchill (eds.), MIT Press, Cambridge, MA, pp. 287-318.

Miller, G.A., 1956. The magical number seven, plus or minus two: some limits to our capacity for processing information. *The Psychological Review*, 63:81-97.

Montague, R., 1960. Logical necessity, physical necessity, ethics and quantifiers. *Inquiry* **4**:259-269.

de Mori, R., 1998. *Spoken Dialogues with Computers*, Academic Press, London.

Nass, C. & K.M. Lee, 2000, Does computer-generated speech manifest personality? An experimental test of similarity-attraction, *Proceedings CHI 2000*, 329-336.

Nickerson, R. 1977. On conversational interaction with computers. *User-oriented design of interactive graphics system*, Association of Computing Machinery, New York, 101-113. Reprinted in: *Readings in Human-Computer Interaction*, R. Baecker and W. Buxton (eds.), San Mateo: Morgan Kaufmann, 1986.

Nielsen, J. 1992. The Usability Engineering Lifecycle. *IEEE Computer* **25**(3):12-22.

Nielsen, J. 1993. *Usability Engineering*, Morgan Kaufmann, San Diego.

Norman, D. 1986. Cognitive engineering. In: *User centered system design: new perspectives on human-computer interaction*, Hillsdale, NJ, Erlbaum.

Norman, D. 1988. *The psychology of everyday things*, Basic Books, New York.

den Os, E., Boves, L., Lamel, L. & Baggia, P. (1999). Overview of the ARISE project. *Proceedings of Eurospeech'99* (pp. 1527-1530). ESCA, Budapest, Hungary.

Oviatt, S., Bernard, J., and Levow, G.A. (1998). Linguistic adaptations during spoken and multimodal error resolution. *Language and Speech. Special issue on Prosody and Conversation*, **41**, pp. 419–422.

Oviatt, S. and Cohen, P., 2000. Multimodal interfaces that process what comes naturally, *Communications of the ACM*, **43**(3):45-53, 2000.

Pieraccini, R., E. Levin and W. Eckert. 1997. AMICA: The AT&T mixed initiative conversational architecture, *Eurospeech'97*, Rhodes, Greece, 1875-1878.

Reeves, B. and Nass, C., 1996. *The media equation: How people treat computers, television, and new media like real people and places*, CSLI Publications/Cambridge University Press, Stanford, CA.

Roy, N., J. Pineau, S. Thrun, 2000. Spoken Dialogue Managment using Probabilistic Reasoning. In: *Proceedings ACL 2000*, Hong Kong.

Sadek, D., 1994. Towards a Theory of Belief Reconstruction: Application to Communication. *Speech Communication* 15:251-263.

Sadek, D. and R. de Mori, 1998. Dialogue Systems. In: *Spoken Dialogues with Computers*, R. de Mori (ed.), Academic Press, pp. 523-562.

Schank, R and K. Colby, 1973. *Computer Models of Thought and Language* San Francisco, CA, Freeman.

Searle, J., 1969. *Speech Acts*, Cambridge University Press, Cambridge.

Shannon, C., 1948. A mathematical theory of communication, *The Bell System Technical Journal*, **27**: 379-423, 623, 656,

Shieber, S., 1994. Lessons from a Restricted Turing Test. *Communications of the ACM* 37(6):70-78.

Shneiderman, B., (1998), *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 3rd edition, Addison-Wesley, Reading.

Singh, S., M. Kearns, D. Litman, M. Walker. 2000. Empirical Evaluation of a Reinforcement Learning Spoken Dialogue System. *Proceedings of the Seventeeth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX, August 2000, 645-651.

Smith, W and D. Hipp, 1994. *Spoken natural language dialog systems: A practical approach*, Oxford University Press.

Souvigner, B., A. Kellner, B. Rueber, H. Schramm, F. Seide, 2000. The Thoughtful Elephant: Strategies for Spoken Dialog Systems. *IEEE Transactions on Speech and Audio Processing* **8** (1):51-62.

Stork, D. (ed.), 1996. *HAL's legacy: 2001s computer as dream and reality*. MIT Press, Cambridge, MA.[46]

Sturm, J., den Os, E. & Boves, L. (1999). Dialogue management in the Dutch ARISE train timetable information system. *Proceedings of Eurospeech'99* (pp. 1419-1422). ESCA, Budapest, Hungary.

Sutton, R., 1991. Planning by Incremental Dynamic Programming. In: *Proc. Ninth conference on Machine Learning*, 353-357.

Sutton, R., and A. Barto, 1998. *Reinforcement Learning: An Introduction*, Cambridge, MA, MIT Press.

Sutton, S., Cole, R., de Villiers, J., Schalkwyk, J., Vermeulen, P., Macon, M., Yan, Y., Kaiser, E., Rundle, B., Shobaki, K., Hosom, P., Kain, A., Wouters, J., Massario, M., Cohen, P., 1998. Universal Speech Tools: The CSLU Toolkit, *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Sydney, Australia, 3221-3224

[46] On-line: `http://mitpress.mit.edu/e-books/hal/`, checked on 29/08/2000.

Swerts, M., and Krahmer, E., 2000. On the Use of Prosody for On-line Evaluation of Spoken Dialogue Systems, *Proceedings Second International Conference on Language Resources and Evaluation*, Athens, Greece, May 31 - June 2, 2000.

Swerts, M., Litman, D. and Hirschberg, J., 2000. Corrections in spoken dialogue systems, *Proceedings ICSLP 2000*, Peking, China.

Taylor, P., King, S., Isard, S. & Wright, H. (1998). Intonation and dialogue context as constraints for speech recognition. *Language and Speech. Special issue on Prosody and Conversation*, *41* (*3-4*): 493-512.

Theune, M. 2000. *From Data to Speech: Language Generation in Context*, Ph.D. thesis, Technical University Eindhoven.

Turing, A., 1950. Computing machinery and intelligence. *Mind* LIX (236):433-460.

Traum, D., L. Schubert, M. Poesio, N. Martin, M. Light, C. Hee Hwang, P. Heeman, G. Ferguson, J. Allen, 1996. Knowledge Representation in the TRAINS-93 Conversation System. *International Journal of Expert Systems* **9**(1):173-223.

Veldhuijzen-van Zanten, G., 1999. User-modeling in Adaptive Dialogue Management. In: *Proceedings of Eurospeech'99*, Budapest, Hungary, September 14-19, pp. 1183-1186.

Veldhuijzen-van Zanten, G., 1999. Dialogue Management. Manuscript.

Walker, M., Langskilde, I., Wright, J., Gorin, A., Litman, D. (2000), Learning to predict problematic situations in a spoken dialogue system: Experiment with How May I Help You?, *Proceedings of the 1st North-American Chapter of the Association for Computational Linguistic (NAACL)*

Walker, M., D. Litman, C. Kamm and A. Abella, 1997. PARADISE: A framework for evaluation of spoekn dialog agents. In: *Proceedings of the 35th Annual Meeting of the Association of Computational Linguistics*, Madrid, Spain.

Walker, M., Wright, J. Langkilde, I. (2000b), Using natural language processing and discourse features to identify understanding errors in a spoken dialogue system, *Proc. of the Interntational Conference on Machine Learning* (ICML), Stanford, CA.

Walker, M., C. Kamm and D. Litman, 2001. Towards developing general models of usability with PARADISE, *Natural Language Engineering*, in press.

Weegels, M., 1999. Users' (Mis)conceptions of a voice-operated train travel information service, *IPO Annual Progress Report*, Eindhoven, The Netherlands.

Weegels, M. 2000. Users' conceptions of voice-operated information services. *International Journal of Speech Technology* **3**(2):75-82.

Weizenbaum, J. 1966. ELIZA. *Communications of the ACM* 9:36-45.

Weizenbaum, J. 1974. Automating Psychotherapy. *Communications of the ACM* 17(7):425.

Young, S., 2000. Probabilistic Methods in Spoken Dialogue Systems. *Philosophical Transactions of the Royal Society*, **358**: 1389-1402.

Zue, V. Conversational Interfaces: Advances and Challanges. *Proceedings of Eurospeech'97*, 9-18, Rhodes, Greece.

Zue, V., S. Seneff, J. Glass, J. Polifroni, C. Pao, T. Hazen and L. Hetherington, 2000. JUPITER: A Telephone-Based Conversational Interface for Weather Information. *IEEE Transactions on Speech and Audio Processing*, **8**(1):85-96.