Jaakko Hakulinen

Software Tutoring in Speech User Interfaces

ACADEMIC DISSERTATION

To be presented with the permission of the Faculty of Information Sciences of the University of Tampere, for public discussion in Pinni auditorium B1100 on December 8th, 2006, at noon.

Department of Computer Sciences
University of Tampere

Dissertations in Interactive Technology, Number 6
Tampere 2006

ACADEMIC DISSERTATION IN INTERACTIVE TECHNOLOGY

Supervisor:	Professor Kari-Jouko Räihä, Ph.D., Department of Computer Sciences, University of Tampere, Finland
Opponent:	Associate Professor Lars Bo Larsen, Ph.D., Department of Communication Technology, Aalborg University, Denmark
Reviewers:	Professor Arne Jönsson, Ph.D., Department of Computer and Information Science, Linköping University, Sweden Associate Professor Jacques Terken, Ph.D., Department Industrial Design, Technische Universiteit Eindhoven, The Netherlands

Electronic dissertation
Acta Electronica Universitatis Tamperensis 581
ISBN 951-44-6802-3
ISSN 1456-954X
http://acta.uta.fi

Dissertations in Interactive Technology, Number 6

Department of Computer Sciences
FIN-33014 University of Tampere
FINLAND

ISBN 951-44-6746-9
ISSN 1795-9489

Tampereen yliopistopaino Oy
Tampere 2006

Abstract

Speech has been in use as a computer user interface for some time already. For more than two decades, speech-recognition-based services have been publicly available, and the number of available services has risen steadily.

However, speech user interfaces have not reached the wide popularity sometimes hoped for or expected. This is due to the limitations of spoken human−computer interaction. Many of these arise from the current computer technology available and our capabilities of building speech-based user interfaces. However, the very nature of speech imposes its own limits as well, which cannot be overcome.

The biggest challenge with speech user interfaces is that the interaction objects are invisible. Therefore, users must know what objects and operations are available in each system, and what they are called. In human–human interaction, the unlimited use of natural language and the possibility of cooperative meta-communication about the task enable us to work with each other so that, for example, collaborative problem-solving using speech is effective. Computers, on the other hand, can understand only limited subsets of natural language, and their communication skills are considerably weaker than those of people.

For speech interfaces to be usable, existing systems either provide the necessary guidance to users as part of the user interface or require the users to learn about usage from some sort of guidance material. In speech-only interfaces, the guidance can be embedded in speech interaction via explicit prompts, hints, and confirmations. This makes small applications reasonably usable but larger applications and frequent use inefficient. The other option, external guidance material, is not favored – users tend not to read manuals, instead trying to learn by trial and error.

In this study, software tutoring has been applied for speech user interfaces. A software tutor is a software component that teaches the use of a software application. A tutor can monitor users' actions and adapt appropriately. The teaching happens *in situ;* the users learn about the application while using it. This kind of user guidance is particularly popular in computer and video games, whose users cannot be assumed to read manuals, since they want to start playing immediately.

The study presents two kinds of software tutors: a speech-based tutor and multimodal tutors. The speech-based tutor is embedded in a speech interface for e-mail reading and provides guidance for new users. The multimodal tutors use text and graphics to teach the usage of a speech-based timetable system. The tutors are connected to the system so that they can, for example, visualize the spoken interaction in real time.

The nature of the tutors, their technical solutions, the iterative development process, and formal evaluations are reported. The results show that the tutors can support new users better than the previously used static text-based guidance materials can.

Acknowledgements

This work was done in the Speech-based and Pervasive Interaction Group, a part of Tampere Unit for Computer-Human Interaction. The unit is a part of the Department of Computer Sciences, in the Faculty of Information Sciences, in the University of Tampere. Therefore, I would like to thank all the people affiliated to the aforementioned hierarchy whom I have been in contact with.

In particular, I must thank Markku Turunen for coordinating the SPI-group, being my closest colleague over the years, and even for getting me to work in the TAUCHI unit back in 1998. Another very important person is Esa-Pekka Salonen, who had an important role in the early phases of the work when the founding ideas were born. I'm also thankful for all the other SPI-members, past and present.

I owe gratitude to Kari-Jouko Räihä for two reasons. He supervised this work and has created TAUCHI, a very good work environment. Of course, all TAUCHI members contribute as well, each in their unique way.

The department, the faculty, and the university, have provided many things ranging from physical facilities to administration and even free-time activities.

Funding for this work was provided by Tampere Graduate School in Information Science and Engineering. I could not have created such a coherent thesis in this kind of timeframe without this support. My compliments to TISE coordinator, director, and board.

Finally, I'd like to thank the reviewers of the thesis, Arne Jönsson and Jacques Terken, for their constructive comments.

Contents

List of Publications

This thesis consists of a summary and the following original publications, reproduced here by the permission of respective copyright holders.

In the summary part, the publications will be referred to by the corresponding Roman numerals (I – VII). The work presented in the publications was performed in the speech-based and pervasive interaction research group at the University of Tampere. The group has contributed greatly to the work. Most importantly, the applications for which the tutors were built, and the underlying framework for the applications and tutors (Jaspis), have been developed as a group effort. The author has had a major role in the development of Jaspis, working in collaboration with the main author of the framework, Dr. Turunen, throughout its existence. Because of this role and, more importantly, because of the importance of the technical solutions tightly bound to Jaspis in this work, a publication presenting the framework and many of the applications built on it (Publication I) has been included in this compound thesis.

The idea of software tutoring in speech user interfaces was born in the research group. The author was one of the people participating in these discussions and decided to study the idea in detail in the form of this dissertation work.

The author has designed and implemented the tutors, and led the design of the evaluation studies. The studies were also conducted and analyzed by the author.

The included articles have been written cooperatively in the research group. The order listed for the authors in the papers reflects the roles in the writing process. First position indicates the role of main author.

1 Introduction

The development of spoken-language technology has enabled real-world speech-based applications, and, while not ubiquitous, these are becoming more popular. Numerous successful spoken-language dialogue applications have been in public use since 1981 (Cox *et al.*, 2000). In addition to telephone-based information services, voice control functionality in cars is commonplace (Heisterkamp, 2001).

Speech provides a natural way of communication for people. Human–human communication with speech takes place constantly. People typically learn to speak and understand speech very early in life. Because of this, speech has sometimes been considered the ultimate solution for the usability of computers. Works of science fiction provide examples of idealized spoken human–machine interaction. In real life, however, speech is not the ultimate solution to all tasks. For example, presenting spatial information with speech is challenging (Oviatt, 1996b). More importantly, currently technology in this field limits the practical usage of speech. Recognition vocabularies are limited in size and complexity because of the demands of speech recognition and natural language understanding. Currently speech technology is not capable of using syntactic and semantic information to the extent humans do and therefore is not capable of similar speech recognition accuracy (Huang, Acero & Hon, 2001, p. 12).

Those who use speech user interfaces must know what the system in question understands. Because of the technological limitations, they cannot speak as they please (Yankelovich, 1996). Graphical user interfaces differ from speech in the sense that interaction objects are visible to users in the former, and one can visually search for actions (e.g., menu options). Speech-based user interfaces somewhat resemble command-line interfaces in this respect. Therefore, speech is not as natural as one might naïvely imagine. Users need to be guided in using speech user interfaces.

The necessary guidance can be embedded in the speech user interface in different ways. The interface can ask users explicit questions, list the possible options, and carefully prompt the user so that the responses naturally match the system's limits. This help can be adaptive also, so that the amount of guidance is reduced if interaction is successful and more guidance is given when problems occur. This way, the problems caused by the fact that speech has a slow output rate can be reduced. Such solutions have been successful in many real-life spoken-language-dialogue systems. However, they can provide all of the necessary guidance only in rather straightforward applications with clearly defined tasks. The guidance provided also slows the pace of the interaction and reduces the power and naturalness of spoken interaction, therefore having potential to be problematic in applications that are used frequently by a single user.

Guidance can be provided separately from the application. The separate guidance materials can provide descriptions that are more extensive and teach high-level concepts. The guidance material can be a printed manual, interactive guidance material connected to the actual application, or something in between. The biggest problem is that users are not willing to spend time reading manuals; they want to start using the application right away (Carroll & Rosson, 1987). In addition, manuals themselves can possess usability and accessibility problems hindering or even preventing their use.

Software tutoring is a guidance component that is connected with or integrated into a software application. However, it is not part of the interface, since it can be removed, or turned off, when a user does not need it anymore. Software tutoring is an active component. It monitors the user's interaction with the application and adjusts the guidance accordingly. Tutoring can also have an active role in the interaction. It can give explicit instructions to the user on what to do and guide the user through the system.

This thesis examines software tutoring in speech user interfaces, and spoken-language-dialogue systems in particular. The idea of studying tutoring was born in discussions of how to solve the user guidance challenge. The requirement for guidance is a fundamental issue in speech-based user interface design; therefore, there is need for innovative research. Software tutoring has not been studied previously in the context of speech user interfaces. Since tutoring has the potential to be an efficient guidance method, there was reasonable motivation to study it.

The research questions addressed in this work are:

- How can software tutoring be implemented in spoken-language-dialogue systems?
- How should the tutors be designed?
- Can we gain some benefits with tutoring over other guidance methods?

The research methods applied in this work include implementation of working prototype tutors for existing spoken-language-dialogue applications, iterative development of these with user tests, and evaluations of the resulting tutors in more formal settings to gain quantitative data related to the effects of tutoring. The effectiveness of tutoring is most likely highly dependent on the design of the tutors. The interface design solutions employed can make tutoring succeed or fail. It is not possible to address the entire scope of software tutoring design in one piece of research. Because of this, iterative development has been utilized.

I have implemented two different kinds of tutors: speech-based tutors and multimodal tutors. The two approaches are significantly different since the first needs to use the same channel for interaction with a user as the application being taught does, while the latter can utilize graphics and other modalities. The speech-based tutor was developed for an e-mail-reading application and the multimodal tutors for a bus schedule application. While using different applications made it impossible to directly compare the implementations of the two tutoring approaches, it was considered more important to test the tutoring concept with different kinds of applications. The two applications have many differences. In addition to the different domains, the dialogue styles differ; the e-mail system features user initiative dialogue while the interaction with the bus schedule system has mixed initiative.

This thesis consists of seven articles, previously published in conference proceedings and journals, and an introduction to the subject of the study. Chapter 2 provides an overview of relevant aspects of speech user interfaces and spoken interaction. Both speech – its nature in general and speech technology – and its limitations are studied, since their understanding provides necessary background for designing spoken-language-dialogue systems. Chapter 3 summarizes studies of user guidance, from traditional paper manuals to interactive guidance, in order to offer understanding of the challenges and possibilities of various guidance methods. Chapter 4 contains discussion on employment of a user-centered approach in both speech-based user interfaces and user guidance, since users' point of view is the main focus of the study. Chapter 5 combines these subjects in a discussion of software tutoring for a spoken-language-dialogue system. Then, Chapter 6 introduces the work presented in the publications and, finally, Chapter 7 contains discussion and conclusions stemming from the work.

2 Speech User Interfaces

2.1 INTRODUCTION

The nature of speech and spoken interaction, as well as the capabilities of speech technology, must be understood, and that knowledge used to design successful speech user interfaces. When guidance for users of speech user interfaces is designed or studied, it is necessary to understand how such user interfaces work.

There are many factors affecting spoken interaction and speech interfaces and they can be categorized in different ways. For example, Suhm (2003a) groups the limitations of speech interfaces into six categories: speech recognizer, spoken language, environment, human cognition, user, and hardware. In this work, I first discuss the features of speech in general, then spoken interaction and its requirements. These are followed by discussion of speech technology to the extent relevant to speech interface design. Finally, an overview on different kinds of speech interfaces currently available is given.

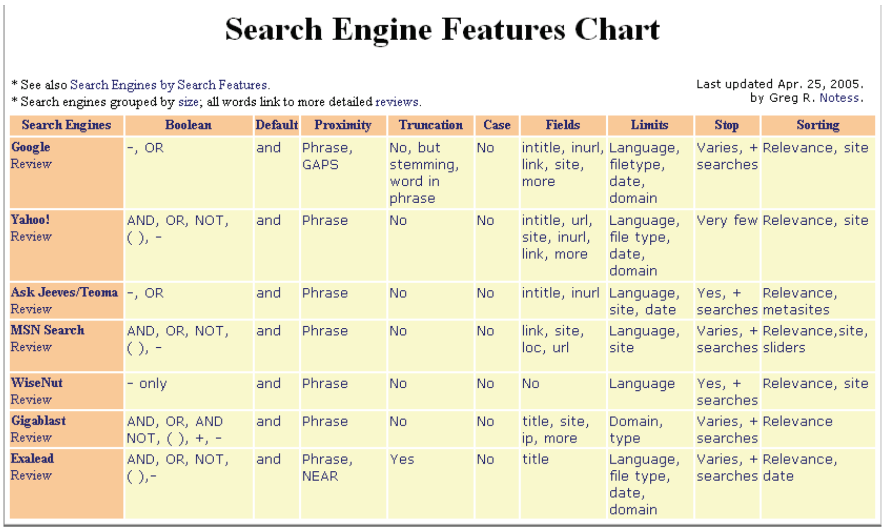
2.2 SPEECH

Speech, be it between humans or a human and a machine, has features that are very different from, e.g., visual presentation of information. Understanding these features forms the foundation of the knowledge required for designing speech-based user interfaces. Speech can be categorized as sequential and temporal method of communication, which is error-prone, ambiguous, and public by its nature. When speech is contrasted to contemporary human-computer interaction, characterized by direct manipulation, a significant difference is that interaction objects are not visible in speech-based interaction.

Sequential nature

Speech is a sequential output modality (Schmandt, 1994, p. 102). The listener receives information in the order in which the speaker decides to present it. Everything that is presented with speech must be first linearized. Tables, which are two-dimensional visual entities, provide an excellent example of this. It is very hard to present tables with speech so that they are comprehensible. Figure 1 contains an example in which an HTML table has been linearized. The linearization repeats column labels for each entry. This removes the need to remember the labels when the linearized form is presented using speech. In addition to repetition of certain elements, prosody and non-speech audio can aid table comprehension (Spiliotopoulos, Xydas & Kouroupetroglou, 2005). The challenge can be tackled with interactive solutions as well (Yesilada *et al.*, 2003) and by intelligently summarizing the content of the table.

The linearity limits not only the order in which the listener receives the information but also the speed and level of detail. While it is possible to skim through a text, the rate of speech can be increased only so far (Schmandt, 1994, pp. 57–58). Interactivity is required if speech is to support anything similar to text skimming (Arons, 1997). While a normal human being in a regular conversation produces speech at a respectable rate of around 10–12 phonemes per second (Levelt, 1999), a fluent reader can read text even more quickly. Those who use speech synthesis regularly learn to comprehend it so well that they can set the pace of the speech to rates beyond human speaking capabilities (Schmandt, 1994, p. 108). However, this requires constant use of a particular speech synthesizer, and the content read must be reasonably predictable.


Search Engine Features Chart

* See also Search Engines by Search Features.
* Search engines grouped by size; all words link to more detailed reviews.

Last updated Apr. 25, 2005.
by Greg R. Notess.

Search Engines	Boolean	Default	Proximity	Truncation	Case	Fields	Limits	Stop	Sorting
Google Review	-, OR	and	Phrase, GAPS	No, but stemming, word in phrase	No	intitle, inurl, link, site, more	Language, filetype, date, domain	Varies, + searches	Relevance, site
Yahoo! Review	AND, OR, NOT, ( ), -	and	Phrase	No	No	intitle, url, site, inurl, link, more	Language, file type, date, domain	Very few	Relevance, site
Ask Jeeves/Teoma Review	-, OR	and	Phrase	No	No	intitle, inurl	Language, site, date	Yes, + searches	Relevance, metasites
MSN Search Review	AND, OR, NOT, ( ), -	and	Phrase	No	No	link, site, loc, url	Language, site	Varies, + searches	Relevance,site, sliders
WiseNut Review	- only	and	Phrase	No	No	No	Language	Yes, + searches	Relevance, site
Gigablast Review	AND, OR, AND NOT, ( ), +, -	and	Phrase	No	No	title, site, ip, more	Domain, type	Varies, + searches	Relevance
Exalead Review	AND, OR, NOT, ( ),-	and	Phrase, NEAR	Yes	No	title	Language, file type, date, domain	Varies, + searches	Relevance, date

Last updated Apr. 25, 2005. by Greg R. Notess.

Search Engines = Google Review , Boolean = -, OR, Default = and, Proximity = Phrase, GAPS, Truncation = No, but stemming, word in phrase, Case = No, Fields = intitle, inurl, link, site, more, Limits = Language, filetype, date, domain , Stop = Varies, + searches, Sorting = Relevance, site

Search Engines = Yahoo! Review, Boolean = AND, OR, NOT, ( ), - , Default = and, Proximity = Phrase, Truncation = No, Case = No, Fields = intitle, url, site, inurl, link, more,…

Figure 1. Table and a portion of its linearization using Tablin (http://www.w3.org/WAI/References/Tablin/).

Temporal nature

Closely related to its sequential nature is speech's temporal nature (Schmandt, 1994, p. 102). The spoken message must be perceived synchronously with its production. When a word has been spoken, the speaker must repeat the word if the listener did not hear it. This requires spoken interaction to devote reasonable resources to interactive error management in which misheard and unheard information can be repeated and confirmed.

Speech interaction also requires reasonable mental resources. While other modalities, such as vision and motor actions, can be used effectively while the user is speaking, this works only in semiautomatic tasks, such as walking, and, in most cases, driving a car. Things such as active reasoning interfere with the understanding and production of speech. This is one reason why speech-based interfaces cannot be a solution for every interface problem (Shneiderman, 2000). To some extent, human reasoning takes place at linguistic level and therefore competes for the same resources as spoken communication. Remembering and deduction may not be successful when a user is required to speak, and problem-solving skills in general are reduced in this context (Shneiderman, 2000). This effect can strengthen when users must carefully phrase their utterances, as is often the case with speech-based user interfaces and their limited language models. This, combined with the temporal nature of speech and spoken communication, means that a badly designed speech user interface can significantly handicap users' mental capabilities.

The temporal nature of speech extends also to speech production. In spoken dialogue, people often produce a sentence "on the fly" instead of determining the phrasing of their utterance beforehand. This results in speech featuring hesitations, self-correction, and other irregularity (Huang, Acero & Hon, 2001) that is not usually present in written language. Disfluencies in speech production have been used in the studies of human speech production process since different kinds of errors provide information on the processes involved (Lewelt, 1999).

Error-proneness

Speech communication requires error correction since speech is transmitted over a noisy channel (McTear, 2004, p. 81), which can render the message incomprehensible and cause misrecognitions. In face-to-face dialogue, the distortion can be because of external noises and insufficient sound volume. The above mentioned errors in human speech production form a source of errors as well. In machine-mediated communication, the technical components contribute further to the degradation of signal. Microphones have their own limitations, and telephone lines, especially cellular phone connections, have rather low bandwidth.

However, in most cases, the greatest variability in speech is caused by speaker differences and variations within a single speaker (Huang, Acero & Hon, 2001, pp. 417–418). This variability results in the need for stochastic models in speech recognition. The pronunciation of words differs greatly depending on the context – the surrounding words, the speaker's mental and physical state, etc.

Ambiguous nature

One more source of errors in spoken interaction is the ambiguity of language (McTear, 2004, p. 91). Surprisingly many words, phrases, and sentences can have more than one meaning. In most cases, the context (both the preceding discussion and the physical context), the dialogue partners' knowledge of each other, etc. guide listeners to consider only the intended meaning of an utterance. In spoken language, prosody can aid further in resolution of ambiguity (Cutler, Dahan & van Donselaar, 1997). In human–computer dialogue, the ambiguities can be considered in the design phase. Since speech recognition grammars are often rather limited, the dialogue restriction can be designed so that ambiguities in user input remain minimal. Similarly, system utterances must be designed such that misunderstandings in the other direction should not arise. Because speech recognition errors can result in ambiguity, longer user utterances can easily result in enough misrecognized words to make determining the meaning of the utterance difficult or impossible.

Public nature

Speech is a public form of communication (Schmandt, 1994, p. 103). Since speech sound spreads almost omnidirectionally, it can be heard by others who are present, even if the speech is not directed to them. When two people try to avoid being heard by others in public, they can lower their voices and whisper. This makes it hard also for one dialogue partner to hear what the other is saying. When machines are partners or mediators in dialogue, headphones can be used to listen to speech without the audio leaking to the ears of others. However, one's own speech is still audible to others. This limits the use of speech in confidential applications. For example, teenage girls do not feel comfortable talking on telephone in front of their parents (March & Fleuriot, 2006).

Interaction objects are invisible

In completely speech-based interaction, the interaction objects are not visible. Especially in human–computer dialogue, there is usually a task that involves some objects, such as e-mail messages, and operations that can be performed on these objects. The strength of graphical interfaces and the reason for their user-friendliness is that they make these interaction objects visible. Operations can be found via menus, and a user does not have to remember the name of a particular command or a list of all available operations.

When interaction objects are invisible, users must somehow know what operations can be performed and what one must say to make each operation happen. In human–human dialogue, what operations are possible can usually be worked out from context to the necessary extent and one can use speech to ask about the possibilities. Also, the entire scope of natural language is available to express the operations. Since computers have limitations and can understand only subsets of natural language, the set of possible operations may not be clear to a user. A well-designed speech-based user interface supports those spoken expressions that users would naturally use and effectively provides information to users on what operations are available. However, there is great variability among users in how they like to express different operations, and the scope of a single application is usually very hard to express concisely.

2.3 Spoken Interaction

The features of speech result in challenges that require solutions in spoken interaction. Human-human interaction is the most common form of spoken interaction and the basic rules of this interaction should be followed. Since there are at least two partners in interaction, the temporal nature of speech requires turn-taking methods. The nature of speech as an error-prone communication channel requires error correction protocols to be applied in spoken interaction. Finally, people as conversation partners easily adapt during interaction, which can both cause problems and help solve them. These issues are discussed in the following.

Human-human interaction as the reference

Speech is primarily used in human–human interaction, making this the source of people's knowledge of spoken interaction. Because people base their expectations on human–human interaction (Weegels, 2000), it is important for computers to work in a similar manner. While a wide range of interaction types naturally occurring between people can be considered speech-based, computers are currently capable of supporting only some of these. When it is not possible to mimic human-human interaction, it may be better for machines to work in a distinctly different manner so that users know not to follow human–human interaction models. This way, some of the limitations can be overcome, and, in some cases, a computer can actually work more efficiently when it works in a "computer-like" manner. For example, visually impaired users often prefer synthetic voices with very little prosodic variety since the predictability of a voice improves the comprehension. Benefits of limited language in input have been studied in text-based (Jackson, 1983) and speech-based interaction (Sidner & Forlines, 2002) and there is an attempt to provide standardized subset grammar of natural language (Tomko & Rosenfeld, 2006).

Speech as a communication modality can provoke anthropomorphism in users. While, on a certain level, people tend to treat machines as social actors, as they would humans (Reeves & Nass, 1996), speech is thought to strengthen this reaction (Suhm, 2003a). Users may assign human qualities to systems and easily personify an interface. Because of this, creating an appropriate persona for a spoken-dialogue application is important (Kotelly, 2003).

A set of rules used in human–human dialogue has been identified by Grice (1975). The so-called "Grice's maxims" state that the quantity of information provided in each contribution should match the requirements of the task at hand, no more or less. The quality maxim says that contributions should be "genuine and not spurious." The contributions should also be "appropriate to immediate need," and the manner of contributions should be good; i.e., speech should be clear and understandable. Grice argues that humans follow these rules in normal task-based dialogue even if that may not seem to be the case. If my contribution does not seem to provide you with enough information, you do not assume that I am breaking the first maxim but figure that you have perhaps not understood something. The maxims can also be apparently violated on purpose – e.g., in irony, and because of the rules a listener is able to recognize the irony. It is interesting to consider what kind of reactions a computer breaking the maxims would cause in users. In many cases, users would probably assume that the software is not working properly, instead of assuming that the computer is being ironic.

Turn-taking and initiative

Because of the temporal nature, the spoken interaction is usually synchronous, with the exception of voice mail and some experimental systems (Schmandt *et al.,* 2002). When synchronous, both participants must have constant focus on the interaction and they must share turns to maintain successful interaction.

Turn-taking is an important part of spoken interaction. Two people cannot speak at the same time and understand each other. This does not mean that people would not speak at the same time. They often do, but actually as part of the turn-taking process (Sacks, Schegloff & Jefferson, 1974). Usually the spoken message, prosody, gesturing, and facial expressions signal when a user is finished and is ready to let the other speak (Schmandt, 1994, p. 6). These cues are ambiguous and therefore sometimes misunderstood. More importantly, people often want to say something while the other is not finished yet and therefore speak at the same time on purpose.

Turn-taking is a very complex process in human–human dialogue, and modeling it precisely in human–machine dialogue is not possible with current knowledge. Therefore, speech-based systems use more limited, strict turn-based interaction wherein a computer provides dialogue turns to users rather explicitly. The limitations of turn-taking are perhaps the most significant factor that currently differentiates the interaction style of human–machine spoken dialogue from that of human–human dialogue. The spontaneous interaction between humans and the flexibility it gives to interaction are in great contrast to the strict turn-taking and carefully uttered speech that usually constitute human–machine dialogue. While the details of human–human interaction are not replicated, the basic rules should still be followed. Phrasing the utterances so that it is clear that a computer has finished its utterance is important. Appropriate prosody, so that the ends of sentences are clearly signaled, is also a relevant but secondary signal (Wichmann & Caspers 2001).

In human–computer dialogue, the turn-taking is limited by the computer. Current methods still cannot understand prosody effectively enough, and speech recognition may not produce good enough results for intermediate recognition results to be used to reason about turn-taking. Because of this, the computer usually just waits for the user to stop speaking (Huang, Acero & Hon, 2001, pp. 422–424). This is the most unambiguous and easy to detect turn-taking signal from a user, which does not require unnatural user behavior, such as pressing a button after each utterance. When a computer tries to detect that a human is taking a turn, there are comparable limitations. Some systems are not able to listen to a user while speaking. In other systems, a user may be able to interrupt system speech by speaking, which makes interaction more efficient but users must avoid use of acknowledgements, such as 'uh-huh', common in human-human interaction. In human–human dialogue, concurrent speaking occurs reasonably often and there is a significant amount of acknowledgements, or "back-channeling" (Schmandt, 1994, pp. 196–197), that are used to inform the other dialogue partner whether a message is understood, agreed, etc. Support of acknowledgements is not common in speech-based systems, but attempts in this area have been made within the research community (Nakano *et al.*, 1999; Ward & Heeman, 2000).

In addition to dialogue turns, dialogue can be analyzed with the concept of initiative. In a dialogue, one of the partners can be considered to have the initiative at any given time. In human–human dialogue, the initiative moves flexibly between the participants and "plays a role in the structuring of the discourse" (Walker & Whittaker, 1990). While most of the current spoken-dialogue systems aim at flexible mixed-initiative interaction, the flexibility is very limited compared to that of human–human interaction (McTear, 2004, p. 110). In many cases, it means just that a user can provide more information than the system explicitly requests. Error correction is also a part of interaction where a user can commonly take initiative. While mixed-initiative interaction is theoretically a more efficient manner for users to express their needs, system-initiative interaction is often preferred by users (Williams *et al.*, 2003), especially if they do not have much domain knowledge (Witt & Williams, 2003). For example, Walker *et al.* (1998) report that the flexibility of mixed-initiative interaction results in confusion in users. Speech recognition and understanding technology cannot currently support truly natural language; therefore, users need to know to some extent the limitations of the system. In system-initiative dialogue, the explicit questions guide a user much better than do the open prompts in more open dialogue strategies. However, when graphics are included into the interaction, they can provide the necessary guidance, thus removing the need for system initiative prompts (cf. Sturm *et al.*, 2003).

Completely user initiative human–computer dialogues can also be successful. These systems require more user learning but once the users master the systems, they can be efficient tools. Command and control type of applications, such as those available in automotive systems, are one example.

Error correction

Even in human–human communication, error correction is frequent. People often do not hear or recognize what has been said and ask for repetition or clarification (e.g., by saying, "What?"). Non-verbal communication can also signal a need for error correction. In addition, people can mishear each other and find this out only later when the dialogue partner's utterances do not match what one would accordingly expect. This is one area where Grice's maxims are used.

Errors similar to those in human–human spoken communication are found also in human–computer dialogue. Speech recognition errors are always present in spoken communication. Since computers have a more limited understanding of language than humans do, errors are more common. In addition, the computer has more trouble detecting errors.

Error correction can be seen as part of grounding activity, where dialogue partners try to maintain a common understanding of the dialogue state. The two partners can be considered to have separate private models of the dialogue (Cahn & Brennan, 1999), and keeping these in synchrony is necessary for successful interaction. Since errors occur in spoken interaction easily, language is ambiguous, and the scope of natural language is practically unlimited, it is important to make sure that the dialogue partners have a common understanding. Grounding can take place with numerous different methods. One can explicitly ask the dialogue partner questions. One can also include implicit confirmations in the utterances and assume that the dialogue partner will correct if these reveal any misunderstandings. In face-to-face conversations, gestures and facial expressions can signal understanding, or lack of it. In human–machine dialogue, explicit and implicit confirmations are commonly used to maintain grounding. What computers lack is the ability to determine whether the dialogue partners have a common understanding. This detection, and initiation of the necessary correction, is usually left to the human. To better support grounding in human–computer dialogue, Heeman *et al*. (1998) recommend that grounding should be separated from the rest of the dialogue model.

People's adaptiveness

While the basic design principle for human–computer interfaces, including speech-based interfaces, is to design them to be as natural as possible for users, the user can also adapt to an interface. In fact, people are extremely flexible and adapt easily. If there are problems in the interaction, people try different approaches and then tend to stick with one that works. In speech-based user interfaces, this often means that people use those words that they have found the system to understand. It is our experience that when errors occur, people start to simplify the language they use. They start to use ungrammatical inputs such as "from Central Square, to Hervanta, what time?" This is based on users' idea of how computers work and has been detected already in text-based natural language interaction (Guindon, 1988). The natural direction of adaptation in user behavior is toward this simple model, and in controlled experiments users return to a more complex model of interaction only after several errors with the simple model prove that it is not supported. In a real-life situation, these users would probably have already quit trying at this point.

Another form of adaptation in error situations is to speak more slowly, use longer pauses between words, and hyperarticulate. Pause elongation and interjunction when users need to repeat their input are illustrated in Figure 2. These features, while often helpful in human–human dialogue, can reduce recognition rates since speech recognizers are commonly trained with fluently read speech (Oviatt, 1996b).

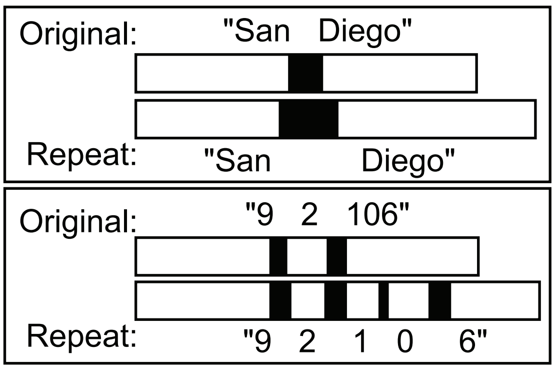


Figure 2. Users introduce more and longer pauses to speech in error situations (after Oviatt, 1996b).

Different wordings in system prompts result in different user behavior (Bulyko *et al.*, 2005), and therefore selecting appropriate phrasing is important when designing speech user interfaces. Oviatt reports (1996b) that many sources of variability in users' speech can be reduced two- to eight-fold through appropriate design of system behavior. In designing speech-only interfaces, an important fact to consider is that people tend to mimic the language used by the conversation partner. This "linguistic convergence", "alignment", "accommodation", "entrainment" or "adaption" (Pearson *et al.*, 2006) can be used to guide users to speak as required (Oviatt, 1996b). The alignment is strong in human–computer interaction and has been found to depend on users' expectations on the sophistication of the computer (Pearson *et al.*, 2006).

Users' capability to adapt can be exploited in many cases, but people cannot adapt to everything. For example, pausing between words is an unnatural way of speaking, and such adaptation, even if possible, is easily forgotten under stress. Because of this, user adaptation cannot be an overall solution (Oviatt, 1996b). The adaptiveness can also cause significant problems. For example, if a user is using correct words but there is a speech recognition error for some reason, the user may start unnecessary experimentation with other phrasings.

2.4 Speech and Dialogue Systems Technology

Technology limits spoken human–computer interaction significantly. Since the research on automatic speech recognition started in 1960's, a lot of progress has been made so that the technology is now usable in real life. However, there are still significant limitations and it is important to understand these when speech user interfaces are concerned. In the following, the current state of the technology is discussed based mostly on recent books on the subject.

Components of speech-based systems

Spoken dialogue systems need components for understanding and producing speech and for controlling the system behavior. Commonly the systems have at least a speech recognizer, a speech synthesizer, and a dialogue manager. Natural language understanding is also commonly a separate module and natural language generation modules can be found as well. In addition, systems usually have some sort of backend, e.g., a database to handle the actual domain data. Finally, technical components taking care of the audio input and output, such as telephony cards, can often be found from a dedicated module. The components of spoken-language-dialogue systems can be combined in different ways.

Information flow in its basic form goes from audio input via speech recognition and language understanding to dialogue management, which consults the backend, then to natural language generation, and finally to speech synthesis (McTear, 2004, p. 80). Such a linear pipeline architecture works well when information flow is indeed this simple but has trouble coping with the challenges of interactive systems. Therefore, more flexible architectural structures are common in speech-based systems. Many systems use a star topology wherein a central component controls the other parts of the system. This enables more flexible information and control flow between components. The Galaxy-II reference architecture (Seneff *et al.*, 1998) is one such solution. However, the interaction model still tends to be strictly turn-based. The blackboard architecture, as demonstrated in Hearsay-II (Erman *et al.*, 1980), enables concurrent processing of speech on many levels. Speech recognition could produce results that are immediately processed on higher levels while more speech is being recognized. Agent-based architectures also have been used in speech-based systems; one example is the Open Agent Architecture (Martin, Cheyer & Moran, 1999). Modeling different components as agents enables modular partitioning of the implementation. The Jaspis architecture (Turunen & Hakulinen, 2000a) uses small agents and blackboard-style centralized information storage to enable adaptive selection of appropriate agents for each situation.

The temporal and concurrent nature of human–human spoken interaction has not been fully supported by human–computer spoken-language-dialogue systems so far. Providing such flexible interaction requires a lot not only from the speech understanding but also from the architecture. Information flow must be real-time so that information on the start of the user's utterance is processed while the user is still speaking. In addition to the blackboard architecture, some later architectures, such as TRIPS (Allen, Ferguson & Stent, 2001), provide such flexibility.

Next, the speech technology components are discussed to the extent relevant to the design of speech interfaces. The order is that of the pipeline architecture.

Speech input

One element that is significant in shaping the style of interaction is the flexibility of a system in listening to the user's speech. In the most basic form of interaction, the user may speak only when the system explicitly indicates a turn and must utter the input during a certain timeframe. A slight improvement involves using more advanced voice activity detection (VAD), so that speech is detected only when the user is speaking. This removes the need to utter input immediately when a turn is given. Both of these systems can be considered single duplex, since the system cannot listen to a user while it is speaking.

So-called barge-in functionality (McTear, 2004, p. 90) makes the interaction more effective by allowing users to speak over the system prompts and interrupt them. Depending on the technical setup, this can reduce the recognition rate in some cases. The use of barge-in enables some user interface solutions, such as rather long lists by allowing the user to utter the selection over the prompt upon hearing the option desired.

Full duplex systems can listen to the user's speech while producing spoken output, unlike barge-in solutions, where the system stops speaking when it hears the user speaking. This enables back-channeling, which is important also from the turn-taking standpoint.

Speech recognition

Speech recognition is commonly considered a probabilistic process (López-Cózar & Araki, 2005, p. 20) wherein the most probable cause for a given audio signal is found on the basis of a model of language. The recognition problem can be presented as the following Bayes' rule.

$$\arg\max_{word} P(word \mid acoustics) = \arg\max_{word} \frac{P(acoustics \mid word)P(word)}{P(acoustics)}$$

The result of a single recognition task is a word and its probability. There can also be a list of words and their probabilities, a so-called N-best list (Huang, Acero & Hon, 2001, p. 664). In general, the results are the most probable words as judged from the acoustic evidence and a language model. The result may or may not include the words that were uttered. When an application receives a speech recognition result, it can never be sure what the user said. The system must act on the basis of the probabilistic result. The probabilistic nature of the problem is why error correction is important. It is always possible that the application chooses to do the wrong thing, and such errors must be correctable.

Language models

Speech recognition produces results based on an audio signal and a model of language. This model consists of two parts: an acoustic model and a language model (McTear, 2004, p. 84).

The acoustic model specifies the acoustic properties of single units of language. The unit size can be a phoneme, usually modeled using trigrams that include the context of the phoneme, or the units can be longer, e.g., syllables or even words. The acoustic model is created by statistical means from a large corpus of speech.

The two major language models currently used are context-free grammars and N-gram models. Context-free grammars are often used in spoken-language-dialogue systems as they can be hand built and work reasonably well in limited domain system directed interaction. In large-vocabulary speech recognition, a stochastic model is required. These models are created from corpora using statistical methods. N-gram models, often tri-grams, are the most common stochastic models. They model the probabilities of words from the context – i.e., the previous words. (McTear, 2004, p. 86; Huang, Acero & Hon, 2001, pp. 545-585)

The most important aspect of the language model in speech recognition, from the communication point of view, is that it limits the language that can be recognized. Only a limited number of words is included in any given language model, and only these words can be recognized by the speech recognizer. When a user utters a word that is not included, it cannot be recognized correctly. Instead, the speech is either rejected or something within the given language model is produced as the result. This is called the out-of-vocabulary (OOV) problem (López-Cózar & Araki, 2005, p. 19). The probability associated with the recognition results in question can serve as an indication of the fact that the recognition is not correct, but, unfortunately, it is not always a reliable figure with current recognizers. Avoiding OOV errors by expanding the grammar is not always an acceptable solution either; the larger the language model, the more error-prone the recognition is. Developers must achieve balance between recognition errors and OOV errors.

Attempts to address the OOV problem include the usage of multiple grammars and recognizers at the same time. Gorrell, Lewin, and Rayner (2002) used a statistical language model to provide effective help information to users when a grammar-based recognition fails. Gustafson, Lundeberg, and Liljencrants (1999) used two recognizers; the result of a phoneme recognizer with a completely open grammar was compared to a grammar-based result to spot the out-of-vocabulary utterances.

A language model can also limit the recognition by restricting the allowed order of words. A well-designed grammar allows only reasonable word combinations and thus eases the work of the speech recognizer. However, since speech is not always grammatical and involves hesitations etc., "grammars are almost always incomplete" (Huang, Acero & Hon, 2001, p. 582).

An alternative to recognizing an entire utterance via a given grammar is "word spotting," where a speech recognizer tries to spot given words or phrases and ignore everything else. This solves the OOV problem but in practice is challenging to implement, and, when this approach is used, significant words can be lost, and thus the actual meaning of the user's utterance. (López-Cózar & Araki, 2005, p. 18)

Speech recognition errors

Because of the nature of speech, speech recognition is a probabilistic process, and errors form an unavoidable part of speech-based interaction. Understanding the different kinds of errors that can occur in speech recognition and their effects is an important part of the design of speech-based user interfaces.

On word level, an error can cause additional words to appear in recognition results, words may be replaced by other words, and words may not be included in the recognition result at all even if they were uttered. These errors are called insertion, substitution, and deletion errors, respectively (Huang, Acero & Hon, 2001, p. 420). An incorrect speech recognition result may consist of any combination of these errors. Words may also be recognized when no speech was present, causing a "false alarm" (López-Cózar & Araki, 2005, p. 19).

A speech recognizer may also reject the entire utterance, if it cannot find any suitable match for the input from the grammar. How often this occurs is greatly dependent on the speech recognizer in use. Some recognizers tend to return a result no matter what the input is. Out-of-vocabulary inputs and even random noises that have nothing to do with speech can cause speech recognition results that cannot be distinguished from viable results. On the other hand, some recognizers, especially when configured inappropriately, reject even most of the perfectly valid input utterances. This can occur in, for example, cases where the user's accent conflicts with the acoustic model used in the recognition.

Recognition rate metrics

Speech recognition engines are often compared using precise metrics. For example, word error rate (WER) is often reported when speech recognizers are described (Huang, Acero & Hon, 2001, p. 420). The WER percentage is calculated according to the following formula:

$$WER = \frac{deleted + substituted + inserted}{words\ in\ original\ message} * 100\,.$$

While metrics like WER are seemingly comparable, in practice they do not reveal much about a speech recognizer unless the testing conditions and procedure are carefully described. Testing in carefully controlled laboratory environments provides different results from testing in settings that are closer to real life. For example, Huang, Acero, and Hon report a rise in error rate from 4.5% to 8.6% for a dictation task when noise was added to the audio. They also report error rates ranging from 0.77% to 36.7% for different recognition tasks (Huang, Acero & Hon, 2001, p. 12).

Simple metrics that summarize speech recognition quality as single figures do not indicate much about how well the recognizer suits a particular task. It is more the context and type of errors that matter. For example, in a spoken-language-dialogue system with a large grammar aiming at natural language support, the types of errors occurring are significant. If natural language understanding spots only important concepts from the speech recognition result and can ignore the rest of the result, the location of speech recognition errors is important. A recognition rate of 95% may be unacceptable if the errors are always in the keywords. By contrast, if keywords are recognized correctly and ignored words are often incorrect – which may well be the case, since short words such as "a," "the," "and," and "it" can be hard to recognize – a recognition rate of 80% may be perfectly adequate. Using a keyword error rate (i.e., calculating the error rate for only those words that are necessary for successful semantic interpretation of user utterances) provides a more significant metric with many systems where robust parsing is used (López-Cózar & Araki, 2005).

Completely random scattering of the errors across the recognition results is not as bad as a situation where the errors are concentrated in a few words that are of high importance. Perhaps the worst case would be constant errors in distinguishing between the words "yes" and "no," which would render the common last resort of error management, yes/no-type questions, useless, even harmful.

The cost of errors is also dependent on where they occur. Errors in irreversible operations are, naturally, very harmful. Since the probability of errors can be controlled by selecting a more or a less strict grammar and guiding the user to speak accordingly, a dialogue designer must take advantage of this possibility and consider the costs of errors and error management, such as confirmations and corrections, to minimize the overall cost. Interface design can reduce the problems caused by recognition errors. For example, Sanders and Le (2004) report that the effect of WER on task completion was reduced during system development, "due to improved strategies for accomplishing tasks despite speech recognition errors."

Natural language understanding

To be able to respond to user input, a speech-based system must understand the meaning of the input to some extent. In the area of natural language processing, computational linguists have developed a rich set of tools for language understanding. However, in speech-based systems, understanding the meaning of user utterances is usually a rather easy problem to solve. In most cases, an application defines a context strict enough so that the ambiguity of language is not a problem. The language that can be recognized in the speech recognition is limited by the language model used, so that the language understanding need only work within this scope. The language understanding can also be incorporated into some speech recognition language models. For example, the W3C Speech Recognition Grammar Specification (W3C, 2004b) features tag elements that can be used to assign variable values and even do computation when corresponding grammar rules match during speech recognition. This way, instead of a list of words, the speech recognition results can be in the form of, for example, a frame structure that is readily processable by a dialogue management engine.

One special problem in speech-based interaction differentiates the need for language understanding from that in text-based systems. The speech recognition errors, coupled with the fact that speech, especially spontaneous speech, is not, by default, grammatical, makes understanding challenging (McTear, 2004, p. 91). Depending on the grammar used, recognition errors can sometimes replace a word with one from a completely different part of speech. With hesitations and self-corrections added to the recognition errors in effect, one can obtain recognition results that do not appear to be sensible language. Because of this, robust parsing methods are commonly employed (Huang, Acero & Hon, 2001, p. 874). Text-based language processing methods aim at parsing the entire utterance, e.g., by unifying it to some grammar. Such methods are difficult or impossible to apply to speech recognition results. Robust parsing methods, on the other hand, often parse the utterance bottom-up (e.g., Allen *et al.*, 1996) and may just spot separate concepts from the input. It is hard to implement such things as reference resolution with these methods, but, as discussed, they are not usually necessary, since in the application context simple utterances can be understood well enough without any complex analysis.

From the user's point of view, understanding the limitations of natural language understanding (NLU) can be important. However, in most cases, these match the restrictions of speech recognition, such that anything that can be uttered within the limitations of speech recognition can be understood by the system. The problem is actually the user's prejudice concerning computers' language understanding capabilities. People do not commonly believe that computers could understand natural language and, when problems arise, assume the computer just cannot understand such complicated language.

Dialogue management

Dialogue management is the component that models the flow and structure of the dialogue from the computer's point of view and decides how the computer will advance the dialogue in its turn. In most cases, dialogue management in spoken-language-dialogue systems is rather straightforward, based on finite state machines or a "form-filling model" (McTear, 2004, pp. 113–116). The dialogue management always imposes some limitations in terms of the interaction possible between a computer and a user. It is important that the user know the limitations of dialogue management. In systems where the computer holds the initiative in the dialogue, the active role of the system makes sure that the user only follows the computer's lead. When initiative is mixed or always given to the user, the limitations and possibilities of dialogue management need to be somehow communicated so that the user can use the system to its full extent and not try to do things that are not supported.

One of the important tasks of dialogue management is error handling. This is a challenging task. Luperfoy and Duff (1997) have identified five steps in the error management process: detection, diagnosis, determining a repair plan, executing the plan, and closure and return to the primary dialogue. Turunen and Hakulinen (2001) suggest division of the last step to make informing the user about the error an explicit step. They also add the prevention of further errors to the process. In practice, error management often consists of providing users with enough confirmation information that errors can be noticed and corrected. While the onus is given to the user, the detection of users' error correction attempts is not always trivial and great care must be given to its implementation.

The dialogue management cannot be trivially evaluated in isolation. The information a dialogue manager receives is highly dependent on the speech recognition and understanding modules, and the dialogue management capabilities should match the other components. Evaluation of the general usability of a speech system assesses, in part, the dialogue manager. These evaluations are considered in the next chapter and include some methods aimed at focusing on testing solely the dialogue manager.

Speech synthesis

Speech synthesis is the process of generating speech from text or from other machine-based information representation. Most commonly, speech is generated from text. Several such text-to-speech systems are available for numerous languages. Speech synthesis differs from generating a spoken message by concatenating prerecorded sound files in that speech synthesis can generate any message. There exist limited-domain speech synthesizers that make the difference fuzzy, since many of these are based on concatenation technology as well. Technology used to generate speech varies from the concatenation and processing of segments of prerecorded audio (e.g., diphone and unit selection synthesizers) to the modeling of the human speech channel by means of simple filter models (formant synthesizers) (Schmandt, 1994, pp. 91–93) or even complete three-dimensional physical models of human speech production apparatus (articulatory synthesizers) (Badin *et al.*, 1998). The technology used, along with other factors, limits the quality of synthesized speech. The best commercial synthesizers, requiring hundreds of megabytes of memory, can sound like humans for short segments of material. However, longer texts often reveal limitations and mistakes in prosody modeling, pronunciation, and other areas. Synthesizers using more modest resources can produce more machine-like but still reasonably understandable speech.

Naturalness and intelligibility

When speech synthesis is being evaluated, researchers often divide the quality assessment into two parts: naturalness and intelligibility (Schmandt, 1994, pp. 95–96). The preprocessor of the synthesizer can also be considered separately (SpeechWorks). In general, the more natural a speech synthesizer is, the better it is considered to be. Unnatural voices are disliked because they cause more cognitive load to listeners. Also, the quality of synthesis affects users' perception of the quality of the entire spoken-dialogue system, influencing even global-level user satisfaction (Möller & Skowronek, 2003). It can also affect users' behavior. Nass *et al.* (2003) report that synthetic speech caused participants to exhibit less disclosure of personal information to a synthesized-speech user interface than to either recorded speech or a text-based interface.

Intelligibility is not directly connected to naturalness, although the two do often go together. A speech synthesizer may produce speech that is easy to understand and follow, at least in short utterances, but may turn users away due to its unnatural nature. While the intelligibility of synthesized speech is close to that of natural speech, there is still a difference (Venkatagiri, 2003). The effect also depends on the listener. For example, older people seem to have more problems in the comprehension of synthesized speech than young ones do (Eskenazi & Black, 2001). The quality of recorded speech suggests that it should be used wherever possible instead of speech synthesis. The studies of mixed synthetic and natural speech suggest that users indeed prefer the mixture of natural and synthetic speech over pure synthesized speech. However, performance metrics favor plain synthesized speech (Gong & Lai, 2003).

Intelligibility can be measured quantitatively fairly easily. Various listening tests have been used for testing of different speech synthesis solutions. They range from more traditional tests of the intelligibility of single syllables (e.g., with the Modified Rhyme Test) to testing users' comprehension of text segments several paragraphs long (Huang, Acero & Hon, 2001, pp. 836–837). The usefulness and sensibility of different tests depends partly on the synthesizer to be evaluated. For example, comparison of the intelligibility of elements shorter than words, especially without a meaningful context, does not make much sense with so-called unit selection synthesizers where concatenated units are sometimes entire words or even longer segments.

Cognitive load

Listening to speech, and producing it, requires reasonable cognitive resources, shared with problem-solving. Depending on its quality, synthetic speech may require significantly more cognitive resources than natural speech does. With low-quality synthesized speech, especially when the listener is not accustomed to the voice, intense concentration can be required if the message is to be understood. The required listening effort is often addressed in speech synthesis evaluation questionnaires (Klaus *et al*., 1993).

The cognitive overhead required for understanding speech reduces the user's capabilities in other areas. Remembering things, which is necessary in speech user interfaces due to the temporal nature of speech, is harder when synthesized speech is used.

2.5 DIFFERENT SPEECH USER INTERFACES

Speech has been used in many kinds of applications, and new areas of application have been envisioned as well. The first public speech-based applications were telephone-based systems (Cox *et al*., 2000) and currently the main areas include automotive, broadcast, consumer, defense, disabled, education, legal, medical, and telephony markets (Moore, 2005). While this thesis considers mainly spoken-language-dialogue systems, other common uses of speech technology are discussed in the following as well in order to provide larger context.

Spoken-language-dialogue systems

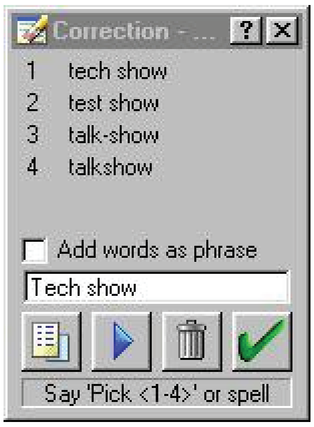
Spoken-language-dialogue systems are speech-based systems in which spoken dialogue is used in a task-oriented interaction. Most spoken-language-dialogue systems are simple information retrieval or transaction systems. For example, timetable services (e.g., Turunen *et al.*, 2005; Flycht-Eriksson & Jönsson, 1998; Strik *et al.*, 1996) commonly fall into this category. In most cases, these systems use only speech in the interaction, and the basic model of interaction is that of a task-oriented dialogue between humans. The services often aim at replacing some of the telephone-based services in which human operators have been used. This reduces costs significantly and enables new kinds of services that were not economically viable before.

Spoken-language-dialogue systems are often based on telephony (López-Cózar & Araki, 2005, p. 2). The ubiquity of cellular phones in the industrialized world, coupled with spoken-dialogue technology has enabled access to services from anywhere. This mobile access also justifies many speech-based services that would not be used when graphical access to the World Wide Web is available. Mobile use extends the context of use: services are not necessarily used only indoors where personal computers reside. Instead, they are used wherever needed. The mobile context increases the challenge for speech technology since the audio signal is more likely to include varying background noise. Users may also have to divide their attention between different things, not concentrating entirely on interaction with the spoken-language-dialogue system. This often yields various kinds of "off-talk" – speech not directed to the system (Opperman *et al*., 2001).

Command and control and dictation in desktop computing

In the traditional personal computer domain, speech has been used most commonly in dictation. Numerous dictation applications, from several manufacturers, have been made available in many languages (Huang, Acero & Hon, 2001, p. 926). However, the number of dictation products available has actually decreased in the 2000s, partly due to rearrangements in the industry. The systems have not achieved the widespread popularity for which their advocates have hoped. Since keyboard text entry is an extremely common input method, it has also become natural to people, so it can be more natural than speech recognition (Karat *et al*., 1999).

Computer-based dictation requires a very large vocabulary and therefore is a very challenging task. However, it is possible to use speaker-dependent or adaptive recognizers that can be individually trained for each user. This approach is used in all dictation systems (McTear, 2004, p. 87). Furthermore, the speech in dictation systems is similar to read speech, not spontaneous speech, which simplifies the recognition task significantly (McTear, 2004, p. 88). Finally, as seen in Figure 3, error correction can use the graphics on the computer screen as well, to make error detection and correction more effective and pleasant (Larson & Mowatt, 2003).


Correction - ...

1 tech show
2 test show
3 talk-show
4 talkshow

☐ Add words as phrase

Tech show

Say 'Pick <1-4>' or spell

Figure 3. Error correction window from IBM ViaVoice. The window presents an N-best list after a user activates the error correction with the "Correct that" command.

Dictation has been most popular among those who have not traditionally typed their texts themselves, including doctors and lawyers. In many of these cases, the dictation task is rather limited as well, in terms of both the dictionary and grammar applied. For example, in many cases, medical diagnoses use only a very limited subset of language, and special dictation products for these tasks yield much better recognition rates than general dictation systems do (Huang, Acero & Hon, 2001, p. 926). Another user group is those who have problems typing. Repetitive stress injuries are a common result of extensive computer use and have forced many people to use dictation since their condition does not allow them to type anymore (Huang, Acero & Hon, 2001, p. 929). However, for those who can type, using the keyboard is usually a more efficient way of inputting text than dictation is. This is mostly due to the low efficacy of error correction in dictation. Error correction times have been reported to be up to three times as long as the actual text entry time (Karat *et al.*, 1999).

Dictation applications often include also a command and control type of functionality, so that the computer can be controlled entirely via speech. Usage of these systems is mostly limited to those who are for some reason incapable of using a mouse since they provide access to a graphical interface, which is not designed to be compatible with speech. While speech can be reasonably efficient in selecting menu items, operations where the mouse pointer must be operated are tedious. Such spatial referencing is a weak point of spoken interaction.

Vehicle command and control

Speech interaction leaves the hands and eyes free. Because of this, speech is already commonly available in cars. Radio and cellular phone control, among other non-critical operations, can be performed via speech (Heisterkamp, 2001). The reason for car companies' interest is the "driver's hands and eyes busy" situation; speech technology is considered to enhance safety. The use of speech should provide greater safety than does the situation in which the driver controls the radio manually, for example. Speech output is also common, especially in the popular GPS-based navigation aids that provide spoken guidance and enable the driver to keep the eyes on the road and traffic signs. Speech input is also coming to navigation systems (Minker *et al.*, 2003). Aeronautic applications form another area where the usage of speech has been studied for a long time.

The "busy eyes" situation is also commonplace when a user is walking. While the hands may be free, users' eyes are busy, limiting also manual control of devices such as telephones. In addition, the cognitive load from driving or walking reduces the resources available for the speech-based interaction. On the other hand, motor actions and verbal processing are handled by different, rather independent parts of the human brain and therefore do not interfere with each other (Shneiderman, 2000).

Mobile and ubiquitous computing

In addition to being a natural manner of communication, speech has some technical strong points. Speech input and output devices can be physically very compact (Rosenfeld, Olsen & Rudnicky, 2001). A loudspeaker providing decent-quality speech output at a low volume level requires only a few square centimeters of space, and a microphone can be even smaller. Because of this, the use of speech is tempting in mobile devices, where the physical size of the hardware limits input and output capabilities. Mobile context of use introduces also various hands and eyes busy situations where speech can be efficiently used. Speakers and microphones are also cheap compared to many other components and can be built to be rather robust so that they can be applied in ubiquitous computing scenarios.

The omnidirectional nature of audio can be a problem or a benefit in the ubiquitous computing scenario. One loudspeaker can reach many users in all directions and break through their attention barriers. This feature has traditionally been exploited in audio warning signals such as fire alarms and can be used in speech-based alarms as well (Stanton, 1993). More subtle speech-based solutions could, e.g., support users' awareness of workplace activity (Kainulainen, Turunen & Hakulinen, 2006).

The ubiquity of speech communication enables also the use of speech as an implicit input modality. People use speech all the time in meetings and other workplace discussions. Automatically recognizing this speech enables computers to provide relevant information to participants without explicit prompting to do so and facilitates later access to an audio recording of the discussion (Brown *et al.*, 2001). Speech can also be voluntarily targeted both to a machine and to other people (Lyons *et al.*, 2004).

Multimodal dialogue systems

Current speech-based user interfaces in most cases use only the speech modality. They ignore facial expressions, gestures, and postures (Cassell *et al.*, 2001), which signal understanding, turn-taking, conversation state etc. in human−human conversation. Ongoing research on multimodal interaction promises to help in this respect, but so far this element of interaction has not been addressed in publicly available commercial applications. Only the audio channel is used, and therefore dialogue with a computer bears a greater resemblance to telephone conversations than to face-to-face interaction. Furthermore, even the speech input is used to only a small extent. Prosody is in most cases intentionally ignored in speech recognition. In system output, it is utilized to make speech more comprehensible and to distinguish questions from assertions. There is continuing research into automatic processing of emotions (Schröder, 2001) and other prosodic information, but this remains at a level far from real-life application. Because of this, speech-based communication with machines is limited to the linguistic meaning of speech. Dialogues similar to those between two people cannot really be achieved, at least with current technology.

Multimodal systems have been under development for several decades. Starting from the seminal "put-that-there" system (Bolt, 1980), most of these systems have featured speech. In human–human dialogue, modalities other than speech are used continuously. Gestures and facial expressions are very important, and drawing and other tools are commonly used in more challenging problem-solving situations. For example, in collaborative physical tasks, gestures play an important role (Fussell *et al.*, 2004). Therefore, multimodal interaction promises very efficient and natural interaction. However, the fusion of multiple inputs into one message understood by the system is a considerable challenge, and similarly the fission of system output into multiple synchronous modalities is a task that still requires significant research (López-Cózar & Araki, 2005, pp. 34–36, 43). Modeling multimodal interaction is even more challenging than modeling a speech-based unimodal dialogue.

Special user groups

Speech, as a nontraditional method in human−computer interaction, using a modality other than vision and motor control, can be considered an alternative means to communicate with computers. Various user groups who have problems with common interfaces can benefit from speech technology. Examples include dyslexic people (Moreno *et al.*, 2002) and those with motor impairment (Manaris, Macgyvers & Lagoudakis, 2002) and even illiterate people. Visually impaired users make use of speech particularly often (Raman, 1996), especially speech output, to access computers. Screen-reader applications aim to provide the visually impaired with the possibility of using the same computer applications that other users do (Barnicle, 2000). Also, dedicated speech-based applications can provide access to services originally designed for standard users. Most importantly, speech-based HTML readers (Raynal & Serrurier, 2002; Zajnec, Powell & Reeves, 1998) provide visually impaired users with access to those Web services that comply with standards and recommendations. The reader applications are designed for speech-only output and therefore provide a better interface than does simply using a normal Web browser in conjunction with a screen reader. In addition, special speech-based services can be created for visually impaired users. More common, however, is for speech-based services to be developed for the general public, thus benefiting many impaired users as well, allowing them equal access to the service.

At the same time, usage of speech can exclude some users. The biggest group may be those whose mother tongue is not the one in which the services are provided. In particular, speech recognition can be very sensitive to nonstandard pronunciations common in non-native speech. For example, Glass and Hazen (1998) report error rates that more than double in cases of strong foreign accents. Indeed, speech technology requires extensive localization if it is to provide the services well in each language area concerned (Huang, Acero & Hon, 2001, p. 944). Accordingly, concern has been expressed about potential exclusion from speech services in some smaller language areas (Miettinen & Toivanen, 2001, p. 10).

3 User Guidance

3.1 Introduction

While improved ease of use in software applications has removed some of the burden borne by user guidance, there is still need for manuals, tutorials, help desks, etc. User guidance in computer systems has been an issue as long as software packages have existed. Users need guidance in one form or another. While they are happiest with human tutoring, from an organizational point of view it may be more economical to place the burden on the individual users by giving them a manual (Covi, 1995). The more widely applications are available, the easier the guidance should be from the user's perspective. Different kinds of media and designs for guidance have been developed in the past few decades, and a significant amount of research has gone into determining what kind of guidance is effective.

One important aspect of guidance design is user motivation. Users' motivation is in accomplishing something with the software application, not in studying how to use it (Carroll & Rosson, 1987). Giving an extensive manual to users easily scares them, and the manual will not be read. If finding information via the guidance materials seems to require more work than figuring out the same thing by trial and error, the guidance will not be used by most users. Another aspect of motivation is what users want to learn. Usually we assume that a user wants to learn to use a software application well. However, it can be argued that users often want only to get some task done and not understand all of the related concepts (Williams & Farkas, 1992). Users' motivations depend also greatly on the type of the application. When the purpose of a system is entertainment, extensive training is not acceptable, like it can be with a military application.

Guidance material does not help a user if it is not accessible when needed. When an application is used on a desktop computer, even large printed manuals can usually be accessed. However, these manuals are sometimes placed in storage rooms where nobody will access them. When applications are used in more varying situations, as is commonly the case with laptop computers and especially with cellular phones, carrying printed manuals is not really a possibility anymore. Only electronic documentation can travel with the user and the application.

Another accessibility challenge is that faced by special user groups. Speech-based systems are commonly used by visually impaired users, for whom printed guidance material is largely useless. Screen readers and other accessibility solutions enable access to software applications for many such users. Online help, when appropriately implemented, can be accessible to these users by means of the same tools they use to access the application.

Guidance needs to be easy to use. In many cases, an application that requires significant learning effort can be acceptable if users can find the information they need during the learning period. However, when the guidance is equally hard to use, the user is left in a hopeless situation. Both paper and digital, possibly interactive, guidance materials can have usability issues. It is possible to write a printed manual that is hard to read and use for finding specific information. Paper documents have many features that simplify their use, such as tables of contents and indices. Electronic documentation, by contrast, commonly feature search functionality, which is not the case with paper documents, but shares usability problems with the rest of interactive technology.

Knowing who the users are and what their needs are is important in documentation writing, just as in the design of the application itself. Iterative development is a good methodology also in development of manuals (van der Meij, 1992). Guidance that is in digital form can work together with the application to facilitate finding information relevant to the situation, but it can prove problematic – such as when the usage of guidance distracts from the user's work with the actual application. Even if the guidance and the application do not hinder the usage of each other, the need to switch between the application and the help material is an important source of difficulties. Multimodality has been considered one solution to this problem (Capobianco & Carbonell, 2003).

User guidance in general aims at providing users with enough information that they can effectively use a software application. This information is often considered a mental model of the system; users have in their head some kind of model of the application, on which they base their actions.

There are various kinds of guidance methods and techniques. The sections that follow provide an overview of these starting from human tutoring and discussing static and interactive digital guidance as well as printed manuals. The different techniques are not always exclusive, and they can be combined in different media and guidance approaches.

3.2 Human Tutoring

One of the most commonplace guidance methods is human tutoring and personal guidance. Very often, co-workers and technical staff in the workplace guide users in handling their problems with computers and software applications. The use of telephone help lines is another common practice in industry and academia, and many products include free help-line service for, e.g., a year. Many users are also ready to pay for such a service. Help-desk service by e-mail is common also. When the problems are serious, people prefer face-to-face guidance, while telephone and online discussions are acceptable in less serious cases (McKendree, 1986).

Human tutoring is considered the most effective form of teaching (Borenstein, 1986), and novices, in particular, may find human advising to be faster and more familiar than manuals and help systems are (McKendree, 1986). In developing intelligent tutoring systems, involving computer-based tutors, the aim is to reap at least some of the benefits of human tutoring. Human tutors and experts do not just provide direct answers to questions; especially when the question is incomplete, they try to infer the advice-seeker's plan and answer on the basis of that (Pollack, 1985), make assumptions, provide alternatives and assistance in avoiding the problem, and offer pointers to additional reference materials (Aaronson & Carroll, 1987).

Since human tutoring is so effective, there are ongoing efforts to realize these benefits through computer applications. Intelligent tutoring systems (ITS) have been successful especially in teaching mathematics, science, and technology (Graesser *et al.*, 2001). These systems commonly have a set of models on which the guidance is based. The models include a domain model; a user (student) model; and an instructor, or pedagogical, module. Commonly, intelligent tutoring systems are tutorial dialogue systems where tutoring consists of, most commonly written but lately sometimes also spoken, dialogue between the tutor and the student. In most cases, the subject taught does not have a concrete representation in the tutoring environment. Sometimes the systems present some sort of model or a simulation of the domain, such as a 3D model of a ship (Fry *et al.*, 2001).

3.3 HELP TEXTS

Help texts were the first form of guidance developed for computer systems (Cavi, 1995) and have been an actively studied subject. Many results of the documentation studies in this area are relevant to other kinds of documentation and user guidance as well. Help texts have been delivered in printed format and as online help.

Printed material

Printed documents provide a guidance method that is based on mature technology and whose usage is familiar to users. The distribution and usage of the manual is also independent of the system and its limitations and problems. Printed manuals are used with all kinds of products, and much experience exists in writing them. In computer science, there has been a significant amount of study focused on writing paper documentation. In addition to user manuals, paper documentation can take the form of reference manuals, tutorials, or "cookbooks" (Major, 1985). While this area of research is not as active anymore, many of the results can be used as starting points in the design of more interactive user guidance as well.

Perhaps the most important finding of the user guidance documentation studies is that users do not like to read manuals. Instead, they want to start to use the system as soon as possible, with manuals being read as a last resort when problems cannot be solved through trial and error. This phenomenon was named "the paradox of active users" because the active nature of users actually detracts from the effectiveness of their work with a software application when they end up with suboptimal ways of working (Carroll & Rosson, 1987). Because of this, the manual design ideology of the minimal manual was created by Carroll and colleagues. Minimal manuals are task-oriented and contain procedural guidance that improves users' learning performance (Gong & Elkerton, 1990). Minimal manuals try to take into account the characteristics of adult learners, such as that they are impatient and best motivated by self-initiated exploration (Rettig, 1991). Minimal manuals give users opportunities to explore the system themselves. However, exploration on its own does not always work. Wiedenbeck and Zila (1997) found that people with a high level of computer experience still needed exercises in order to achieve the best learning results. Minimal manuals also aim at concise usage of text space, support error recognition and recovery, and are modular (van der Meij, 1992).

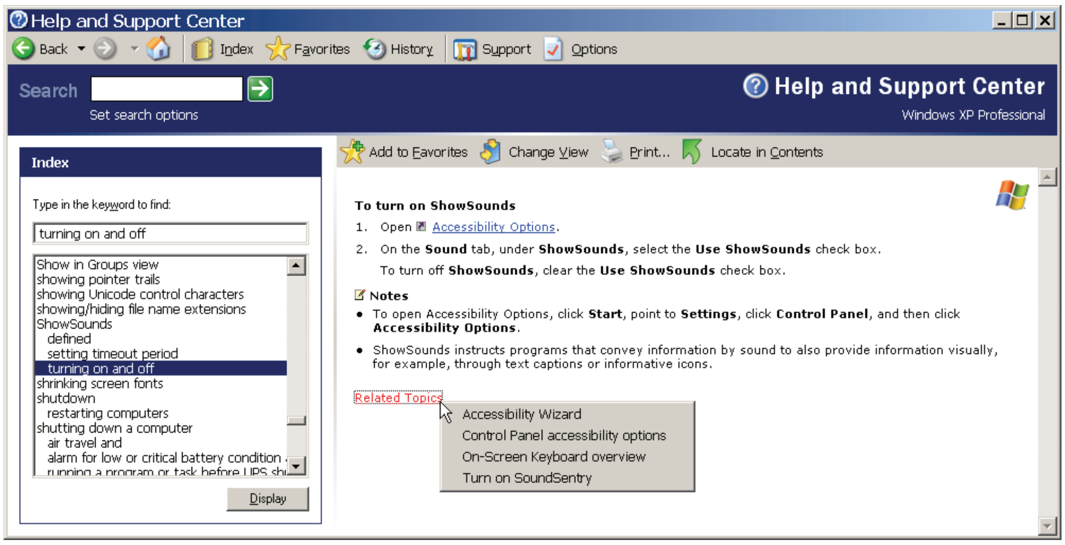
One of the issues considered in documentation studies is the level of information provided. Carroll and Aaronson (1988) separate "how-it-works" and "how-to-do-it" guidance. The latter is a help message that can teach a user how to do something at a technical level. Help messages can also be on a higher conceptual level, describing the system concepts and logic – i.e., of "how-it-works" type. Users can, sometimes, work out higher-level knowledge from "how-to-do-it" guidance and low-level procedures from "how-it-works" guidance. Black, Carroll, and McGuigan (1987) found the best learning results to occur with a minimal manual that forced users to do some reasoning themselves. Such guidance helps users in building a mental model of the application.

Also, the concept of initiative from spoken dialogue has been applied to documentation. Novick and Ward (2003) define single and mixed initiative by using the concepts of grounding and agreement so that mixed initiative interaction includes agreement. They apply the distinction to the asynchronous interaction between a document writer and a reader so that when procedural instructions are given in the documentation, a user can agree by following the instruction, or disagree by doing something else. They believe that considering this explicitly would aid the authors of documentation since they would prepare better argumentation for the instructions so that users know enough to agree, or disagree.

Online help

As computers grew more powerful, it became the norm for guidance to be in digital format and delivered with the software. This way, the documentation has greater availability, easier access, more interaction, higher accuracy, lower cost, and can exploit multimedia and artificial intelligence techniques (Duffy, Mehlenbacher & Palmer, 1992). Computers also enable efficient searching of large collections of guidance material. However, users still prefer printed manuals, especially since they may have usability problems with online help (Purchase & Worrill, 2002). Another potential improvement with electronic documentation is the possibility of using video and animations to provide, e.g., visual demonstrations. However, such visual examples of graphical interface operations seem to encourage mimicry, which does not result in efficient learning (Palmiter & Elkerton, 1991).

When the guidance is a part of the software application, it can work in tandem with the application. In GUI systems, guidance is often context-sensitive; for example, activating the help functionality automatically provides guidance for the currently open dialog box. The connection can also work the other way around, so that the guidance material can, for example, include a button that opens a dialog or application mentioned in the guidance material. Windows help functionality, as seen in Figure 4, can open operating system dialogs (in this case, the "Accessibility Options" dialog) and provide searching facilities, hypertext links to related topics, and many browsing utilities. While such possibilities are potentially useful, the way the actual help text is written is more important than the mechanics of the help system (Borenstein, 1986).



Type in the keyword to find:

turning on and off

Show in Groups view
showing pointer trails
showing Unicode control characters
showing/hiding file name extensions
ShowSounds
  defined
  setting timeout period
  turning on and off
shrinking screen fonts
shutdown
  restarting computers
shutting down a computer
  air travel and
  alarm for low or critical battery condition

To turn on ShowSounds

1. Open Accessibility Options.
2. On the **Sound** tab, under **ShowSounds**, select the **Use ShowSounds** check box.

   To turn off **ShowSounds**, clear the **Use ShowSounds** check box.

Notes

- To open Accessibility Options, click **Start**, point to **Settings**, click **Control Panel**, and then click **Accessibility Options**.
- ShowSounds instructs programs that convey information by sound to also provide information visually, for example, through text captions or informative icons.

Related Topics

Accessibility Wizard
Control Panel accessibility options
On-Screen Keyboard overview
Turn on SoundSentry

Figure 4. Windows help and support center application.

3.4 Guidance As Part of an Interface

As software has become more complex and the capabilities of computer hardware have increased the amount of screen real estate and output bandwidth available, it has become possible to include more user guidance in an interface.

When guidance is so tightly embedded in the interface that it becomes part of it, the interface can be categorized as an information-rich interface (Ames, 2001) and interface objects can become self-explanatory objects (Covi, 1995). Modern large computer screens with high resolutions make it possible to add guiding text to the application interface so that the interface explains itself. Another method that can be used to provide guidance in the interface is "tool tips," where a brief description of the underlying interface component appears next to the mouse cursor when it has remained in one place for a short while.

In spoken-language-dialogue systems, guidance is commonly part of the interface. Methods such as hints, implicit confirmations, and explicit prompts are used for this (Yankelovich, 1996). A user is guided at every point of the dialogue so that he/she knows what to say. However, extensive information cannot be provided, due to the slow output rate and sequential nature of speech. Conceptual-level information is very hard to provide, and even the basic guidance can quickly start to feel tedious as the user learns the system. Tapering off the prompts as the dialogue proceeds can assist with this issue in many cases, but inter-session adaptation is not always possible, making the repeated use of a system tedious.

Wizards are a form of guidance that helps users to accomplish tasks by guiding them through a task in simple steps. In graphical interfaces, wizards are particularly common in the application installation phase. The basic idea of a wizard could be applied also to speech-based interaction. However, it is very closely related to system-initiative dialogue with explicit prompts; therefore, the term "wizard" is not commonly used in the field.

3.5 INTELLIGENT HELP

Intelligent help is a broad concept that covers many different kinds of guidance. Intelligence, in its broad meaning, can be incorporated into guidance in many ways. Distinctions can be made between active and passive help (Fischer, Lemke & Schwab, 1985).

In passive help systems, where a user must be active to receive help, intelligence can adjust the content of the guidance to users' skill level and choose an appropriate type of guidance on the basis of, for example, the application context (Hiyoshi, Shimazu & Takashima, 1993). Another use of artificial intelligence technologies is in providing natural language support and in user modeling, as featured in UC, the Unix Consultant (Wilensky *et al.*, 1988).

One interesting use for intelligence is to provide guidance proactively when user behavior suggests that the user is not aware of something. This kind of guidance is called active help (Mallen, 1996). As is common with proactive features, active help is a good thing when it is provided correctly but considered very annoying when guidance is provided unnecessarily or incorrectly. Users are easily led to dislike intelligent systems when the intelligence fails, and intelligent help has been reported to lose credibility for even small deficiencies (Carroll & Aaronson, 1988). One, rather unsuccessful (Swartz, 2003), example of active help is the Microsoft Office Assistant, which offers guidance to users on various topics when it detects certain usage patterns.

An active help system needs to be either integrated with or separated but connected to the application, so that it can monitor user actions. Passive help systems, too, can benefit from tight integration with the host application. Help systems have been grouped into divorced, separated, and integrated (Knight, Kilis & Cheng, 1997) according to their relation to the application.

A computer system can potentially analyze the end result of the user's work on some application and provide constructive criticism. This is the idea behind critique systems (Fischer, Lemke & Mastaglio, 1990). They are aimed at helping users work more efficiently or achieve better results.

Naturally, this requires advanced analysis of the user's results and careful modeling of the application.

While intelligence is commonly considered helpful, it has been argued that the lack of robustness and complexity of design in such systems makes the development of more static manuals more effective, at least in the short term (Capobianco & Carbonell, 2003).

3.6 SOFTWARE TUTORING

Software tutoring is a form of user guidance in which an interactive software component tutors the user. It is active guidance for making sure that the user learns to use the system. A tutor can direct the interaction, which includes a user, a software application, and the tutor. Software tutoring is used most often in initial training.

Software tutoring is an interactive extension to the more widely available static tutorials. These can be printed or videos, and they show a user how to do some specific things with a software application. The idea is that the user copies the actions described in the tutorial and learns by actually performing the task. In the passive form of software tutoring, the user must follow the guidance exactly and make sure that the result is correct. Tutorials have many similarities to the minimal manual approach. Both are task-oriented, are modular, and contain procedural guidance. In practice, tutorials also require conciseness since users could not concentrate on the task otherwise. In some cases, minimal manual modules can be considered tutorials, and the guidelines for minimal manuals can be applied when tutorials are developed. The biggest difference may be in the user-initiated exploration. Minimal manuals aim at encouraging the user to explore, while tutorials often try to keep the user focused on the task at hand.

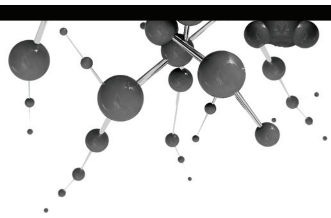
In software tutoring, the tutor is a software component. It can be intelligent, monitoring the user's actions and making sure the user does things in the correct way. A software tutor may also enable the user to work more freely, possibly working with real, as opposed to example tasks even during the tutorial period. This improves user motivation and can make the tutoring more relevant to the user's actual needs.

Software tutoring can be characterized by a set of features that differentiate it from other forms of guidance. One of these is that the tutor can give explicit instructions to the user and can have initiative in the interaction (Rich *et al.*, 2005). Another is that the tutor can monitor user actions and, for example, see whether the instructions were followed successfully (Contreras & Saiz, 1996). It is important that the teaching occur in the real context – i.e., with the actual application (García, 2000).

The guidance is also interactive and adaptive; when a user has problems, more detailed guidance is provided, and the operations that the user performs can affect the subjects in which the user will be tutored.

Examples of software tutoring and closely related software guidance systems include DiamondHelp, emphasizing a collaborative paradigm for interaction (Rich *et al.*, 2005), and TNT, which implemented multimodality using speech in tutoring for a visual interface (Nakatani *et al.*, 1986). The CACTUS system was designed for the automated generation of software tutors (García, 2000). Scenario Machine by Carroll and Kay (1985), an application of their "training wheels" approach (Carroll & Carrither, 1984), may also be considered a software tutor. It allows a user to go through a single scenario with an application so that at any given point only one transition is allowed. The integrated initial guidance of Kang, Plaisant, and Shneiderman (2003), on the other hand, approaches guidance by adding sticky notes to the interface. Kelleher and Pausch (2006) used stickies as well but added a stencil, a transparent layer containing guidance over a user interface. Holes in the guidance stencil allow user to interact with the software. Since only relevant parts of the interface are accessible at each step, training wheels type guidance results. Yet another tutor type approach is DocWizards by Bergman *et al.* (2005), which, in addition to textual guidance and highlighting of relevant GUI widgets, provide a possibility for the end users to modify the guidance recordings as necessary. A field where software tutoring has become the norm, albeit often in a rather simplistic form, is computer and video games. Tutorials are a recommended form of user guidance in this area (Sweetser & Wyeth, 2005; Desurvive, Caplan & Toth, 2004) since they enable people to start playing without reading manuals and the tutorial allows the user to feel as if he/she is already playing the game.

Intelligent tutoring systems research is relevant to software tutors because software tutors can be seen as a subset of intelligent tutoring systems. The use of domain and student models and an instructor module has been applied in software tutoring as well (Mallen, 1996). Very rarely is the actual tutorial subject present in the interaction with an intelligent tutoring system. However, this is the case in software tutoring. The student is using the actual software application during tutoring. This ensures that tutoring is successful and that the user can perform real tasks. There is no need to consider the differences between the model of the subject and the real thing as would be the case if the learning environment included a model of the application.



4 Iterative Development and Evaluation

4.1 Introduction

The research approach in this study is user-oriented, and the work concerns human–computer interaction. The design methodology applied and the evaluations are user-oriented; while a significant amount of software engineering work is reported upon, the technical solutions are always subject to the requirements of usability. This chapter examines user-centered issues relevant to the software tutoring of spoken-language-dialogue systems. Speech user interfaces and user guidance both have certain special methods and requirements for design as well as for evaluation. In addition to the methods applied in this work, the chapter aims at providing on overview of relevant work in the area.

4.2 User-Centered Development

In the 1970s, a new point of view arose in the field of software development. User-centered development raised the user to the center point of the development process. In user-centered processes, users are considered throughout the design and implementation. As processing power and the input and output capabilities of computers evolved, user interfaces could be better designed to accommodate users' needs and to support ways of working that were more efficient and easier to learn.

User-centered design and development has various measurable benefits, including decreased development, support and training costs, increased sales and user productivity, and decreased user errors (Mayhew & Mantei, 1994). Many of these benefits result from the main aim of the user centered development, improved usability of the product. Usability has arisen as one of the significant features of software applications. However, it is not trivial to design systems so that they are usable. The development of methodologies and processes has been one of the important results of user-centered design research.

On software development process level, iterative development has traditionally been fundamental in the user-centered approach (Gould & Lewis, 1985). Such methodology enables usability testing and many other methods of user-centered development. When prototype versions are tested with representative users, valuable information is gained on problems in the interface. While this approach is not common, it is recommended that user-centered methodology be used not only in design but also in the discovery and development phases of a project (Vredenburg *et al.*, 2002).

There is a large collection of user-centered design methods. Vredenburg *et al.* (2002) report the most common methods to be field studies (including contextual inquiry), user requirement analysis, iterative design, usability evaluation, task analysis, focus groups, formal heuristic evaluation, user interviews, prototyping without user testing, surveys, informal expert review, card sorting, and participatory design.

Methods of usability evaluation include observing users by means of, for example, think-aloud protocols. In addition, users' opinions are often gathered via questionnaires and semi-structured interviews. Expert reviews as well have many variations, including heuristic evaluations and cognitive walkthroughs, where experts evaluate software by means of their expertise, checklists, and users' point of view.

The development of speech-based user interfaces

In the development of speech user interfaces, the speech technology has a central role and must be evaluated carefully. In graphical user interfaces, where input and output techniques have matured and interaction is designed so that input does not require a statistical interpretation, nor contain ambiguity, such careful consideration of technology is not necessary. In speech user interfaces, variability in speech and audio signal transmission, and the ambiguity of language make the performance of speech technology a significant factor in interaction design. The selection and evaluation of the technology includes general quantitative measurements of technology components. Some of the metrics employed were reviewed in Chapter 2.

Since the technology presents limitations to interaction, knowing these limitations is very important, so that they can be taken into account in the design of the interaction. Appropriate configuration of technology components can be vital also. In some cases, development of the technology is possible also; for example, speech synthesis may be adapted to a specific application by recording suitable material for a unit selection synthesizer.

Since human–human interaction is the most prominent form of speech communication, it is often considered the starting point for human–computer dialogue design. However, users may not use similar language when communicating with a machine as when they communicate with people (Brennan, 1991; Guindon, 1988), and therefore human-human interaction should not be applied to speech interface design without qualification. When services for which a personal telephone-based service is available are automated, corpora are often collected from the existing services. Such data can help in understanding the terminology, specifying the functionality required, and performing language modeling for speech recognition and system outputs. What the corpora may not provide is a suitable model for the interaction. Technology limits the interaction possibilities to such an extent that the interaction methods employed in human–human services do not work. This is most prominent in error situations, when people tend to simplify their language to such a style as would never be used in human–human dialogue. Jönsson and Dahlbäck (2000) have presented one methodology to apply human-human dialogue corpus to speech interface design. Dialogues can be re-written, or distilled, to make them more like potential human-computer interaction. The process includes modifying the "system" turns so that Grice's maxim of quantity is strictly followed. Dialogue contributions, which cannot be considered system or user turns, are also removed. Unnecessary content, such as repetitions, is removed, as well as system turns, which would not take place when a computer is a dialogue partner. The authors report that in some respects such distilled dialogues can be more realistic than Wizard-of-Oz based material.

In addition to personal services, computerized systems can be used as a starting point when speech-based systems are developed. For example, interactive voice response systems where telephone key presses are used as input are often replaced with speech-recognition-based systems. Web-based interfaces to services are also commonly available. Such systems provide a reasonable base for development since the functionality has already been systematized, but the challenging nature of speech recognition as an input modality leaves a lot of work for the developers.

A common methodology in the development of speech user interfaces is "Wizard-of-Oz" (WOz) studies. In a WOz setup, all or parts of a system are simulated by a human. The term "system-in-a-loop" is used to describe systems where only part of the system is replaced by a human operator. In most cases, at least speech recognition is simulated. When users believe that they are talking to a machine, we can collect realistic data. However, there are challenges in the usage of WOz methodology. The simulation of speech recognition errors is especially challenging. One approach to tackle this challenge was presented by Skantze (2003), who used an unusual WOz setup, where a speech recognizer was used and a wizard acted based on the speech recognition result. The methodology is also time-consuming, it has extensive implementation costs (unless appropriate tools are available), and the task of the wizard is demanding and requires training. The possibilities with WOz methodology range from very realistic setups where users believe they are using a real system to testing fast mockups implemented with tools such as Suede (Klemmer *et al.*, 2000), where users are perfectly aware of what is happening.

Iterative development is very important in speech user interface design. User tests provide feedback on user interface issues such as the design of system prompts. In addition, the speech technology can be evaluated at the same time. This can be problematic since significant problems with the technology can obscure issues in the interface design. During the iterative development, improvement to technology components can also be included. Another common method in usability engineering, heuristic evaluation, requires either checklists of usability issues or advanced expertise. Currently, checklists for speech user interfaces are not widely available, but progress is being made (Suhm, 2003b).

The development of user guidance materials

User-centered development of documentation, especially online documentation, is similar to that of interactive systems in general. Methods such as user tests with think-aloud protocols, video recordings, log analyses, and interviews are usefully applicable, as are user surveys and beta testing. Even Wizard-of-Oz techniques can be applied when advanced interactive guidance is being developed. (Mehlenbacher, 1993)

While online help can be viewed as part of the application, its development process can be considered a separate one. The developers are also often a different group of people. Patrick and McGurgan (1993) present a methodology for online help development. This methodology involves a plan and a process. The plan includes a project overview and the business objectives of the help; lists the components of the documentation set; and profiles users, the user environment and user tasks, and ways to access the help. A detailed outline of the help content is also constructed, and screen format and guidelines for the use of graphics are determined. The maintenance part of the documentation life cycle is specified. The process includes familiar usability tools: prototyping and usability testing. Together with editing and development of usability test plans, these form an iterative cycle at the center of the document development process. Later parts of the process include, for example, integration testing with the application.

When minimal manuals are written, their task-oriented nature requires identification and analysis of relevant tasks. A GOMS model has been utilized in the documentation design since it enables designers to focus on real user tasks, can help reduce the size of the documentation, and can also be used in the error recovery analysis (Gong & Elkerton, 1990).

Another method for developing guidance is to ask people to review the guidance and report their attitudes. Palmiter and Elkerton (1991) asked people to review textual and demonstration-based guidance on the same subject and to note any dissimilarities.

4.3 THE EVALUATION OF SPEECH USER INTERFACES

Various methods have been developed to evaluate the usability of interactive systems. These include testing methods where representative members of the user population use the evaluated software. In addition to monitoring users' actions, these tests commonly include inquiry of users' opinions. Usability can also be evaluated by experts using various walkthrough methods and checklist based approaches. In addition to the generally applicable methods, dedicated and adapted practices exist to evaluate both speech user interfaces and guidance materials.

In speech technology, evaluation of technical components has been far more extensive than usability evaluation has. Larsen (2003a) claims that "usability of voice driven services is still poorly understood." The evaluation of speech user interfaces differs from that of graphical interfaces. Perhaps most important is the transient nature of speech, which affects many important elements, such as transparency, learnability, error handling, and user control (Larsen, 2003b). It is also worth noticing that the use of think-aloud protocol is usually troublesome and emphasis must be made on the requirement that test tasks do not put words in the user's mouth.

Usability metrics are commonly divided into objective and subjective metrics. The former measure users' effectiveness with the system, while subjective metrics collect users' opinions on the system. Objective metrics can be divided further into quantitative and qualitative measures, by whether the results are "independently meaningful numbers" or are instead expert judgments or other estimates (Bernsen & Dybkjær, 2000).

In the evaluation of spoken-language-dialogue systems, the most widely used objective metrics include the percentages of correct system answers, successful transactions, repair utterances, sentences containing more than one word, and user-initiated turns; the number of "help" requests and barge-ins, completed tasks, and sub-tasks; dialogue and task completion times, mean user and system response times, and the mean length of utterances (Larsen, 2003b). Surveying the literature, Möller (2005) reports 36 different objective metrics, which he divides into five categories: dialogue- and communication-related, meta-communication-related, cooperativeness-related, task-related, and speech-input-related. Many of these metrics measure the entire system or interaction.

When individual components are evaluated, component-specific metrics are used. For speech technology components, the metrics are often straightforward to develop since the components can be tested in isolation. Measuring the components responsible for interaction management is more challenging due to intercomponent interactions. Danieli and Gerbino (1995) present metrics to evaluate dialogue management specifically. Their metrics are transaction success, the calculation of normal and correction turns, and implicit recovery. Implicit recovery is the measure of how well a dialogue manager is capable of handling situations where speech recognition or language understanding has failed. In addition, Danieli and Gerbino use a metric called contextual appropriateness, referring to the number of system utterances that are appropriate in the current dialogue situation. Appropriateness is related to Grice's maxims; for example, when a system presents incorrect information or too much information, it does not meet contextual appropriateness requirements.

Subjective measurements are usually collected using questionnaires. The development of a valid and reliable questionnaire is not a trivial task. There are questionnaires for evaluation of user interfaces, but these have been developed with graphical user interfaces in mind, making their applicability for speech-based user interfaces questionable. Larsen (2003b) identifies only two questionnaires that have been developed for speech-based user interfaces and systematically address validity and reliability. These are BT-CCIR and SASSI. He argues that improvement in understanding of the subjective measurement of speech-based user interfaces is vital for their development (Larsen, 2003a).

In the evaluations included in this thesis, the set of questions from SASSI has been used to collect participants' opinions on the features of spoken-language-dialogue systems. Subjective Assessment of Speech System Interfaces (SASSI) is a questionnaire that has been developed by testing a pool of 50 questions. Analysis based on 214 completed questionnaires resulted in a set of 34 questions, which were grouped by six user perception factors. Hone and Graham (2000) call these system response accuracy, likeability, cognitive demand, annoyance, habitability, and speed. While the development of the questionnaire is not considered finished, Hone and Graham consider SASSI to "have face validity, and a reasonable level of statistical reliability."

PARADISE (Walker *et al.*, 1997) provides an entire framework for conducting studies of spoken-language-dialogue applications. It separates task requirements from dialogue behaviors and thus allows comparison of different applications. This is enabled by attribute value matrices (AVM) where tasks are modeled as the information exchanged between dialogue agents. Task success is measured with a normalized metric, kappa coefficient, instead of plain task success rate. Kappa incorporates the possibility of receiving correct results by chance (cf. Cohen's Kappa in statistics). Together, kappa and AVM enable comparison among dialogue strategies and the calculation of performance for both whole dialogues and sub-dialogues.

To apply PARADISE, tasks must be modeled with AVM. If open tasks – i.e., tasks that users define themselves (Walker, Hirschman & Aberdeen, 2000) are used, AVMs are defined after the experiments. Kappa values for task success are calculated based on test results. During the experiments, subjective metrics are collected. Objective metrics are usually collected afterward from recordings and log files. Final part of PARADISE procedure is to calculated regression models to relate users' subjective measurements of satisfaction, which are taken to model usability, and objective evaluations (task success and costs), for calculating how much each item of objective measure of system performance affects user satisfaction as seen in Figure 5. This enables the optimization of systems. PARADISE models have been found to generalize across systems, but not all user groups fit the same models (Walker, Kamm & Boland, 2000).

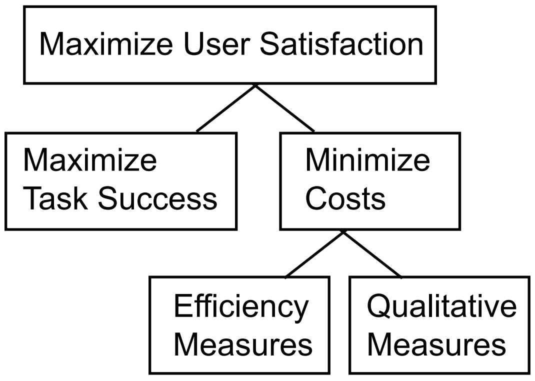


Figure 5. The structure of objectives in PARADISE (after Walker *et al.*, 1997).

Both objective metrics and subjective evaluations are left somewhat open in PARADISE discussions. Subjective metrics to evaluate user satisfaction usually consist of a short questionnaire with Likert-scale answers; user satisfaction is a sum or average (Smeele & Waals, 2003) of the numeric codes of the answers. One may question whether a single number derived in such a manner is indeed a relevant measure of user satisfaction, let alone usability. Smeele and Waals (2003) add a general "Grade" rating to the questionnaire used by Walker *et al*. and obtain regression models with better prediction power. Möller (2002), on the other hand, states that a simple arithmetic mean cannot be used as a measure of user satisfaction and questions whether the division into efficiency measures and qualitative measures is fine-grained enough. Objective metrics are also open to discussion. For example, Walker, Passonneau & Boland (2001) included dialogue act tagging in addition to previously used metrics in the analysis phase, which increased the fit of the regression model by an absolute 5%.

One issue, which has only recently been raised in the field of speech user interface evaluation, is what Hirschman and Thompson (1996) call "adequacy evaluation": the evaluation of the level of performance required of each component in a system. For example, as discussed, mere recognition rate is not an important figure unless we know, which is the acceptable level of recognition rate. Larsen (2003b) relates this to "utility" in Nielsen's definition of usability (Nielsen, 1993). This issue has been addressed in the SERVQUAL methodology adapted to speech user interfaces by Hartikainen, Salonen, and Turunen (2004). In SERVQUAL, participants not only evaluate systems from different points of view but also provide their opinion as to the acceptable and desired level for each item.

Another issue is the difference between the task completion rates as perceived by a user and as observed by the test conductors. These two judgments have been found to differ (Walker *et al*., 1998), and some researchers (Möller, 2005) use both. In some cases, user perception of task success has been used instead of kappa coefficient in PARADISE applications (Walker, Kamm & Boland, 2000). This is interesting since in this case users' subjective evaluations can be found from both sides of the regression model.

Yet another issue related to the evaluation of speech user interfaces is the role of the systems. In contrast to the traditional HCI view, where speech systems are seen as applications, many spoken-language-dialogue systems can be approached as services. This has been addressed by the application of the aforementioned SERVQUAL method. Möller and Skowronek (2003), too, take this point of view. They use the quality-of-service taxonomy as the basis for their evaluation, specifically in the design of a questionnaire used to collect subjective evaluations.

4.4 THE EVALUATION OF USER GUIDANCE

The task of guidance material is to help users learn to use a software application. Some forms of guidance can also support users while they are working. While this is a reasonably well-defined task, measuring its success can be a challenge.

The readability of the guidance text is prerequisite for successful guidance. There exist various readability metrics, such as the Flesch-Kincaid readability test, which can automatically warn writers of too complex language. These calculate scores mostly from word and sentence length. Empirical methodologies for testing documentation include asking users to search for specific information in the document and paraphrase or summarize excerpts (Mehlenbacher, 1993). These methods provide views on the readability of the user guidance. However, the real context (i.e., the software application) is ignored in these tests, and the actual learning results may or may not correlate with such results.

Test setups with more ecological validity require users to either work freely or accomplish specific tasks with an application while, or after, using the relevant guidance in learning to use the application. In these evaluations, various metrics can be used.

Users' actions with the guidance and the application can be analyzed. Giving users tasks to work with enables more detailed analysis and the comparison of different guidance materials. Measurements collected from these analyses include the time to complete training and tasks, the number of errors committed and corrected, task success (Borenstein, 1986; Gong & Elkerton, 1990; Lazonder & van der Meij, 1995; Black, Carroll & McGuigan, 1987; Harrison, 1995), the number of successful searches, how many times the answer was found via the optimal path or random search, the number of questions asked (Maes, Goutier & van der Linden, 1992), the time spent on error recovery, the number of experimenter interventions, and reading times (Carroll & Kay, 1985). Analysis of users' interaction can yield more qualitative results also, guiding the design. For example, Maes, Goutier, and van der Linden (1992) report on the types of help searches, and van der Meij (1992) studied errors by dividing them into selection rule errors, method errors, and operator errors, then considers the number of successful error recoveries and the use of error-recovery strategies. It is also possible to measure users' learning by conducting a separate test with questions about the material taught and rating the users' answers (Lazonder & van der Meij, 1995; Carroll & Aaronson, 1988). However, the approach is used relatively rarely.

Questionnaires, interviews, and surveys can be used to gathers users' opinions on the guidance they have received (Maes, Goutier & van der Linden, 1992; Palmiter & Elkerton, 1991; Harrison, 1995) and to collect more general views and attitudes towards user guidance (Purchase & Worrill, 2002).

Since guidance cannot separately teach each and every task possible with software applications, guidance should be able to teach skills that the user can transfer to other, more or less similar tasks. When the guidance situation does not include the actual software application, there is also need to transfer the skills from the training to the actual application. When the skill transfer is a straightforward application of the knowledge learned, it is considered "near transfer." When the skills are applied in a new situation or environment, "far transfer" occurs (Perkins & Salomon, 1992). These two categories have been examined in documentation studies, commonly by providing users with tasks that require either near or far transfer and comparing the results of the two groups to see what kind of learning has taken place (Carroll & Kay, 1985; Wiedenbeck, Zila & McConnell, 1995; Wiedenbeck & Zila, 1997).

In the experiments, where the software tutoring presented in the following chapters was evaluated, both subjective and objective measures were utilized. To collect participants' opinions on featured spoken-language-dialogue systems, questionnaires based on SASSI were used. The SASSI questionnaire could not be used to gather opinions on guidance materials due to its focus on spoken interaction. A questionnaire based on work of Hassenzahl *et al.* (2000) was devised for this purpose. Objective measures were collected from audio recordings made during the experiments. Metrics, such as error rates grouped by error types, were analyzed and they provided insights into participants' learning experiences. Task success rates were also calculated but they did not show significant differences between different guidance materials.

5 Software Tutoring in Spoken-Language-Dialogue Systems

5.1 Introduction

In our consideration of software tutoring in speech user interfaces, we now turn to examining software tutoring in more detail and, in particular, its application to teaching the usage of spoken-language-dialogue systems. This chapter addresses the issues relevant to the design and implementation of the software tutors described in the publications. Software tutoring is defined and requirements on what to teach are discussed. Use of modalities and technological issues are also considered.

The motivation for the study of software tutoring in speech user interfaces comes from two sources. The main motivation is that tutoring provides one approach to solving the challenge of user guidance in spoken-language-dialogue systems. It can be one way to widen the range of accepted speech-based solutions since it can be used to introduce spoken-language-dialogue systems to new users.

Tutoring is interesting also from the interaction point of view; it results in three-party dialogues. It is not clear how such dialogues should be designed and modeled such that users can cope with them. The separate character of tutoring is also related to the idea that multiple services could be presented to a user more consistently by means of a single agent supporting user–service communication. It is also possible to avoid the addition of an actual dialogue partner and create a tutorial kind of "guided mode" (Karsenty & Botherel, 2005) where guidance is part of the system prompts.

Potential and known benefits

While simple spoken-language-dialogue systems can work using just embedded assistance, systems that are more complex need some sort of user guidance to explain the limitations and capabilities of the application. Even static tutorials have been found to have positive effect on users' perceptions of a spoken-language-dialogue system (Kamm, Litman & Walker, 1998). Software tutoring can be effective in providing users with the fundamental concepts of the application, and users completely new to speech user interfaces seem to benefit from tutors. The tutor can take care of these users and make sure that they learn to actually use the system. Without such guidance, users may, for example, speak too quietly or at a wrong time and never successfully use the system in question or any similar one. In general, tutoring can simplify learning. People who are not too confident with technology are often afraid of new systems. Tutoring that explicitly tells the user what to say in the early part of the interaction can help such users. When the user can just pick up a phone to call the system and then receive the necessary tutoring on the first call, the threshold for starting to use the system can be lowered.

Risks and limitations

Naturally, tutoring does not suit everybody. Users who are already familiar with similar systems and are confident with them can easily feel offended if they are required to follow controlling guidance. They are better served with reference-manual-type documentation or quick-start leaflets. Tutoring can easily become irritating and tedious for beginners as well; badly designed tutors can cause frustration, even confusion.

An even worse risk is the tutor containing errors. When a tutor crashes, ends up in an endless loop, or something similar, this, too, can turn users away from the system. This is a relevant consideration since errors are easy to introduce into software systems. Traditional guidance methods, such as printed manuals, are far less prone to such technical problems. The more complex a tutor is, the more likely it is to contain programming and design errors.

5.2 WHAT TO TEACH TO USERS

Different users have different needs for guidance. Users' prior knowledge, skills, and capabilities greatly affect their needs. Weegels (2000) identifies three possible sources of prior information: knowledge of human–human interaction, of the application domain, and of similar services. Furthermore, experience, existing and forthcoming, with an application can be used as a basis for grouping users. The three groups are experts, those who will use the system only once, and those who use the system for the first of many times (Resnick & Virzi, 1995). In addition to skill level with the system and domain-specific knowledge, one may consider factors such as users' hastiness (Komatami *et al.*, 2003). The aim of the guidance may not be to teach the user everything about the system but to provide the necessary amount of information. Fischer, Lemke and Schwab (1985) have identified three levels of system knowledge that users have, as shown in Figure 6: concepts a user knows and uses without problems; concepts used only occasionally; and the user's mental model of the system, referring to the set of concepts that the user thinks are part of the system. In many cases, none of these corresponds to the actual functionality of the system.

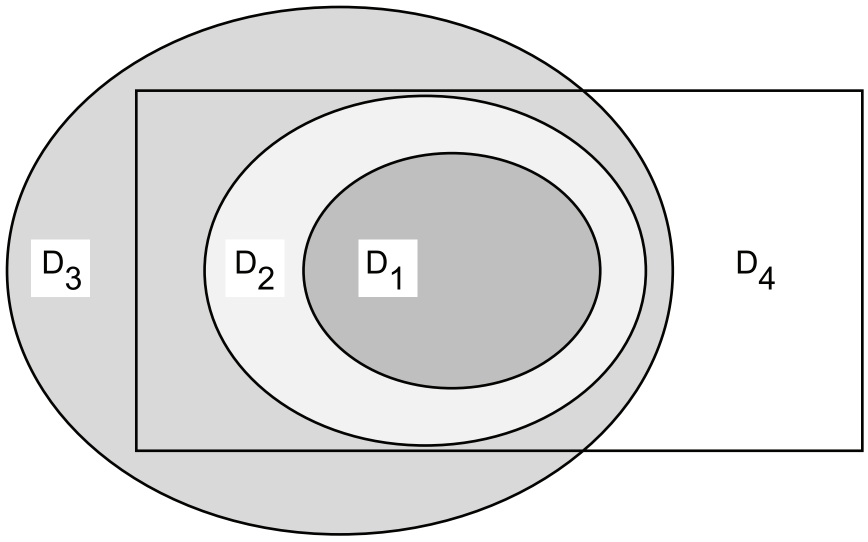


Figure 6. The levels of system knowledge: D_1 = active concepts, D_2 = occasional concepts, D_3 = mental model, D_4 = system's actual functionality (after Fischer, Lemke & Schwab, 1985).

Users new to speech user interfaces

The most difficult thing for new users is to get started (Houghton, 1984). While users do not know how speech user interfaces work, they have their prior knowledge of other things, something that is very important to understand (Carroll & Rosson, 1987). One important source of prior knowledge related to speech user interfaces is, obviously, human–human interaction. This is where linguistic capabilities come from, and this is perfectly applicable information. As has been discussed in Chapter 2, there are differences between human−computer and human–human spoken dialogue, most importantly in turn-taking behavior, error correction responsibilities, and language understanding.

Another source of information is the user's general knowledge of computers. In the industrialized countries, most people have used computers at least occasionally and have some ideas about how computers work. For example, 63% of adult UE25 citizens have basic computer skills (Eurostat, 2006). People commonly assume that computers do not really understand natural language but work in a more mechanical manner. Because of this, people may underestimate the capabilities of spoken-language-dialogue systems and use overly simplistic language. On the other hand, this may also correctly direct their expectations to match the actual, limited communication skills of today's systems.

Finally, people may have some ideas about how speech recognition systems should function that are drawn from works of fiction (McTear, 2004, p. 41). This may not be irrelevant information since fiction has provided inspiration to scientists in the field as well (Lai, 2001).

These varying sources of information and expectations make it necessary to teach new users in some way about the capabilities and limitations of human–machine spoken dialogue. The users need to understand the extent of language understanding: the scope of the supported vocabulary and what kinds of expressions the system can understand. The users must be informed also about the turn-taking possibilities, such as any lack of barge-in capability. Some general idea of the capabilities of dialogue management may be useful as well. Speaking style is also often significant, to support speech recognition and even to address turn-taking. A thorough understanding of these issues is not necessary for users, and the necessary knowledge can usually be communicated with a single good example.

The error-prone nature of speech recognition is something that users need to understand, and it is necessary to know the methods of error correction if one is to maintain successful dialogue. Including ample error-related information in tutorials has been shown to have significant positive impact (Lazonder & van der Meij, 1995). Because speech recognition errors are system errors, as compared to user errors, provision of error information in speech user interface tutorials is especially vital. If users cannot distinguish between system and user errors, they may unnecessarily alter their actions and thus create new problems.

Users new to the application

Users who have used speech-based applications before are likely to know the main differences between human–human and human–machine dialogue. Of course, the limitations of machines vary from system to system, so users often need to adjust what they already know when they learn a new system. If there are differences between user assumptions based on the other application and the new application's actual functionality, these may hinder learning. A tutor can make sure that the interaction is successful and that users know the new interaction style.

In most cases, users also need to learn the words that are used in a specific application. The dialogue structure as well varies with application domain. The approach of universal speech user interfaces aims at maximizing the skill transfer between speech-based systems (Rosenfeld, Olsen & Rudnicky, 2001). However, even in this approach, users need to learn the functional scope of the new system. If a tutorial is well designed, a competent user can go through it quickly and learn the system while users who need more help receive all the guidance they need.

5.3 The Defining Features of Software Tutoring

Software tutoring shares many features with other forms of user guidance and can be implemented in many ways. Tutorials consisting of text alone share common issues with text documents, while the use of graphics relates tutor design to multimedia design. However, there are defining features that, taken together, distinguish software tutoring from other forms of user guidance. In this study, the following set of features has been considered to form the definition of tutoring and they can be found from the tutors presented in the publications. Detailed examples of the tutors in operation will be presented in Sections 6.4 and 6.5.

Teaching in the real context

As we have noted, the fundamental idea in software tutoring is that the guidance works together with the actual software application the users are learning to use. The tutoring can be an integral part of the application or a separate system, somehow connected to the tutored application.

Through teaching usage with the actual system, there is no need to build a separate learning environment. The latter would not only be laborious but also introduce the risk of the learning environment not matching the actual application (García, 2000).

When guidance is designed carefully, users can work with their real tasks already during the tutorial. For example, when the usage of an e-mail-reading application is being taught by the tutor described in publications II–IV, users can read messages in their own mailboxes while learning the system. This not only increases users' motivation but also makes sure that teaching of the context, as far as it is dependent on users' own data, is realistic. In the e-mail example, users who have hundreds of messages in their inbox will learn to use the system with such an inbox while those who receive little mail and carefully organize their messages can learn to work with a small inbox. One challenge when tutoring takes place in the real application environment is that the need to switch from learning mode to problem-solving mode may decrease users' performance (Houghton, 1984).

Explicit instructions

Tutorials differ significantly from other forms of user guidance in that explicit instructions are given to users. While intelligent assistance leaves the initiative to the user, a tutorial has most of the knowledge and initiative (Rich *et al.*, 2005). This applies to printed tutorial materials as well as to interactive software tutoring. Tutors give a guided tour of the system and request the users to perform tasks.

The explicit instructions have several benefits. When given explicit instructions, users – especially unconfident ones – feel more at ease in learning a system. The guided tour that is possible this way can be structured so that a sensible subset of system functionality is covered and the functionality is presented in the context of some real tasks.

Most importantly, when explicit instructions are combined with software tutors' ability to monitor user actions, it becomes easier to see whether the user is successful. For example, critique systems need to be more intelligent since they need to deduce what the user is trying to do. In speech user interfaces, the explicit instructions make it possible to see whether the user is capable of uttering speech inputs successfully. When a tutor asks explicitly that the user give a specific input, it is easy to see whether the user was successful. Given the probabilistic nature of speech recognition, spotting problems with sufficient accuracy would be much harder, if not impossible, if we did not know what input the user was trying to give and when.

Monitoring of users

When following passive material, users must, by comparing the result against the guidance material, make sure they are doing the right thing. Interactive tutors can monitor users and provide feedback when noticing errors in user actions. The feedback can improve the learning since positive feedback supports learning and negative feedback corrects users' behavior in the right direction before they learn incorrect ways of working.

Monitoring users' actions goes hand in hand with giving explicit instructions. However, monitoring users can provide other benefits than just spotting problems. Guidance can be given in the appropriate context when a tutor knows what a user is doing. While out-of-context guidance can be effective, users do not favor it (Mackay, 2001).

An active tutor can also make sure that users learn what they have, in theory, been taught. Intelligent tutoring systems emphasize this very much. Dialogues in many of these systems consist of questions to users to find out what they know. The systems keep models of what the user knows and what still needs to be taught. The software tutor may also ask explicit questions from users, but a more natural approach is to ask a user to do something with the system, then monitor what happens. This not only removes the need to implement a separate questioning module for the tutor and the need for the user to learn about it but also makes sure the user is capable of actually performing the task, not just knowing it in theory.

With tutoring in complex systems, especially ones that include potentially destructive operations, the tutor designer may want to limit users' options. This "training wheels" approach requires that the tutor have some way to either modify the system state or preprocess users' actions before the application is allowed to process them. Software tutoring requires some way to monitor users' actions with the system and, ideally, also monitor the system state and control it. How easy it is to gain such access depends on the environment in which the system has been implemented, and this is one important consideration in the development of software tutoring.

Adaptive guidance

The knowledge of users' actions makes it possible to adapt guidance to individual users. In paper, video, and other non-interactive tutorials, users are usually informed how to complete a predefined example task that in many cases has little to do with the user's real needs. Interactive guidance can potentially let users do their own tasks during the tutoring. This way, the tutorial can include working with actual tasks. This is likely to not only increase motivation but also help the guidance to be more relevant to the user's needs.

Monitoring the user's actions can suggest what should be taught to the user and what the user has learned successfully. In complex applications, knowledge of the user's actions can also help to provide guidance on the specific features the user is likely to need and to ignore those that are not relevant to the user's tasks. This can make guidance more efficient, since only relevant subjects need to be taught. Also, users with varying skill levels can be supported. This is perhaps the most important form of tutor adaptation. When problems are detected, more guidance is provided. Users who do not need so much guidance can proceed more quickly, and those who need help receive it automatically.

Timely synchronized guidance

Since tutoring is attached to the application the user is learning, the guidance can be synchronized with events in the application. For example, in the tutor for the e-mail reading application, guidance on navigation inside e-mail messages is given when a user has chosen to read a message. The guidance can be interleaved appropriately with system usage in this manner.

When a modality other than that of the application is used, the system events and their commentary can be simultaneous. For example, Nakatani *et al.* (1986) used speech to tutor users in the usage of vi, a visual text editor. They noted that the possibility of providing guidance simultaneously is one of the strengths of speech in their domain. Similarly, when speech user interfaces are taught, simultaneous commentary may be accomplished in some cases with graphical tutoring. However, results reported in publication VI suggest that great care must be taken not to overwhelm users with too much simultaneous information.

Structured guidance

Tutors are active guidance components and therefore can enforce following of a certain structure in the guidance material. Naturally, the users should retain the highest level of control, so that they can, e.g., quit the tutoring any time they want and skip parts that they find unnecessary. The benefit of the tutor being in general control is that it can provide the material in such an order that it is easy to understand. When users are studying, e.g., online manuals, they may read fairly unrelated pieces of documentation and not receive a decent overall view of the system. In the multimodal tutors presented in publication V through VII, the linear structure of the guidance is enforced by using wizard-like continue and back buttons, so that users can control the pace but not the structure of the guidance.

Tutorials, including video and printed ones, are built on the idea of providing users with consistent, story-like coverage of certain system functionality. Software tutoring can work in a somewhat similar manner while allowing the users more freedom. When new users are tutored, they can be guided very strictly through the most important functionality so that they know the necessary things. Later, the tutoring can move into the background and provide assistance only when it seems necessary.

5.4 Modalities in Tutoring

The modalities used in software tutoring are the foundation for the entire interaction design of the tutor. The strengths and weaknesses of the different modalities and the technical requirements and available support must be considered. The application/tutor interrelation offers a new angle for multimodality considerations. By definition, in software tutoring there are two components: the system the user is learning and the tutor. These two components can use the same modality or modalities, or they can be split into different modalities if reasonable ones remain beyond those already occupied by the system. Using the same modalities keeps the tutor close to the system. Examples can be given in the modality applied in real use, for consistency and ease of transfer. The technical feasibility of the modality is also guaranteed. A problem in this can be overloading of the single modality. Using different modalities avoids the overload and provides possibilities for simultaneous system and tutor activity.

Speech in teaching of speech user interfaces

Speech has been recommended for guidance in graphical interfaces (Capobianco & Carbonell, 2003; Nakatani *et al.*, 1986). On the other hand, users have been reported to consider speech an annoying, slow, and irrelevant detail when it is part of a graphical help facility (Purchase & Worrill, 2002). Harrison (1995) also reports that, while some initial learning is more effective with speech, users prefer written instructions. When the usage of a speech user interface is taught, using speech may pose some additional usability issues. The limitations of speech-based tutoring come from the nature of speech and the increased amount of information provided to the user using speech. Speech is a slow output channel. It is challenging enough to design a speech user interface itself for efficient interaction that the interaction may become tedious when tutoring is added. Providing some respite is that the context of usage in tutoring – a situation where the user does not yet know how to use the system – makes the user less demanding. Another factor to consider is that the temporal and transient nature of the medium can be problematic. Explicit support, such as a "repeat" command, must be provided in case users need to hear guidance again. A certain amount of repetition, such as summaries, must also be provided.

The usage of speech for tutoring in a speech user interface is technically viable. The required speech technology is already available for the application taught. Given flexible system architecture, a tutor can built on the existing resources. Limitations may arise from the fact that the speech synthesizer used may support only one voice and a different voice may be required if the tutor is to be distinguished from the application. Monolithic systems may also make it impossible for a tutor to use the system components. Speech recognition, if used as an input medium for tutoring as well, may require work on account of some need to switch between or adjust speech recognition grammars.

Other modalities may not be available when tutoring in the speech user interface occurs. If the user group is, e.g., visually impaired users, or the use and learning context is mobile, graphics may not be readily available or in any way useful. Therefore, using speech is a safe solution in the sense that it is accessible to all users in all potential usage situations. However, it can be questioned if the use of a tutorial is a smart choice when driving a car and in other safety critical situations.

Using the same modality as the system takes care of the compatibility of tutoring and real use, at least to some extent. When the examples of how to speak are given, they include also prosodic information. This is hard to communicate with text. The examples of usage can also be given so that they are perfectly accurate. Bad examples can be harmful. If the speaking style of a tutor is not compatible with the requirements of the system, the examples can be misleading – for example, leading users to speak too quietly.

Multimodality

Multimodality, in the form of providing tutoring in a modality different from that used by the system, solves some of the problems of speech-based tutoring but has its own limitations. The most important is perhaps that the accessibility of tutoring is likely to be reduced. The use of graphics excludes at least most visually impaired users. Graphics are a problem also for those cases where speech is used because of its "eyes-free" nature. However, to use the example of cars, a multimodal interface can still teach the speech commands – users can learn the system when the car is stationary (Pieraccini *et al.*, 2004). With a multimodal approach, many benefits of speech are lost and some actually become disadvantages. Different modality makes usage examples more abstract, and such things as prosody, rhythm, and timing are lost.

The gains come in the expanded communication channel. It can reduce the effort necessary to switch between the application context and guidance (Capobianco & Carbonell, 2003). Also, more information can be provided to users. This is especially true since most people can read at a faster rate than that at which they can listen. Also, the separate channel adds the option of simultaneous tutoring and system usage. While not everything is possible – users cannot concentrate on multiple things at the same time – carefully designed concurrency is effective. For example, spoken dialogue can be visualized with text in real time. The requirement is that the visualization of spoken dialogue be presented in timely synchrony.

The visualization of a system, beyond visualizing the spoken interaction, must be presented when the system is paused and a user can concentrate on studying the visuals. There should be no need for a user to concentrate on two things at once. Visualization must also be considered carefully, for users tend not to be interested in any technical information. Visualizations of dialogue structure (Boda, 2000) and ability to modify the system

(Gustafson *et al.*, 1996) may be useful to developers and students of speech technology. However, as reported in publication VI, most users participating in our experiments with such guidance said they do not care for information that is not directly and inevitably relevant to what they need to learn. Such information was, at best, ignored by the participants. Animation can be an efficient part of visualization, especially since spoken interaction is temporal in nature. However, there may not always be benefits in the use of a dynamic visualization over a static display (Harrison, 1995).

Other graphical methods include the use of graphical user interfaces. These can be used not only to control the tutoring but possibly to control the speech-based application. One of the main benefits of graphical user interfaces is that they show users the limitations of the interface. Unlike with speech, and command-line interfaces, there are no out-of-vocabulary types of problems. Therefore, providing a GUI corresponding to the features supported by the speech-based application, we can create a sort of visualization of the speech-based system's functionality. The benefits of this approach depend on users' familiarity with graphical interfaces. Skill transfer from the graphical to speech-based user interface seems to be possible (Terken & te Riele, 2001). The results presented in the publications VI and VII support this. As long as users spend time on using a graphical interface functionally similar to a speech interface, they report on learning about the speech interface from it and rate such a tutorial positively.

5.5 Technical Requirements

Technically, it is not a trivial task to implement a software tutor. By definition, the tutor must be connected to the system in which tutoring is to occur. Depending on the system, this can be reasonably easy or practically impossible. The requirements for the integration stem from what has been discussed in Section 5.3. Other than in the integration, the tutor development is mostly standard software development. Intelligence, to the extent that it is included in tutoring, may need some application of artificial intelligence techniques.

The integration effort required is largely dependent on the system into which the tutoring is to be integrated. Tutoring is sometimes developed at the same time as the application, allowing its needs to be taken into account at development time (Sliski *et al.*, 2001). This is perhaps the most favorable situation. When a tutorial is built for an existing system, some level of instrumentation to the system is required so that its state can be accessed by the tutor.

The instrumentation should enable the tutor to read and modify the system state. To be able to adjust tutoring to users' interaction with the system, the tutor should be able to monitor the system state. The model of the system's state depends largely on the system and on the needs of tutoring. Some indications of user inputs are required, and some information on the domain data with which the user is working is very useful as well. The possibility of controlling the system can be very important also. In speech-based interaction, whose temporal aspect is key, it is highly important that the tutor be able to at least halt the system temporarily. The possibility of discarding the user's input makes it possible to build systems with a "training wheels" approach and, in some cases, can also make the tutoring more fluent. If user input can be modified or new input created, more possibilities are given to the tutor. It can, for example, use the system to show users how to do something.

The challenge of the instrumentation varies. If a tutor developer has access to the system's source code and may modify it, instrumentation should be possible. In most cases, the system architecture dictates how much work is needed. It is preferable that the source code of the system need not be modified. The addition of some components to the system to enable tutor integration is more acceptable, especially if the underlying architecture is modular. For example, in graphical interfaces the message-based architecture can be utilized in the integration of guidance systems (Sliski *et al.*, 2001). In most cases, providing tutoring in closed-source systems is largely impossible. However, it may be possible with the use of some type of inter-process communication or by encapsulating the system within the tutor so that all input and output goes through the tutor. If a speech-based system is taught using speech, integration of the speech inputs and outputs of the system and the tutor may also be problematic. The technical architecture the system is running on must be open enough to support such integration.

In Jaspis-based systems with which I have been working, the shared information storage enables access to all of the dynamic information in the system. A flexible agents – evaluators – managers model has made it possible to include tutoring components in existing systems without modification to their source code. In other architectures, other solutions could be used. In the Galaxy-II (Seneff *et al.*, 1998) architecture, one could consider modifying the HUB script so that it provides the necessary information to a tutor component connected to the HUB. In VoiceXML (W3C, 2004a) environments, a tutor could be implemented with a proxy-type component between a server and a browser. The tutor would need to modify the VoiceXML documents and create new ones as necessary.

6 The Work Published in the Articles

6.1 Introduction

The seven publications comprising this thesis examine software tutoring for speech-based user interfaces in two tutoring scenarios with several concrete tutor implementations. In the following, the two scenarios are described, applied research methods discussed, and each publication summarized.

During the dissertation work, two different tutoring concepts were developed to such a stage that meaningful evaluations could be conducted. The development process itself has been a significant part of the work. The concept of software tutoring has no standard implementation yet, so the design process had to be started from a rather open table. It is not possible to test all possibilities for tutor design, since these are endless. The concepts that were selected for further development were discussed in the research group and evaluated in light of current knowledge of spoken interaction and, in the case of the multimodal tutoring, more general usability and interaction knowledge. These general concepts were then applied in an iterative development cycle.

Implementation and integration into existing systems

For both kinds of tutoring, the technical solutions used to implement the tutors and the integration of the tutor with the application to be taught is vital to the applicability of the tutoring concept.

The Jaspis architecture was used heavily in this work. The agents – evaluators – managers paradigm and the shared information storage, both central concepts in the Jaspis architecture, provided the basis for the technical integration of the tutors into the applications.

In both tutoring cases, the applications existed before the idea of tutoring, so tutors had to be integrated later. The integration has been very successful in the sense that no changes were made to the program code of either of the applications. Only some small components had to be replaced. In most cases, the addition of certain components into the system configuration was enough. The integration was also very efficient in the sense that it did not introduce significant delays to the systems. Tutors could control the applications as needed; in fact, more control was possible than was required by the tutoring designs that were used. Sharing of resources between the tutors and the applications was also possible. The speech-based tutoring was integrated into the speech-based e-mail reading application, Mailman (Turunen & Hakulinen, 2000b), so that it used the same input and output resources as the application. The multimodal tutoring, which tutored users on the Busman timetable system (Turunen *et al.*, 2005), physically distributed the tutor to the different computers from which the application was to be run. The solution is one way in which the Jaspis-based application can be extended to be multimodal.

Iterative development and user tests

Iterative development is a development method commonly preferred in the literature on user-centered development. Theoretical and practical design knowledge can provide only some guidelines and ideas for how a human–machine interface should be designed. Observation of real users using a prototype of a system provides great input for the design by showing where users have trouble and what they like.

Iterative design as used in this work includes many kinds of user feedback. When the first working prototypes of tutoring were developed, the members of the research group were the first test users. Since their knowledge of speech user interfaces and the applications for which the tutors were built had a significant effect on their reactions, this can be considered expert evaluation. This approach offers a cheap and fast way to receive initial comments when experts are available.

In the next stages, students were used in the tests. At this stage, the experiments were more formal, with questionnaires used and interactions recorded for later analysis. At this point, the users did not have any special knowledge of speech user interfaces and were indeed using the tutors to learn the applications. Very important findings were made at this point, and the grounding validation of the concept was received. During the iterative development of the different tutors, 36 people in total participated in various tests.

The iterative design process provides an efficient way to develop a new concept. There is no sense in building finished versions just to find out that, because of some design mistakes, they do not work. The user tests also provide insights into the concept and at best provide new, alternative ideas for how tutoring, in this case, could be designed.

Formal evaluations

The last stage of evaluation for both tutors was carefully controlled. Participants were also from a more diverse population. At this point, in addition to the qualitative measurements received in the iteration phase, quantitative measurements were done. These included both metrics for the interaction and subjective assessment with SASSI (Hone & Graham, 2000; 2001) by the participants. The goal was to see whether tutoring could provide benefits over the Web-based manuals and to gather measurements concerning certain design dimensions of tutoring. In total, 45 people participated in the formal evaluations in which the different tutors and their comparison conditions were studied.

To make the quantitative evaluations meaningful, I selected a baseline condition for use in each evaluation. These conditions used Web-based manuals. In the case of the speech-based tutor, I decided to use an existing online manual. This maximized the ecological validity of the evaluation. The Web pages were the method used for learning the Mailman system in real life. One could have, alternatively, selected as the baseline a setting without any guidance. While this would probably have increased the number of significant differences in the results, it is not a realistic scenario, since the Mailman system is not designed to be learned without any guidance.

Also when two versions of multimodal tutoring were evaluated, the baseline was a Web-based manual. This time, the manual was a static version of the tutors. The textual and graphical content from the simpler tutor was taken and modified slightly so that it worked as a single Web page. The idea was to carefully control the guidance and measure the effects of the interactive part of the guidance materials. Once again, the setting could have been set out so that more differences could have been found. In fact, a small test series where no guidance was given was run after the main evaluation. However, the small scale of this evaluation caused the results not to be statistically significant.

6.2 Foundations – Jaspis and Applications (Publication I)

All implementations presented in this work are based on the Jaspis architecture for speech-based systems. Jaspis was developed by the speech-based and pervasive interaction group at the University of Tampere. The development started in 1998, when the author and Markku Turunen began speech user interface research at the University of Tampere. The Jaspis architecture has been presented in detail in the Ph.D. thesis of Markku Turunen (2004).

Publication I presents the Jaspis architecture from the standpoint of accessibility. Jaspis has been developed with adaptiveness as a major goal. The agents – evaluators – managers paradigm used in the entire architecture is built to enable the system to dynamically select the most appropriate components at runtime. Managers form the high-level organization of the architecture. They contain agents, which implement the actual functionality of the applications. The evaluators are used to select an appropriate agent when a manager receives a turn. All agents are stateless, so the most suitable agent can be selected in any situation no matter which agents have been selected previously. The stateless nature is enabled by the information storage solution, where all of the dynamic information is stored. This possibility of adapting to situations by selecting agents has also made it possible to implement tutoring. The paper briefly presents tutoring also, since this is one of the achievements of the architecture. The paper also discusses, in brief, both of the applications in which tutoring was implemented: the Mailman e-mail-reading system and the Busman bus schedule system, both telephone-based spoken-language-dialogue systems.

6.3 Speech-based Tutoring (Publications II – IV)

Three papers have been published that concentrate on the various aspects of the speech-based tutor. Implementation and technical issues, together with the introduction of the tutoring concept, constitute the content of Publication II. Iterative development and the design issues involved for speech-based tutoring are covered in Publication III. Publication IV describes a formal evaluation aimed at validating the tutoring concept, and our implementation, with quantitative measurements.

The following example is part of the user's interaction with the telephone-based Mailman e-mail-reading system when the tutor is active. The example is based on the English-language version of the tutor implemented during the iterative development. The user has just logged in to Mailman for the first time and the speech-based tutor starts guiding the user. The left column consists of the tutor utterances (rounded rectangular balloons), Mailman utterances (rectangular balloons), and user utterances (oval balloons). Long Mailman utterances, such as message listings, have been truncated. The right column provides discussion on the tutor design.

Hello. I’m the tutor, and I will tell you how to use the Mailman system. Next, Mailman will retrieve your e-mail messages from the mail server. This may take a while.	The tutor is embedded in the speech interface and therefore is accessible to all users. The human voice is used for tutoring, since it is easy to understand, anthropomorphizes the tutor, and distinguishes the tutor from the interface, which uses a synthesized voice. The tutor introduces itself and Mailman so that the three dialogue partners have been identified. The tutor takes the initiative and will hold it for a while. Mail retrieval and its temporal extent are explained.
[System processing sound]	
Now the messages have been retrieved. Next, Mailman will tell you whether it found new or just old messages and how many messages it found.	The tutor’s messages are as tightly synchronized as possible within the interface. The tutor tells what the system will say; this will help the user in learning to listen to synthesized speech.
Five new messages.	

Now I will teach you the basic functionality of Mailman: browsing and reading e-mail messages.	The main tasks of the application are presented first at a high level. The system adapts: there is no mention of folders since this user does not have so many messages that folders would appear. If the user will later have folders, the tutor will introduce them at that time.
Now, you will hear a tone that means Mailman is listening to your commands. After the tone, say "list messages" and Mailman will tell you the messages you have.	The turn-taking protocol is introduced. An explicit instruction is given to the user. It contains a concrete example, telling both what to say and how to speak.
[Beep]	
List messages.	
Good. Now you will hear the message listing.	The tutor checks that the input was given successfully. Had there been a problem, the tutor would have discarded the input and asked the user to try again. With explicitly requested input, the tutor can easily check the recognition result to see whether the input was successful. The tutoring prepares the user for a message listing.
First message, from Jaakko Hakulinen, subject…	

Now, select a message by saying, for example, “second message.” Mailman will give you information about the message.	Another explicit instruction is given. However, this time the user can freely choose any message. This way the user can listen to the message in which he/she is most interested.
[Beep]	
Third message.	
Good. The message has been selected.	Successful input is confirmed.
Third message, from…	
Next, say “read” and Mailman will read you the selected message. You can stop the reading by pressing the zero button on your telephone keypad.	One more explicit instruction. These three (five, if folders are present) inputs, which the user is required to use during tutoring, cover the essential functionality of Mailman.
[Beep]	
Read.	

Mailman makes some interpretations of messages to be able to read them better. For example, in this message, quotations from previous messages have been identified, and they are read by mentioning the quotation when it starts. This means that all the “greater than” signs have been removed.	The explanation of message-reading functionality is given when message reading is about to start. The example mentioned in the guidance is dynamically selected on the basis of the message to be read, so that the user will hear the feature in use. Without this kind of explanation, it could be ambiguous to the user whether the system has made such an interpretation if the message seems to have been read as is.
Message from Jaakko Hakulinen, subject…	
Now, we have covered all the basic functionality of the Mailman system. So you can use, for example, the commands “list messages,” “third message,” and “read.” Now you can freely use Mailman. I will tell you about some other features at suitable moments. When you are finished, you can end the call by pressing the zero key when Mailman is listening.	A summary is given at the end of the tightly structured initial instruction. Initiative is then given to the user. Later on, guidance will be given when an appropriate context for each subject arises. During the initial guidance, there was more tutoring content than there was actual interaction with the system. This works for new users but would be unacceptable for experienced users.

Next, the three publications concerning the speech-based tutor are introduced.

Implementation (Publication II)

Being the first of the papers on the tutoring, this publication introduces the concept in brief and then discusses the technical implementation.

The speech-based tutor was tightly integrated into Mailman, the speech-based e-mail-reading application in which it was tutoring users. It was implemented as a separate dialogue partner and used a recorded voice, which separated it from the Mailman system and its synthesized voice. Tutoring components were included in the configuration of Mailman, and they shared all of the technical resources with the application.

The tutor consisted of agents placed in appropriate managers in Mailman and evaluators, which selected the tutoring agents as necessary. As the Jaspis framework supports the use of small agents, it was easy to add these new agents into the system. On dialogue level, the tutoring consists of 25 agents. The Mailman agents did not have to be modified, since they were given a turn and behaved normally when the tutoring system had nothing to do. A special tutor agent was implemented to make the tutoring invisible to the Mailman agents. Each tutoring agent stored the current input/output results when they took turn and the special agent restored these results so that Mailman could proceed from a situation, which looked just like the one before the tutoring took the turn. Since it had access to input and output results, the tutor could discard users' input as needed and could have even controlled Mailman had there been a need for that.

Tutoring logic was controlled mainly on the basis of dialogue history. Tutoring used the same dialogue history as the Mailman system and tutoring message information was included in this history. A user model was also used by the tutor and tutoring information was added to the model when it was presented to the user. Based on these two knowledge sources, tutoring agents could pick the appropriate spot to present the tutoring information using simple rules, such as requiring certain dialogue events to appear before presenting new information. The persistency of the user model makes it possible to give tutoring to users only in the start of use and after the tutor has presented all the content, it will not appear in later sessions anymore.

A few small components were replaced, but, other than that, the tutoring was invisible to the Mailman components, which worked just as they did before. The updated agents used the dialogue history for such actions as context sensitive help. Since the dialogue history now had also the tutoring events, the agents had to be generalized so that they could ignore certain dialogue events.

Design (Publication III)

After the technical issues were solved, the tutor design could be studied by iterative development experimenting with various design solutions. The second publication on the speech-based tutor concentrated on the design issues. It was published in a conference on design. The iterative development of the tutor is covered in reasonable detail. The final version of the tutoring is also described as the result of the iterations. The main contribution of the paper is knowledge of design issues that were handled successfully or proved problematic during the development.

Software tutoring is defined in this paper with five points: the user is tutored to use an application during actual use, guidance is given according to what the user is doing, the tutor can explicitly instruct the user in what to do, the tutor can monitor user actions and adjust its behavior, and the teaching has a structure. This definition has been applied through the entire work presented in this thesis.

The results in the publication are compiled into lessons learned. These guidelines are: teach one thing at a time, at the start tell users explicitly what to do, try to detect problems, signpost the state of the tutoring, use realistic examples in teaching and give users opportunities to control the tutoring.

The guidelines are in line with findings from studies of other types of user guidance, the issues accompanying speech user interfaces, and general usability guidelines. Users should know the status of the tutoring – e.g., how much tutoring is still to be received – and should have some way to control the tutoring, just like users should know the state of a software application and always feel in control. Things should be taught one at a time, and users should be able to test each in practice before moving on, much like the task oriented guidance in minimal manuals. The interactive tutoring approach worked successfully since telling a user explicitly what to do was efficient and appreciated, and the realistic examples worked well. Finally, the importance of error detection and providing guidance appropriately was emphasized. The later work with graphical tutoring tried to follow the guidelines as well improving over the speech-based tutor especially by providing users more control.

Evaluation (Publication IV)

The third paper on speech-based tutoring reports a controlled experiment. In this work, the final version of the tutor was compared to a Web manual. The aim was to find quantitative differences between the two guidance methods. Means of measurement included questionnaires for evaluating the guidance materials and the Mailman system as well as observations of users' interaction with Mailman and the guidance materials.

The evaluation was arranged using a between-subjects setting. Since the tutor is teaching users how to use a system, a within-subjects setting is very hard, if not impossible, to design when external validity is valued. The setting selected, where the comparison condition was the Web-based manual that was the normal guidance used by Mailman users, emphasized ecological validity.

The results show no differences on task completion and during the tasks, interaction was mostly similar between the conditions. The differences in the interaction were on the usage of speech and key commands. The tutor resulted in more speech commands while the web manual users ended up using more telephone keys for interaction.

The interesting differences were found from participants' interaction with the system when guidance material was available. These differences are directly dependent on the control condition and mostly reflect the users' experience when they learn to use Mailman. There were differences on the number of different kind of problems, and the numbers were in favor of the tutor condition with the exception of one case. The average error percentages and the percentages of problematic key usage were lower in the tutor condition, which also had fewer problems with speaking at a wrong time. Speaking too softly was the problem that occurred significantly more often in the tutor condition.

What the tutor did successfully was to teach the interaction style. This can be seen most clearly in the number of times participants spoke at the wrong time – i.e., when the system was not listening. On the other hand, the tutor's spoken messages seemed to give a bad example concerning the volume required. The speaking style of the tutor was rather tranquil and tutor-condition participants often spoke too quietly. Comparison of the number of various types of errors can be seen in Figure 7.

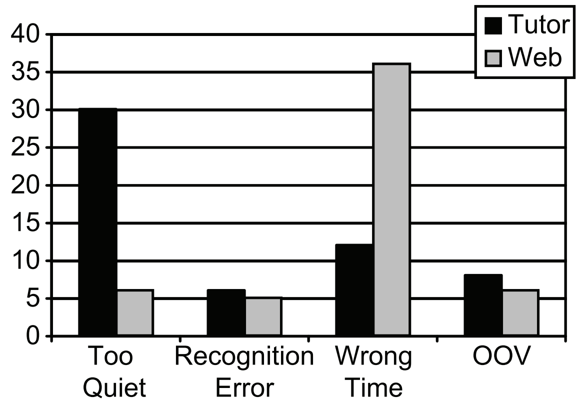


Figure 7. Errors in speech-based tutor evaluations during a training period.

All in all, the strength of tutoring, as judged from all user tests, seems to be that the software tutor can take care of many of those users who would have too many problems with just passive guidance. Not all users necessarily benefit from tutors – more efficient learners may even be slowed down by them – but tutoring can increase the percentage of users who can learn a system with a level of effort they consider acceptable.

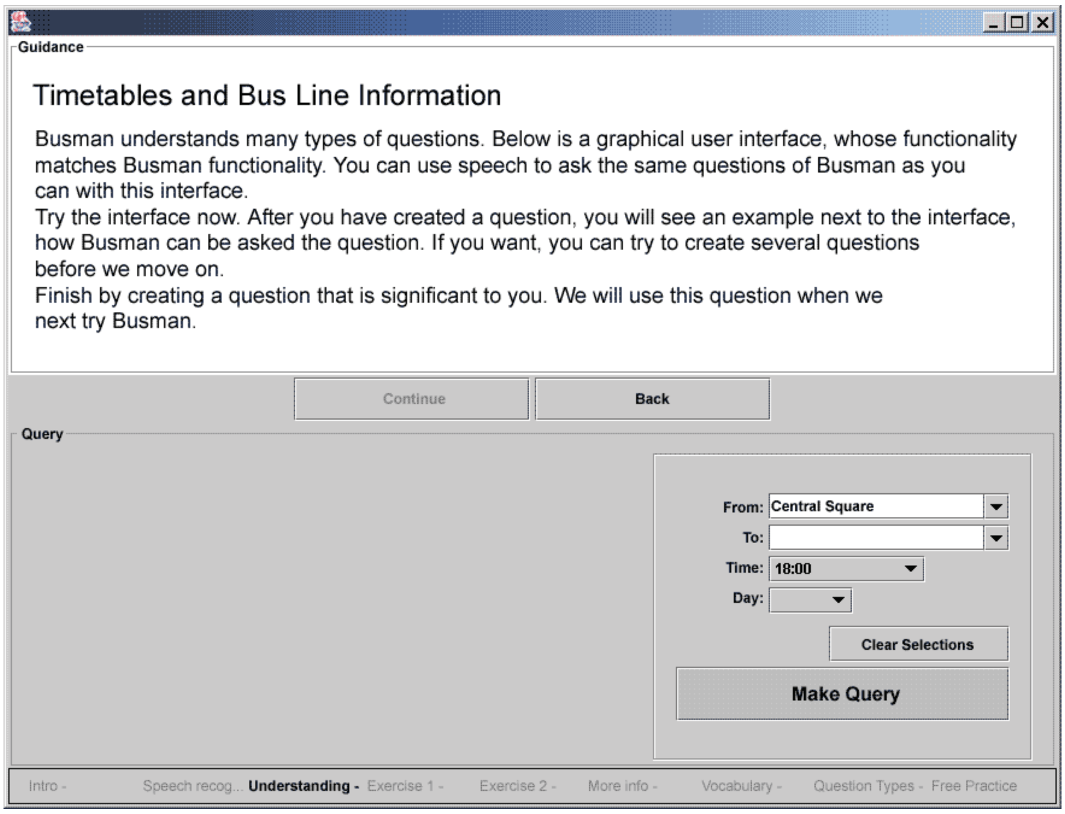
6.4 Multimodal Tutoring (Publications V – VII)

The publications on multimodal tutoring follow the same pattern as those on speech-based tutoring. Implementation issues were published first, design issues and iterative development were covered in the second publication, and a more formal evaluation is reported upon in the final publication.

One motivation for multimodal tutoring was that we were interested in the possibilities of multimodal interaction. We also wanted to test the idea of software tutoring in different settings. Multimodal interfaces have been under active research for a long time, and the speech user interface community is interested in multimodal interaction. This was a possibility to implement support for multimodality in Jaspis. The extension was a reasonably small task. No modification to the Jaspis architecture was necessary, although minor improvements were implemented.

The multimodal tutors were implemented for a different application than the speech-based one. This decision was made because we wanted to test the tutoring concept with different applications featuring different styles of interaction. Using Mailman also for multimodal tutors was considered, since this would have made it possible to compare speech-based and multimodal tutors directly. However, that would have resulted in this dissertation presenting tutoring for only one application, which was considered undesirable.

The example that follows is part of a tutoring session. The example starts after a user has been given some introductory information about Busman, the telephone-based bus schedule system. Speech recognition has also been introduced in brief. The example consists of content from screen captures of the GUI tutor and related commentary. The multimodal tutors were all in Finnish, and the screenshots have been translated into English.


Guidance

Timetables and Bus Line Information

Busman understands many types of questions. Below is a graphical user interface, whose functionality matches Busman functionality. You can use speech to ask the same questions of Busman as you can with this interface.
Try the interface now. After you have created a question, you will see an example next to the interface, how Busman can be asked the question. If you want, you can try to create several questions before we move on.
Finish by creating a question that is significant to you. We will use this question when we next try Busman.

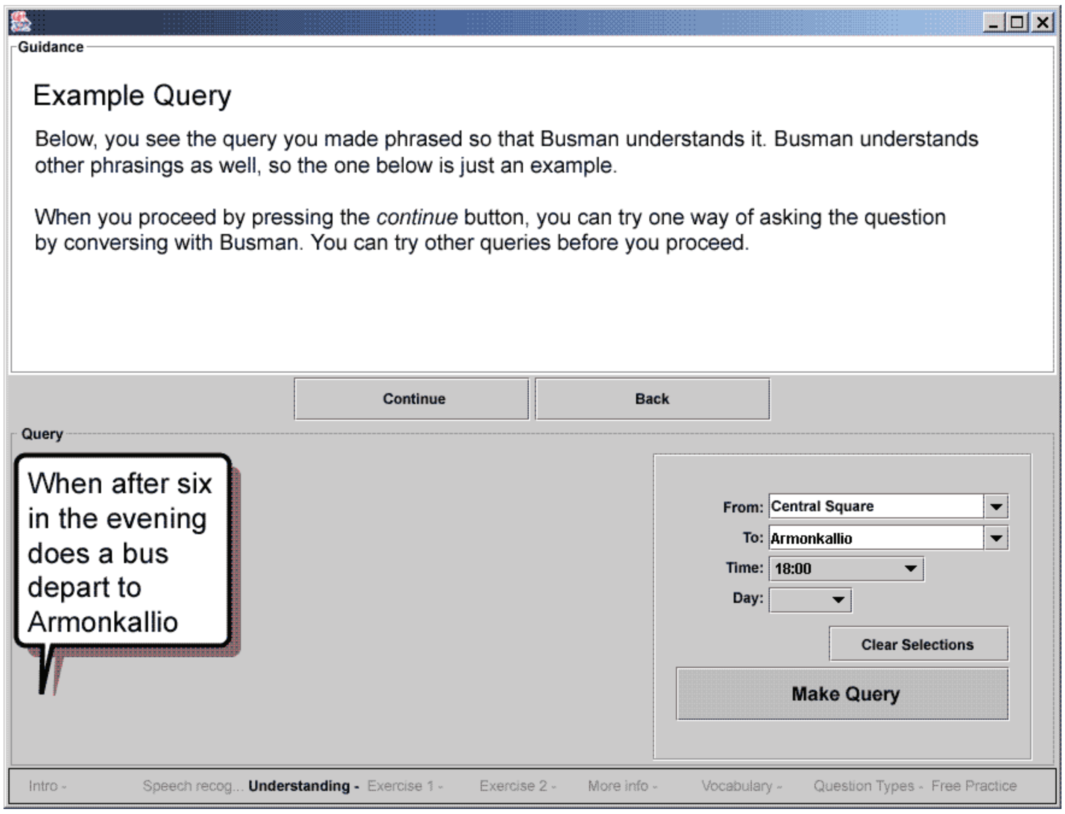
From: Central Square
To:
Time: 18:00
Day:

Clear Selections
Make Query
Intro - Speech recog... Understanding - Exercise 1 - Exercise 2 - More info - Vocabulary - Question Types - Free Practice

Figure 8. A tutor with a Graphical User Interface matching the system functionaly.

In Figure 8 the tutor introduces the GUI part of the tutor to the user. The GUI appears at this point. The GUI is used as an interactive visualization of the Busman functionality. It matches the basic query capabilities of the Busman system but excludes all of the communication-related functionality. At this point, the user is free to experiment with the GUI. The user can use as much time as needed.

The tutor mentions that it will adapt the further tutoring according to the user's choice of question. The tutor recommends creating a question that is relevant to the user.


Guidance

Example Query

Below, you see the query you made phrased so that Busman understands it. Busman understands other phrasings as well, so the one below is just an example.

When you proceed by pressing the *continue* button, you can try one way of asking the question by conversing with Busman. You can try other queries before you proceed.

Query

When after six in the evening does a bus depart to Armonkallio

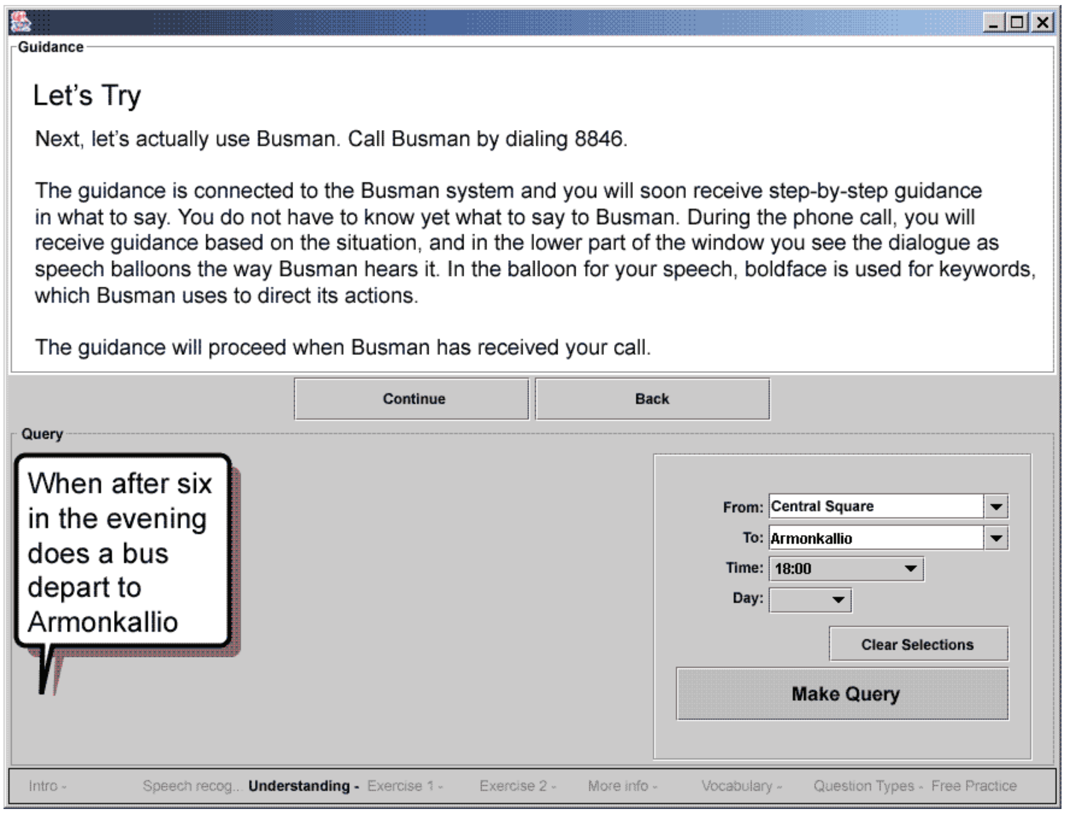
From: Central Square
To: Armonkallio
Time: 18:00
Day:

Continue
Back
Clear Selections
Make Query
Intro - Speech recog... Understanding - Exercise 1 - Exercise 2 - More info - Vocabulary - Question Types - Free Practice

Figure 9. The tutor presenting an example query.

When the user presses the "Make Query" button, an example of how the query can be uttered to Busman is given, as seen in Figure 9. The tutor emphasizes that this phrasing is just an example. The use of a balloon implicitly hints that this is an example of spoken interaction.

It is explained how the tutoring will proceed, and once again a mention is made that the question the user created will be used in the future. The system checks that the question has at least departure and arrival places and a departure time included so that it can be used as an example later on.



Guidance

Let's Try

Next, let's actually use Busman. Call Busman by dialing 8846.

The guidance is connected to the Busman system and you will soon receive step-by-step guidance in what to say. You do not have to know yet what to say to Busman. During the phone call, you will receive guidance based on the situation, and in the lower part of the window you see the dialogue as speech balloons the way Busman hears it. In the balloon for your speech, boldface is used for keywords, which Busman uses to direct its actions.

The guidance will proceed when Busman has received your call.

Query

When after six in the evening does a bus depart to Armonkallio

From: Central Square
To: Armonkallio
Time: 18:00
Day:

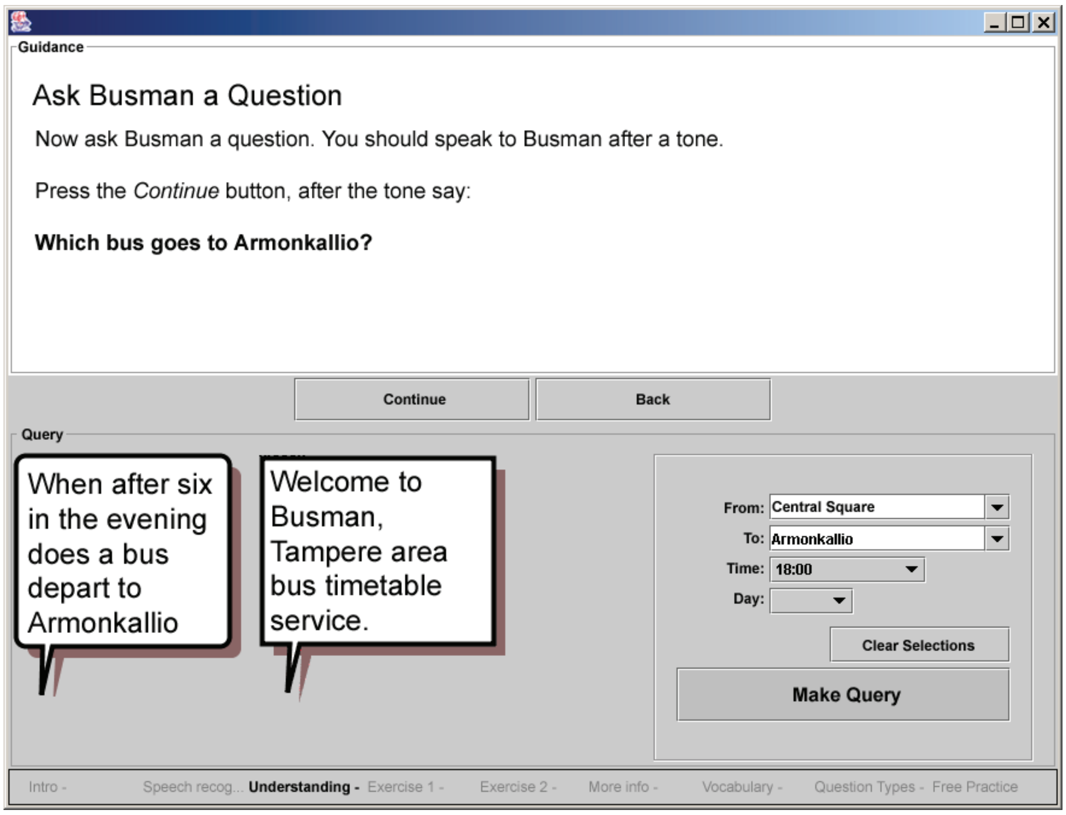
Continue
Back
Clear Selections
Make Query
Intro - Speech recog... Understanding - Exercise 1 - Exercise 2 - More info - Vocabulary - Question Types - Free Practice

Figure 10. The tutor instructs the user how to reach Busman.

Instructions on how to reach the Busman system are now given, as seen in Figure 10. The tutor also mentions that the guidance is connected to the Busman system and more instructions will be given, so that the user does not need to worry about knowing the system already. The real-time visualization of spoken dialogue with speech balloons is introduced to the user.

At this point, the tutor connects to the Busman system and starts waiting. When the user's call is received and Busman has spoken its opening prompt, the tutor halts the system and provides more guidance.

The speech output of the system is presented as balloons just before the speech starts. The balloons scroll onto the screen from the right, and the oldest balloon is removed when more space is needed. The two modalities, speech and text, are used in synchrony to help the user follow the interaction. This can also aid the user in learning to listen to the synthesized speech and helps if the user does not hear some part of the system's speech.



Ask Busman a Question

Now ask Busman a question. You should speak to Busman after a tone.

Press the *Continue* button, after the tone say:

Which bus goes to Armonkallio?

When after six in the evening does a bus depart to Armonkallio

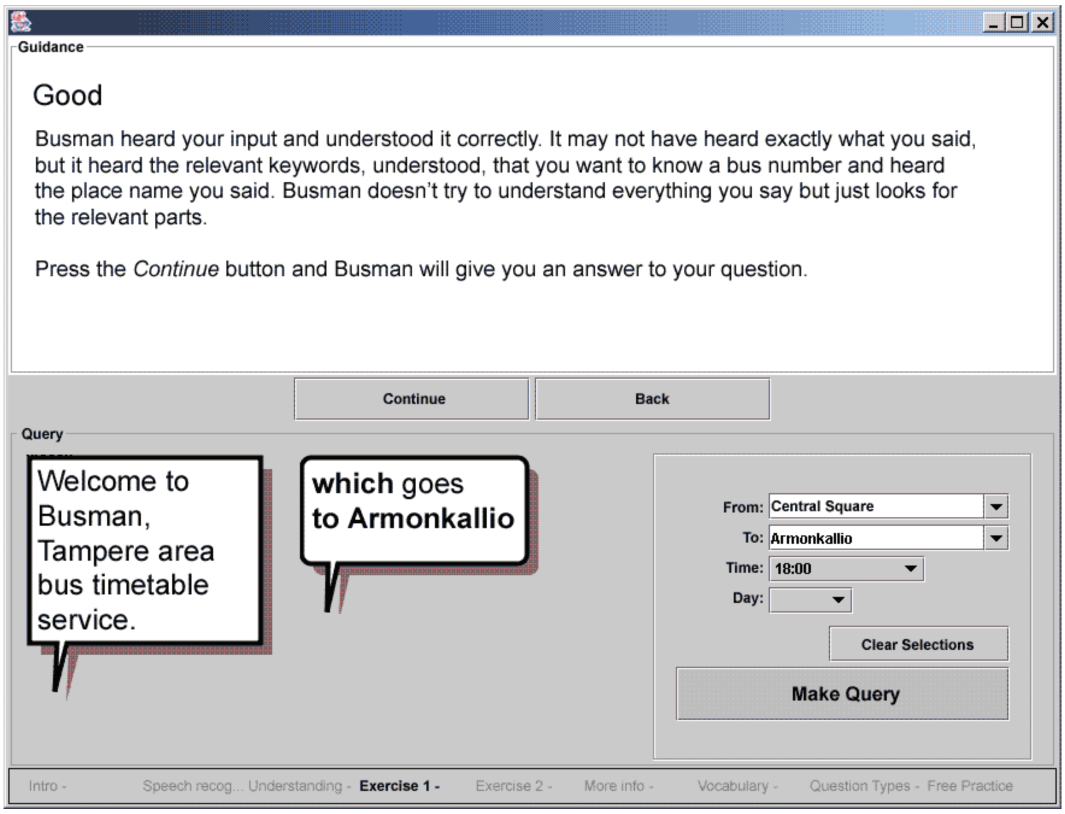
Welcome to Busman, Tampere area bus timetable service.

From: Central Square
To: Armonkallio
Time: 18:00
Day:

Figure 11. The tutor gives an explicit instruction to the user.

Next, the turn-taking protocol of the Busman system is introduced. The user is then asked to ask a specific question of Busman, as seen in Figure 11. The question is a pruned version of the query the user created with the GUI. It does not include any time-related information at this point. This aims at making the first question more likely to be successful and to introduce the system gradually. Since the time will be specified in later questions, also the dialogue management and the system's use of dialogue history are exemplified this way.

The user can control the pace of the tutoring – i.e., indicate when he/she wants to speak to the system. The tutor waits for the user to press the "Continue" button and controls the system appropriately.


Guidance

Good

Busman heard your input and understood it correctly. It may not have heard exactly what you said, but it heard the relevant keywords, understood, that you want to know a bus number and heard the place name you said. Busman doesn't try to understand everything you say but just looks for the relevant parts.

Press the *Continue* button and Busman will give you an answer to your question.

Continue
Back
Query
Welcome to Busman, Tampere area bus timetable service.
which goes **to Armonkallio**
From: Central Square
To: Armonkallio
Time: 18:00
Day:
Clear Selections
Make Query
Intro - Speech recog... Understanding - Exercise 1 - Exercise 2 - More info - Vocabulary - Question Types - Free Practice

Figure 12. The tutor provides feedback.

User input is visualized as a balloon as soon as it has been recognized. The recognition result is visualized, as seen in Figure 12, so that the user can see what the system actually heard. The word "hear" is used instead of the technical term "recognize." According to the experiments with the tutors, users seem to be able to equate recognition errors to mishearing in human–human communication. Keywords spotted by the system are in boldface. This provides information about language processing without adding much complexity to the visualization.

There are always two or three balloons on the screen, leaving a short dialogue history visible and thus removing some of the problems caused by the sequential and temporal nature of speech.

The tutor checks the user's input and the state of the Busman system. It informs the user about the recognition results, which are also visible as a speech balloon. A semantically correct result is accepted since, with the language model of about 1500 words, it makes no sense to demand perfect recognition. The nature of word-spotting-type recognition and concept-spotting type input processing are also explained in this context. If a significant error had occurred, the user would have been asked to try again after hearing the system's response. This way, the error correction procedure is introduced as well. More detailed instruction on how to speak is delivered when errors occur.

The tutoring continues so that the user is asked to give two more inputs and some more guidance is given. At the end, the user can freely experiment with the tutor while the GUI is available and interaction is visualized with speech balloons.

Implementation (Publication V)

The technical solutions of the multimodal tutors were introduced in the first publication on these tutors. Because of this, the concept was also introduced in the publication. The publication discusses the technical solutions in a wider perspective since the same solution can be applied to visualization and debugging of speech applications in addition to tutoring.

While the tutor was built so that both its logic and the graphical interface were run on the user's computer, it had to be able to communicate with the application in real time. Visualizations presented the system state as the dialogue proceeded. The initial implementation was already fast enough, and the technical solutions implemented were successful.

The implementation of multimodal tutoring solved several challenges. Graphical components, i.e., the tutor interface, or a visualization component, has to run on the user's local computer as a standalone component, while the Busman application runs on a server. To connect these two components, the implementation features an additional "Middleman" component, as seen in Figure 13. The proxy-like Middleman was implemented to simplify creation of the necessary network connection and make the connection more robust. Since the tutoring can be a Java-applet, it cannot be a server, and the application server can be behind a firewall limiting network traffic. Middleman is a server for both components and overcomes these limitations. Middleman was designed also to improve the efficacy of the tutor/application communication by caching requests and results passed through it.

A tutor can send various requests to the application via Middleman. These are run, pause, IOResult, and ISOperation. The first two are used to ask the application to function normally or to halt temporarily. The two other requests are used to query and modify the content of the applications information storage. The Middleman caches the requests and the responses so that there is no need for unnecessary connections and processing after each connection to Middleman. The requests can be asynchronous or half or fully synchronous so that the speed of different operations can be optimized by running the different components concurrently. The caching of requests also improves robustness, since the Middleman can keep the application running even if the tutor component has problems, and vice versa.

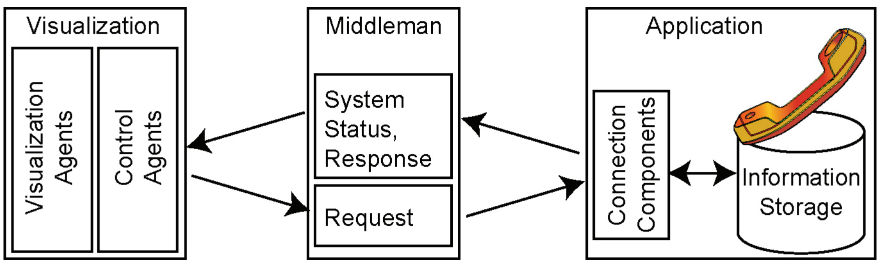


Figure 13: The architectural structure of the multimodal tutors.

In the application end, the usage of tutoring requires the addition of connection components. These can be added simply by including relevant entries into the systems configuration file. Since Jaspis has a shared information storage and a default, XML-based interface to query its content, the standard connection components can provide all of the application's internal data to the tutoring component. Possibility to also modify the information storage enables the tutor to, for example, discard user inputs.

Real-world deployment of the form of multimodal tutoring presented is possible using, e.g., Java applets on a Web page that will communicate with the system on a server via Middleman. What needs to be included is support for multiple concurrent users so that two instances of the tutor would not try to communicate with a single instance of Busman simultaneously.

Design (Publication VI)

The inclusion of graphics and graphical user interfaces provides numerous possibilities for tutor design. The lessons learned from the speech-based tutor directed our design to a certain degree, but still a variety of possibilities in the area of visualizations and GUI interaction remained. Because of this, we ended up evaluating four different concepts, which we called balloon-tutor, form-tutor, GUI-tutor, and animated-tutor.

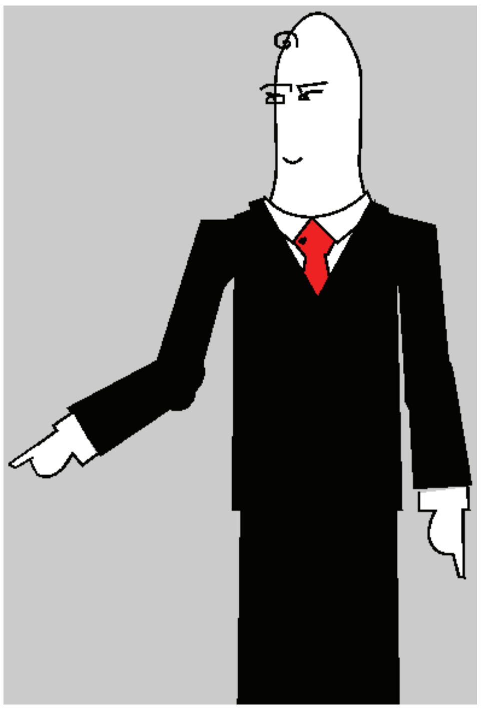


Figure 14. The embodied tutoring agent.

Balloon-tutor was the most basic tutor; in addition to textual guidance, explicit instructions, and simple adaptive guidance, it featured a visualization of spoken dialogue with comic-book-style speech balloons. All of the other tutors featured the same elements as balloon-tutor and some additional items. Form-tutor included also a visualization of the form structure used for dialogue management in the Busman system. Users considered this unnecessary information. Our GUI-tutor included a graphical user interface used to construct queries that could be made of the Busman system using speech. While promising, the GUI was not used as much as we had hoped. Its design was later improved to encourage experimentation. Animated-tutor featured an embodied agent, as seen in Figure 14, doing the tutoring, and the tutor visualized the activity of several components of the system, including speech input and output. The design of the embodied agent provided the persona effect (Lester *et al.*, 1997) for some. However, it was considered irritating by many others and was discarded due to this controversy.

The concept of graphical tutoring was well received in the user tests and all but one user reported that they learned to use a spoken dialogue system with the help of a tutor. The visualization of spoken dialogue was often considered the best part of the guidance and participants' comments suggested that it really helped when speech recognition errors occurred. Another successful feature was the use of a notification sound to direct users' attention from the dialogue application to the tutoring when new tutoring content appeared on screen. The biggest problems seemed to be in explaining to the users the rather open nature of the dialogue system's speech recognition grammar. Many participants considered the given examples as the only valid phrasings. The two tutors which supposedly taught most about the grammars, GUI-tutor and form-tutor, were not commonly used to the extent that they would have achieved this goal.

When the four prototypes were tested, the two most promising versions were balloon-tutor and GUI-tutor. These were further developed by rectifying the usability problems that were revealed in the tests and by including some additional ideas that came about during the tests. One of these ideas was to emphasize the concept of keywords to better explain what kind of language the system understands. In addition to using the keyword term in guidance texts, we updated the visualization of spoken dialogue to bold the keywords.

Evaluation (Publication VII)

The evaluation of the multimodal tutors included the two versions of tutoring and a static HTML manual consisting of the same information. The three guidance conditions can be roughly placed on a continuum according to the amount of user-tutor interaction possible. The HTML document is essentially passive guidance. Both tutors included the real-time visualization of spoken interaction, and they gave users explicit instructions and monitored the resulting dialogue. The more interactive tutor included also a graphical user interface, which the users could use to create spoken queries graphically.

The effects of this interactivity were studied in a between-subjects setting. Questionnaires were used to collect participants' perceptions of the guidance materials and the Busman system. The results of this study suggest that the increased interactivity improved users' perceptions.

In users' subjective evaluations, GUI-tutor (or Form tutor as it was called in this publication) received the best overall evaluation, significantly better than the Web-based tutor, which received the lowest score. The increased interactivity provided by the GUI part of the tutoring improved participants' perception of the guidance. Users seem to be able to transfer the mental model of a system based on the GUI to a speech interface.

Some background information also correlated with certain system evaluation. Participants' reports on their own computer skills correlated with questions "The system is pleasant", "The system is friendly", "The interaction with the system is irritating", "The interaction with the system is frustrating", and "The system responds too slowly". In all cases, those with better computer skill considered the dialogue system worse. However, computer skills did not correlate with error rates or other interaction metrics, nor did they correlate with guidance evaluations.

Task completion and interaction during the tasks was similar between the conditions. During the learning period, error rates were also similar on average but the variance was higher in the static guidance condition. This supports the finding made already with the speech-based tutor that tutoring can better support all kinds of users while static guidance can work well for some but not for others. A tutor is most helpful when users need a lot of help. There were users who had great trouble in learning to use Busman with the online manual, and some did not manage to do this at all. However, to the extent that we have experienced, the tutors can take care of everybody.

7 Discussion and Conclusions

In this dissertation, the concept of software tutoring in speech user interfaces was explored by building various functional prototypes of software tutoring for two different existing spoken-language-dialogue systems. Iterative development resulted in insights into the concept and enabled the development of working software tutors. Formal evaluations showed that tutors can help users to learn a new system with less trouble than traditional static manuals involve. A tutor can detect the problems users have, inform users about them, and provide the necessary guidance.

Designing an efficient tutor is not simple or easy to undertake. The design-related articles in this work disseminate the design knowledge we have gained. An iterative development style is recommended for the designer of tutors, perhaps with evaluation procedures slightly less formal than those featured in this work. Observing actual users learning an application with a tutor is the best way to see whether design decisions are good or bad. According to our experiences, the lessons learned over the decades on guidance design, particularly the principles of minimal manuals, such as efficient use of text and action oriented teaching, are applicable also in tutor design. Naturally, also the knowledge on the design of speech user interfaces must be taken into account.

7.1 THE STRENGTHS OF SOFTWARE TUTORING

The tutors implemented in this work are at their best when users who have never used speech-based interfaces are tutored. The formal evaluation of multimodal tutors, reported in Publication VII, in particular, suggests that there are significant differences between different user groups. Those who are very familiar with computers prefer more controllable and less involving guidance, while less experienced users, female participants, and unconfident computer users had more positive views of various aspects of the tutoring-style guidance.

What the tutoring can do better than other guidance methods is check that users can indeed give commands to the system successfully. Monitoring user actions after giving explicit instructions makes error detection easy. Because of this error detection and guidance, error rates during the training period were significantly lower for those who learned to use Mailman with a tutor as compared to those who use a web manual. Most important strength of the tutor visible in these results, reported in Publication IV, relates to learning the interaction style. Since Mailman did not feature barge-in, many users spoke at times, when the system was not listening. Significantly fewer such errors occurred in a tutor condition, than in a web-guidance condition. The explicit instructions and corresponding check for successful input made sure that every user knows when and how so speak to the system. During the experiment, it was also often noticed that users who feel uneasy about using a new system particularly appreciate the explicit instructions. Overall, the results show most benefits in tutoring when users who are completely new to speech user interfaces are tutored.

Since tutoring can be embedded in a speech user interface, its accessibility is excellent. Whenever the system is accessible, tutoring can be accessed as well. However, this requires that the tutoring be restricted to the modalities and input and output devices used by the system itself. On the other hand, if the additional modalities can be used, other benefits can be gained. Four different tutoring concepts using graphics were tested, as reported in Publication VI, and real time visualization of spoken interaction with speech balloon was received favorably. Such a visualization enabled users to see exactly what a computer has heard and thus spot and understand recognition errors. Visualization of more complex details of system behavior was not as successful as most participants did not value information that is not directly relevant to the spoken interaction.

When compared to guidance embedded into a speech user interface, tutoring can be seen as a separate component also from the users' point of view. The use of a separate dialogue partner for tutoring enables the actual interface to remain static while the amount of guidance is adjusted according to the users' learning. This helps users to gather and maintain a coherent mental model of the speech user interface.

7.2 THE LIMITATIONS OF SOFTWARE TUTORING

While tutoring has numerous benefits and the evaluations presented have shown favorable figures, there are certain limitations and challenges that must still be considered. These are relevant concerns when one considers using tutoring.

First of all, tutoring is not practical in all cases. If most users will use the application only once, a tutor cannot defend its place. Such systems should embed all the necessary guidance into the system. Tutoring is a more sensible solution when users will use a system several times. Users with expertise on similar systems may also despise tutorials and prefer guidance that is completely under their control.

Another burden of tutoring is that it requires certain technical implementation effort, which is not needed, e.g., when static web manuals are produced. Implementation is not a trivial task. While there have been some efforts toward automating tutor generation, they have not been successful to an extent that would help someone developing a tutor for a speech-based interface today.

The effort required in order to integrate a tutor into an existing application depends greatly on the architecture upon which the application is built. The same is true, only to a lesser extent, for new applications for which a tutor is created in concert with development of the application itself. Certain programming effort is also needed to implement any intelligence in tutoring. While not that many lines of code may be necessary, the logic may be rather complex and hard to define. The complexity of spoken interaction results in many situations that must be considered in tutoring logic as well. It has been argued that, for the time being, more traditional guidance would be a better solution than complex, interactive guidance component (Capobianco & Carbonell, 2003).

Accessibility may be a problem for tutoring. The multimodal tutor poses a problem for users in special user groups. Visually impaired users may not be able to use a graphical tutor at all. Often, HTML-based guidance is accessible, to a reasonable extent, to these users by means of voice-based HTML readers or screen-reader applications. An interactive tutor is much less likely to work sufficiently well with screen readers, and therefore it would exclude these users.

7.3 COMPARISON TO OTHER FORMS OF GUIDANCE

Compared to printed manuals, a tutorial can potentially avoid the paradox of active users. While it is easy to ignore printed manuals and just forget them on a shelf, a tutor that is integrated into the system has better chances of actually being used. The fact that software tutoring is very popular in computer and video games suggests just that. Games form an area where users cannot be assumed to have any interest in putting extra effort into learning a system; they are meant to be fun, and learning to play should be easy. Reading manuals is therefore not common. Based on this, we see that a good software tutor could make learning a system effortless, even fun. However, lately there has been some commentary on video game journals and forums about the negative side of tutorials, how they are tedious, and annoying to active gamers, who can easily figure out the needed skills based on their experiences with other games (Edge, 2006). This reminds that the users should have the overall control over tutoring, i.e., at least a possibility to choose not to use one. This issue was raised also in the study reported in Publication III, as several participants in the studies asked for more possibilities to control the rather autonomous speech-based tutor.

Software tutors can also provide guidance in the modality the system is using, and guidance can therefore be more realistic. The structure of the guidance can also be enforced, if desired, more strictly than can that of a freely browsable printed manual. Finally, the interactive nature of tutoring enables it to adapt to the user's needs and to take care of each user on a personal level.

The accessibility of tutors, as compared to printed manuals, can be better or worse. An embedded tutor is always accessible, while a paper manual is probably not carried around in mobile use, and even delivering one could be difficult in the case of many applications. On the other hand, a small piece of paper can be carried around, so it may be a suitable solution in some cases. Also, a multimodal tutor, as is mentioned above, can have significant accessibility problems.

Electronic manuals, especially online help, have the same accessibility as embedded tutors do. However, they suffer from the paradox of active users to perhaps almost as great an extent as printed manuals do. They can be more context-sensitive, but they still lack the proactive nature of software tutoring.

Embedded assistance differs from tutoring in its more passive nature. It is context-sensitive in most cases and can even be proactive. However, it is still part of the user interface and cannot guide new users so carefully. The user must be more active in exploring the system. This suits some users well, but an actively guiding tutor eases the start for other users significantly.

In most cases, a tutorial cannot replace any other guidance method completely. It is still a good idea to have traditional documentation, especially reference manuals, available. Most users who receive tutoring will not need the manuals, but there are situations in which a static, detailed document defends its existence. In speech user interfaces, online manuals are not viable, due to the slow output rate of speech. Embedded assistance, on the other hand, is very common. Tutoring does not replace the embedded assistance, which is such an integral part of spoken-language-dialogue systems. However, tutoring can teach many higher-level concepts and make sure users know the technical limitations of the system and learn the interaction style. After this, the embedded assistance can be dedicated to its main task, guiding the user in dialogue structure and ensuring that users are in such a cognitive state that they can naturally say the right words to the system.

7.4 Conclusions

The first research question in this thesis was "How can software tutoring be implemented in spoken-language dialogue systems?". Technically, the two implemented tutors take very different approaches. The speech-based tutor was tightly integrated into an existing application, sharing many resources with it. The multimodal tutor, on the other hand, distributed the tutor and the application into different computers while maintaining time synchronization. Both solutions were possible because of the flexible architecture, Jaspis. The technical applicability of tutoring depends greatly on the architectural solutions that the application is built on.

Tutoring, as defined in this thesis, requires access to the application so that it can monitor users' interaction and synchronize its actions to the application events. Possibility to, at least, halt the application temporarily is also important. Naturally, if the tutor is built at the same time as the application, the technical requirements of the tutor can be taken into account.

The second research question was "How should the tutors be designed?" The potential variety in software tutoring is endless, even in a limited domain such as spoken-language-dialogue systems. In this work, I have taken two different approaches in order to gain insight into the concept and its application. While the implementations presented in the thesis do not aim to be comprehensive in covering the software tutoring in speech user interfaces, they do exemplify those features of tutoring that I consider essential.

Structured guidance, explicit instructions, and adaptive guidance based on monitoring of the user's interaction with a system and controlling the system to the necessary extent are the features that separate tutoring from other forms of guidance. This was the foundation during the development of the tutors. Much experience on the design of tutors was gained during the iterative development. It was found that it is important to let users have overall control and give them an overview of the progress of the tutoring. Interactivity in tutoring seems to improve users' perception of the tutor. The style where the user was first tutored in one feature and immediately asked to try it out worked well. It is important to teach one thing at a time. The overall context for the design of the software tutors for speech user interfaces is in the guidance material design knowledge, especially tutorial and minimal manual type of guidance and speech user interface design.

The two different solutions, with different modalities, had their own challenges and strengths. The slow output rate of speech was tangled surprisingly well with the speech-based tutor. As long as only one thing was taught at a time, users were willing and capable of listening to and comprehending spoken guidance. The strengths of a spoken tutor are accessibility and realistic examples. A multimodal tutor can potentially benefit from the strengths of the visual modality and simultaneous tutoring and application activity. These are also challenges. The visual elements must have relevance to the spoken interaction. When this was not the case, the participants in the evaluations did not appreciate the visual elements. Simultaneous content must have particularly direct correspondence since people cannot concentrate on two things at the same time. When guidance and usage happen one after another in different modalities, one must also take care that the focus shift between the tutoring and the application usage takes place naturally. This was achieved by the use of sound in the implemented tutors.

The final research question was "Can we gain some benefits with tutoring over other guidance methods?" While the results do not show significant differences in the learning results measured by task completion rates, many users seem to benefit greatly from tutoring in the form of a less troublesome learning period. When the tutoring was tested, each tutor version could teach every test participant the basic usage of the relevant system. The other forms of guidance often resulted in very inefficient use on account of misunderstanding of the communication style required, or even in the user just giving up. This shows that tutoring can be seen as a kind of safety net, or akin to a set of training wheels, helping those who are most in need of guidance.

The ultimate aim for software tutoring in the context of spoken-language-dialogue systems is to provide a partial solution to the "what can I say" problem and to provide "how should I speak" information also. Speech user interfaces do not support the unlimited natural language featured in human–human dialogue, and the communication style as well has significant limitations. Tutoring has been successful in teaching the communication style and the other significant limitations to those who have never used a speech user interface before. Using tutoring in certain speech user interfaces could render the use of these services easier for many users and reduce the negative effect that may occur when speech user interfaces fail to match user expectations.

8 References

Aaronson, A. & Carroll, J. M. (1987) Intelligent help in a one-shot dialog: A protocol study. *Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface, CHI+GI 1987* (Toronto, Ontario, Canada). ACM Press, pp. 163–168.

Allen, J. F., Ferguson, G., & Stent, A. (2001) An architecture for more realistic conversational systems. *Proceedings of Intelligent User Interfaces, IUI-01* (Santa Fe, NM, USA). ACM Press, pp. 1–8.

Allen, J. F., Miller, B. W., Ringger, E. K., & Sikorski, T. (1996) A robust system for natural spoken dialogue. *Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics, ACL '96* (Santa Cruz, CA, USA). pp. 62–70.

Ames, A. L. (2001) Just what they need, just when they need it: an introduction to embedded assistance. *Proceedings of the 19th Annual International Conference on Computer Documentation* (Sante Fe, NM, USA). ACM Press, pp. 111–115.

Arons, B. (1997) SpeechSkimmer: A system for interactively skimming recorded speech. *ACM Transactions on Computer–Human Interaction,* Vol. 4, No. 1, pp. 3–38.

Badin, P., Bailly, G., Raybaudi, M., & Segebarth, C. (1998) A 3-dimensional linear articulatory model based on MRI data. *Proceedings of the 3rd ESCA/COCOSDA International Workshop on Speech Synthesis* (Sydney, Australia), pp. 249–254.

Bergman, L., Vittorio, C., Lay, T., & Oblinger, D. (2005) DocWizards: a system for authoring follow-me documentation wizards. *Proceedings of the 18th annual ACM symposium on User interface software and technology, UIST'05* (Seattle, WA, USA). ACM Press, pp. 191–200.

Barnicle, K. (2000) Usability testing with screen reading technology in a Windows environment. *Proceedings on the 2000 conference on Universal Usability, CUU'00* (Arlington, VA, USA). ACM Press, pp. 102–109.

Bernsen, N. O. & Dybkjær, L. (2000) A Methodology for Evaluating Spoken Language Dialogue Systems and Their Components. *Proceedings of the 2nd International Conference on Language Resources and Evaluation, LREC 2000* (Athens, Greece). pp. 183–188.

Black, J. B., Carroll, J. M., & McGuigan, S. M. (1987) What kind of minimal instruction manual is the most effective. *Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface, CHI+GI 1987* (Toronto, Ontario, Canada). ACM Press, pp. 159–162.

Boda, P. (2000) Visualisation of spoken dialogues. *Proceedings of 6th International Conference on Spoken Language Processing, ICSLP 2000* (Beijing, China). Vol. 1, pp. 146–149.

Bolt, R. A. (1980) "Put-that-there": Voice and gesture at the graphics interface. *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '80* (Seattle, WA, USA). ACM Press, pp. 262–270.

Borenstein, N. S. (1986) Help texts vs. help mechanisms: A new mandate for documentation writers. *Proceedings of the 4th Annual International Conference on Systems Documentation* (Ithaca, NY, USA). ACM Press, pp. 78–83.

Brennan, S. E. (1991) Conversation with and through computers. *User Modeling and User-Adapted Interaction*, Vol. 1, No. 1, pp. 67–86.

Brown, E., Srinivasan, S., Coden, A., Ponceleon, D., Cooper, J., Amir, A., & Pieper, J. (2001) Towards speech as a knowledge resource. *IBM Systems Journal,* Vol. 40, No. 4, pp. 985–1001.

Bulyko, I., Kirchhoff, K., Ostendorf, M., & Goldberg, J. (2005) Error-correction detection and response generation in a spoken dialogue system. *Speech Communication,* Vol. 45, No. 3, pp. 271–288.

Cahn, J. E. & Brennan, S. E. (1999) A psychological model of grounding and repair in dialog, *Working Papers of the AAAI Fall Symposium on Psychological Models of Communication in Collaborative Systems*. AAAI, pp. 25–33.

Capobianco, A. & Carbonell, N. (2003) Online help for the general public: Specific design issues and recommendations. *Universal Access in the Information Society Journal, UAIS,* Special Issue: "Countering design exclusion," Vol 2, No. 3, pp. 265–279.

Carroll, J. M. & Aaronson, A. P. (1988) Learning by doing with simulated intelligent help. *Communications of the ACM,* Vol. 31, No. 9, pp. 1064–1079.

Carroll, J.M. & Carrithers, C. (1984) Training wheels in a user interface. *Communications of the ACM,* Vol. 27, No 8, pp. 800–806.

Carroll, J. M. & Kay, D. S. (1985) Prompting, feedback and error correction in the design of a scenario machine. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '85* (San Francisco, CA, USA). ACM Press, pp. 149–153.

Carroll, J. M. & Rosson, M. B. (1987) Paradox of the active user. In J. M. Carroll (ed.), *Interfacing Thought: Cognitive Aspects of Human–Computer Interaction*. MIT Press, Cambridge, MA, USA.

Cassell, J., Nakano, Y. I., Bickmore, T. W., Sidner, C. L., & Rich, C. (2001) Non-verbal cues for discourse structure. *Proceedings of Joint EACH-2001 ACL Conference.*

Contreras, J. & Saiz, F. (1996) A framework for the automatic generation of software tutoring. *Proceedings of Computer-Aided Design of User Interfaces*.

Covi, L. M. (1995) Such easy-to-use systems!: How organizations shape the design and use of online help systems. *Proceedings of the Conference on Organizational Computing Systems, COOCS'95* (Milpitas, CA, USA). ACM Press, pp. 280–288.

Cox, R. V., Kamm, C. A., Rabiner, L. R., Schroeter, J., & Wilpon, J. G. (2000) Speech and language processing for next-millennium communications services. *Proceedings of the IEEE,* Vol. 88, No. 8, pp. 1314–1337.

Cutler, A., Dahan, D., & van Donselaar, W. (1997) Prosody in the comprehension of spoken language: A literature review. *Language and Speech,* Vol. 40, No. 2, pp. 141−201.

Danieli, M. & Gerbino, E. (1995) Metrics for evaluating dialogue strategies in a spoken language system. *Proceedings of the AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation.* pp. 34–39.

Desurvive, H., Caplan, M., & Toth, J. A. (2004) Using heuristics to evaluate the playability of games. *CHI '04 extended abstracts on Human factors in computing systems* (Vienna, Austria). ACM Press, pp. 1509–1512.

Duffy, T.M., Mehlenbacher, B., & Palmer, J.E. (1992) *On line help, design and evaluation.* Ablex Publishing Corporation, Norwood, NJ, USA.

Edge (2006) Monkey Magic. *Edge,* No. 161, p. 27.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., & Reddy, D. R. (1980) The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys,* Vol. 12, No. 2, pp. 213–253.

Eskenazi, M. & Black, A. W. (2001) A study on speech over the telephone and aging. *Proceedings of the 7th European Conference on Speech Communication and Technology, Eurospeech 2001* (Aalborg, Denmark). ISCA, pp. 171–174.

Eurostat (2006) Eurostat news release 83/2006.

Fischer, G., Lemke, A. C., & Mastaglio, T. (1990) Using critics to empower users. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '90* (Seattle, WA, USA). ACM Press, pp. 337–347.

Fischer, G., Lemke, A., & Schwab, T. (1985) Knowledge-based help systems. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '85 Proceedings* (San Francisco, CA, USA). ACM Press, pp. 161–167.

Flycht-Eriksson, A. & Jönsson, A. (1998) A spoken dialogue system utilizing spatial information. *Proceedings of the 5th International Conference on Spoken Language Processing, ICSLP 1998* (Sydney, Australia). ASSTA, pp. 1207–1210.

Fry, J., Ginzton, M., Peters, S., Clark, B., & Pon-Barry, H. (2001) Automated tutoring dialogues for training in shipboard damage control. *Proceedings of the 2nd SIGdial Workshop on Discourse and Dialogue* (Aalborg, Denmark). ACL.

Fussell, S. R., Setlock, L. D., Yang, J., Ou, J., Mauer, E., & Kramer, A. D. I. (2004) Gestures over video streams to support remote collaboration on physcal tasks. *Human-Computer Interaction*, Vol. 19, No. 3, pp. 273–309.

García, F. (2000) CACTUS: Automated tutorial course generation for software applications. *Proceedings of the 5th international conference on Intelligent user interfaces, IUI 2000* (New Orleans, LA, USA). ACM Press, pp. 113−120.

Glass, J. R. & Hazen, T. J. (1998) Telephone-based conversational speech recognition in the Jupiter domain. *Proceedings of the 5th International Conference on Spoken Language Processing, ICSLP 1998* (Sydney, Australia). ASSTA, pp. 1327–1330.

Gong, R. & Elkerton, J. (1990) Designing minimal documentation using a GOMS model: A usability evaluation of an engineering approach. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '90* (Seattle, WA, USA). ACM Press, pp. 99–106.

Gong, L. & Lai, J. (2003) To mix or not to mix synthetic speech and human speech? Contrasting impact on judge-rated task performance versus self-rated performance and attitudinal responses. *International Journal of Speech Technology*, Vol. 6, No. 2, pp. 123–131.

Gorrell, G., Lewin, I., & Rayner, M. (2002) Adding intelligent help to mixed initiative spoken dialogue systems. *Proceedings of the International Conference on Speech and Language Processing, ICSLP 2002* (Denver, CO, USA,). ISCA.

Gould, J. D. & Lewis, C. (1985) Designing for usability: key principles and what designers think. *Communication of the ACM,* Vol. 28, No. 3, pp. 300–311.

Graesser, A. C., VanLehn, K., Rosé, C. P., Jordan, P. W., & Harter, D. (2001) Intelligent tutoring systems with conversational dialogue. *AI Magazine*, Vol. 22, No. 4, pp. 39–52.

Grice, H. P. (1975) Logic and conversation. *Studies in the Way of Words*. (1989) Harvard University Press. Originally published in *Syntax and Semantics,* No. 3.

Guindon, R. (1988) How to interface to advisory systems? Users request help with a very simple language. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI'88* (Washington, D.C., USA). ACM Press, pp. 191–196.

Gustafson, J., Lundeberg, M., & Liljencrants, J. (1999) Experiences from the development of August – a multi-modal spoken dialogue system. *Proceedings of ESCA Workshop on Interactive Dialogue in Multi-Modal Systems, IDS '99*.

Gustafson, J., Elmberg, P., Carlson, R., & Jönsson, A. (1996) An educational dialogue system with a user controllable dialogue manager. *Proceedings of the 4th International Conference on Spoken Language Processing, ICSLP 96* (Philadelphia, PA, USA). ESCA.

Harrison, S. M. (1995) A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface. *Proceedings of the SIGCHI conference on Human factors in computing systems , CHI '95* (Denver, CO, USA). ACM Press, pp. 82–89.

Hartikainen, M., Salonen, E.-P., & Turunen, M. (2004) Subjective evaluation of spoken dialogue systems using SERVQUAL method. *Proceedings of the 8th International Conference on Spoken Language Processing, ICSLP 2004* (Jeju, Korea). ISCA, pp. 2273–2276.

Hassenzahl, M., Platz, A., Burmester, M., & Lehner, K. (2000) Hedonic and ergonomic quality aspects determine a software's appeal. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2000* (The Hague, The Netherlands). ACM Press, pp. 201–208.

Heeman, P. A., Johnston, M., Denney, J., & Kaiser, E. (1998) Beyond structured dialogues: Factoring out grounding. *Proceedings of the 5th International Conference on Spoken Language Processing, ICSLP 1998* (Sydney, Australia). ASSTA.

Heisterkamp, P. (2001) Linguatronic product-level speech system for Mercedes-Benz cars. *Proceedings of the 1st International Conference on Human Language Technology Research, HLT '01* (San Diego, CA, USA). ACL, pp. 1–2.

Hirschman, L. & Thompson, H. S. (1996) Overview of evaluation in speech and natural language processing. In R. A. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, & V. Zue (eds.), *Survey of the State of the Art in Human Language Technology*.

Hiyoshi, M., Shimazu, H., & Takashima, Y. (1993) A construction tool for context-sensitive guidance systems. *INTERACT '93 and CHI '93 conference companion on Human factors in computing systems* (Amsterdam, The Netherlands). ACM Press, pp. 141–142.

Hone, K. S. & Graham, R. (2000) Towards a tool for the subjective assessment of speech system interfaces (SASSI). *Natural Language Engineering, Best Practice in Spoken Language Dialogue System Engineering*, Special Issue, Vol. 6, Parts 3 & 4, September 2000.

Hone, K. S. & Graham, R. (2001) Subjective assessment of speech-system interface usability. *Proceedings of the 7th European Conference on Speech Communication and Technology, Eurospeech 2001* (Aalborg, Denmark).

Houghton, R. C., Jr. (1984) Online help systems: A conspectus. *Communications of the ACM,* Vol. 27, No. 2, pp. 126–133.

Huang, X., Acero, A., & Hon, H.-W. (2001) *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice Hall PRT.

Jackson, M. D. (1983) Constrained languages need not constrain person/computer interaction. *SIGCHI Bulletin,* Vol. 15, No. 2-3, pp. 18–22.

Jönsson, A. & Dahlbäck, N. (2000) Distilling dialogues – a method using natural dialogue corpora for dialogue systems development. *Proceedings of 6th Applied Natural Language Processing Conference.* (Seattle, WA, USA). pp. 44–51.

Kainulainen, A., Turunen, M., & Hakulinen, J. (2006) An architecture for presenting auditory awareness information in pervasive computing environments. *Proceedings of the 12th International Conference on Auditory Display, ICAD 2006,* (London, UK).

Kamm, C., Litman, D., & Walker, M. A. (1998) From novice to expert: The effect of tutorials on user expertise with spoken dialogue systems. *Proceedings of the 5th International Conference on Spoken Language Processing, ICSLP '98* (Sydney, Australia). ASSTA, pp. 1211–1214.

Kang, H., Plaisant, C., & Shneiderman, B. (2003) New approaches to help users get started with visual interfaces: Multi-layered interfaces and integrated initial guidance. *Proceedings of the 2003 annual national conference on Digital government research* (Boston, MA, USA). Digital Government Research Center, pp. 141−146.

Karat, C.-M., Halverson, C., Horn, D., & Karat, J. (1999) Patterns of entry and correction in large vocabulary continuous speech recognition systems. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '99* (Pittsburgh, PA, USA). ACM Press, pp. 568–575.

Karsenty, L. & Botherel, V. (2005) Transparency strategies to help users handle system errors. *Speech Communication,* Vol. 45, No. 3, pp. 305–324.

Kelleher, C. & Pausch, R. (2005) Stencils-based tutorials: design and evaluation. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI 2005* (Portland, OR, USA). ACM Press, pp. 541–550.

Klaus, H., Klix, H., Sotscheck, J., & Fellbaum, K. (1993) An evaluation system for ascertaining the quality of synthetic speech based on subjective category rating tests. *Proceedings of the 3rd European Conference on Speech Communication and Technology, Eurospeech '93* (Berlin, Germany). ESCA, pp. 1679–1682.

Klemmer, S. R., Sinha, A. K., Chen, J., Landay, J. A., Aboobaker, N., & Wang, A. (2000) Suede: a Wizard of Oz prototyping tool for speech user interfaces. *Proceedings of the 13th annual ACM symposium on User interface software and technology, UIST 2000* (San Diego, CA, USA). ACM Press, pp. 1–10.

Knight, G., Kilis, D., & Cheng, P. C. (1997) An architecture for an integrated active help system. *Proceedings of the 1997 ACM Symposium on Applied Computing* (San Jose, CA, USA). ACM Press, pp. 58–64.

Komatami, K., Ueno, S., Kawahara, T., & Okuno, H. G. (2003) User modeling in spoken dialogue systems for flexible guidance generation. *Proceedings of 8th European Conference on Speech Communication and Technology, Eurospeech 2003* (Geneva, Switzerland). ISCA.

Kotelly, B. (2003) *The Art and Business of Speech Recognition: Creating the Noble Voice*. Addison-Wesley.

Lai, J., (2001) When computers, speak, hear and understand. *Communications of the ACM*, Vol. 44, No. 3, pp. 66–67.

Larsen, L. B. (2003a) Assessment of spoken dialogue system usability – what are we really measuring? *Proceedings of 8th European Conference on Speech Communication and Technology, Eurospeech 2003* (Geneva, Switzerland). ISCA, pp. 1945–1948.

Larsen, L. B. (ed.) (2003b) Evaluation methodologies for spoken and multi modal dialogue systems. *COST278 WG2 and WG3 Report*.

Larson, K. & Mowatt, D. (2003) Speech error correction: The story of the alternates list. *International Journal of Speech Technology,* Vol. 6, No. 2., pp. 183–194.

Lazonder, A. W. & van der Meij, H. (1995) Error-information in tutorial documentation: Supporting users' errors to facilitate initial skill learning. *International Journal of Human–Computer Studies,* Vol. 42, No. 2., pp. 185–206.

Lester, J. C., Converse, S. A., Kahler, S. E., Barlow, S. T., Stone, B. A., & Bhogal, R. S. (1997) The persona effect: Affective impact of animated pedagogical agents. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '97* (Atlanta, GA, USA). ACM Press, pp. 359−366.

Levelt, W. J. M. (1999) Models of word production. *Trends in Cognitive Science,* Vol. 3, No. 6, pp. 223−232.

López-Cózar, R. & Araki, M. (2005) *Spoken, Multilingual and Multimodal Dialogue Systems: Development and Assessment*. John Wiley & Sons, Ltd.

Luperfoy, S. & Duff, D. (1997) Questions regarding repair dialogs. *CHI '97 Workshop: Speech User Interface Design Challenges*.

Lyons, K., Skeels, C., Starner, T., Snoeck, C. M., Wong, B. A., & Ashbrook, D. (2004) Augmenting conversations using dual-purpose speech. *Proceedings of the 17th annual ACM symposium on User interface software and technology, UIST'04* (Santa Fe, NM, USA). ACM Press, pp. 237−246.

Mackay, W. E. (2001) Does tutoring really have to be intelligent? *CHI '01 extended abstracts on Human factors in computing systems* (Seattle, WA, USA). ACM Press, pp. 265–266.

Maes, A., Goutier, S., & van der Linden, E.-J. (1992) Online reading and offline tradition. *Proceedings of the 10th annual international conference on Systems documentation, SIGDOC '92* (Ottawa, Ontario, Canada). ACM Press, pp. 175–182.

Major, J. H. (1985) Putting it all together: A well-designed user's manual. *Proceedings of the 13th annual ACM SIGUCCS conference on User services* (Toledo, OH, USA). ACM Press, pp. 69–76.

Mallen, C. (1996) Designing intelligent help for information processing systems. *International Journal of Human–Computer Studies,* Vol. 45, No. 3, pp. 349–377.

Manaris, B., Macgyvers, V., & Lagoudakis, M. (2002) A listening keyboard for users with motor impairments – a usability study. *International Journal of Speech Technology*, Vol. 5, No. 4, pp. 371–388.

Martin, D. L., Cheyer, A. J., & Moran, D. B. (1999) The Open Agent Architecture: A framework for building distributed software systems. *Applied Artificial Intelligence: An International Journal,* Vol. 12, No. 1–2, pp. 91–128.

Mayhew, D. J. & Mantei, M. (1994) A basic framework for cost-justifying usability engineering. In R. G. Bias & D. J. Mayhew (eds.), *Cost-Justifying Usability*, Academic Press, London, UK.

McKendree, J. (1986) Advising roles of a computer consultant. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '86* (Boston, MA, USA). ACM Press, pp. 35–40.

McTear, M. F. (2004) *Spoken Dialogue Technology: Toward the Conversational User Interface*. Springer-Verlag.

Mehlenbacher, B. (1993) Software usability: choosing appropriate methods for evaluating online systems and documentation. *Proceedings of the 11th annual international conference on Systems documentation, SIGDOC '93* (Waterloo, Ontario, Canada). ACM Press, pp. 209–222.

Miettinen, M. & Toivanen, J. (eds.) (2001) *Puheentutkimuksen resurssit suomessa*. CSC - Scientific Computing Ltd, Espoo, Finland.

Minker, W., Haiber, U., Heisterkamp, P., & Scheible, S. (2003) Intelligent dialog overcomes speech technology limitations: The SENECa example.

Proceedings of the 8th International Conference on Intelligent User Interfaces, IUI'03 (Miami, FL, USA). ACM Press, pp. 267–269.

Moore, R. K. (2005) Research Challenges in the Automation of Spoken Language Interaction. *Proceedings of ISCA/COST ASIDE 2005 workshop on Applied Spoken Language Interaction in Distributed Environments* (Aalborg, Denmark).

Moreno, L., Gonzáles, C., Nunõz, V., Estévez, J., Aguilar, R., Sánchez, J., Sigut, J., & Piñeiro, J. (2002) Integrating multimedia technology, knowledge based system, and speech processing for the diagnostic and treatment of developmental dyslexia. *Proceedings of Intelligent Tutoring Systems: 6th International Conference, ITS 2002* (Biarritz, France and San Sebastian, Spain). Springer, pp. 168–177.

Möller, S. (2002) A new taxonomy for the quality of telephone services based on spoken dialogue systems. *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue* (Philadelphia, PA, USA). ACL.

Möller, S. (2005) Parameters for quantifying the interaction with spoken dialogue telephone services. *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue* (Lisbon, Portugal). ACL, pp. 166–177.

Möller, S. & Skowronek, J. (2003) Quantifying the impact of system characteristics on perceived quality dimensions of a spoken dialogue service. *Proceedings of the 8th European Conference on Speech Communication and Technology, Eurospeech 2003* (Geneva, Switzerland). ISCA, pp. 1953–1956.

Nakano, M., Dohsaka, K., Miyazaki, N., Hirasawa, J., Tamoto, M., Kawamori, M., Sugiyama, A., & Kawabata, T. (1999) Handling rich turn-taking in spoken dialogue systems. *Proceedings of the 6th European Conference on Speech Communication and Technology, Eurospeech-99* (Budapest, Hungary). ESCA, pp. 1167–1170.

Nakatani, L. H., Egan, D. E., Ruedisueli, L. W., Hawley, P. M., & Lewart, D. K. (1986) TNT: A talking tutor 'n' trainer for teaching the use of interactive computer systems. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '86* (Boston, MA, USA). ACM Press, pp. 29–34.

Nass, C., Robles, E., Heenan, C., Bienstock, H., & Treinen, M. (2003) Speech-based disclosure systems: Effects of modality, gender of prompt, and gender of user. *International Journal of Speech Technology,* Vol. 6, No. 2, pp. 113–121.

Nielsen, J. (1993). *Usability Engineering*. AP Professional, Boston, MA.

Novick, D. G., & Ward, K. (2003) An interaction initiative model for documentation. *Proceedings of the 21st annual international conference on Documentation, SIGDOC '03* (San Francisco, CA, USA). ACM Press.

Oppermann, D., Schiel, F., Steininger, S., & Beringer, N. (2001) Off-talk – a problem for human–machine interaction? *Proceedings of the 7th European Conference on Speech Communication and Technology. Eurospeech 2001* (Aalborg, Denmark).

Oviatt, S. (1996a) Multimodal interfaces for dynamic interactive maps. *Proceedings of the SIGCHI conference on Human factors in computing systems: common ground, CHI '96* (Vancouver, British Columbia, Canada). ACM Press, pp. 95–102.

Oviatt, S. (1996b) User-centered modeling for spoken language and multimodal interfaces. *IEEE Multimedia,* Vol. 3, No. 4, pp. 26–35.

Palmiter, S. & Elkerton, J. (1991) An evaluation of animated demonstrations for learning computer-based tasks. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '91* (New Orleans, LA, USA). ACM Press, pp. 257−263.

Patrick, A. & McGurgan, A. (1993) One proven methodology for designing robust online help systems. *Proceedings of the 11th Annual International Conference on Systems Documentation* (Waterloo, Ontario, Canada). ACM Press, pp. 223–232.

Pearson, J., Hu, J., Branigan, H. P., Pickering, M. J., & Nass, C. I. (2006) Adaptive language behavior in HCI: how expectations and beliefs about a system affect users' word choice. *Proceedings of the SIGCHI conference on Human Factors in computing systems, CHI 2006* (Montréal, Québec, Canada). ACM Press, pp. 1177–1180.

Perkins, D. N. & Salomon, G. (1992) Transfer of learning. *The International Encyclopedia of Education*, second edition. Pergamon Press, Oxford.

Pieraccini, R., Dayanidhi, K., Bloom, J., Dahan, J.-G., Phillips, M., Goodman, B. R., & Prasad, K. V. Multimodal conversational systems for automobiles. *Communications of the ACM,* Vol. 47, No. 1, pp. 47–49.

Pollack, M. E. (1985) Information sought and information provided: An empirical study of user/expert dialogues. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '85* (San Francisco, CA, USA). ACM Press, pp. 155–159.

Purchase, H. C. & Worrill, J. (2002) An empirical study of on-line help design: Features and principles. *International Journal of Human–Computer Studies,* Vol. 56, No. 5, pp. 539–567.

Raman, T. V. (1996) Emacspeak – direct speech access. *Proceedings of the 2nd annual ACM conference on Assistive technologies, ASSETS '96* (Vancouver, British Columbia, Canada). ACM Press, pp. 32–36.

Raynal, M. & Serrurier, M. (2002) CYNTHIA: An HTML browser for visually handicapped people. *Computers Helping People with Special Needs, 8th International Conference*. ICCHP, pp. 353–359.

Reeves, B. & Nass, C. (1996) *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*. Cambridge University Press.

Resnick, P. & Virzi, R. A. (1995) Relief from the audio interface blues: Expanding the spectrum of menu, list and form styles. *ACM Transactions on Computer–Human Interaction*, Vol. 2, No. 2, pp. 145–176.

Rettig, M. (1991) Nobody reads documentation. *Communications of the ACM,* Vol. 34, No. 7, pp. 19–24.

Rich, C., Sidner, C., Lesh, N., Garland, A., Booth, S., & Chimani, M. (2005) DiamondHelp: A collaborative interface framework for networked home appliances. *Proceedings of* 25th *IEEE International Conference on Distributed Computing Systems Workshops*. IEEE Computer Society, pp. 514–519.

Rosenfeld, R., Olsen, D., & Rudnicky, A. (2001) Universal Speech Interfaces. *Interactions*, Vol. 8, No. 6, pp. 34--44.

Sacks, H., Schegloff, E., & Jefferson, G. (1974) A simplest systematics for the organization of turn-taking for conversation. *Language,* Vol. 50, No. 4, pp. 696–735.

Sanders, G. A. & Le, A. N. (2004) Effects of speech recognition accuracy on the performance of DARPA communicator spoken dialogue systems. *International Journal of Speech Technology*, Vol. 7, No. 4, pp. 293–309.

Schmandt, C. (1994) *Voice Communication with Computers: Conversational Systems*. Van Nostrand Reinhold.

Schröder, M. (2001) Emotional speech synthesis: A review. *Proceedings of the 7th European Conference on Speech Communication and Technology, Eurospeech 2001* (Aalborg, Denmark). ISCA.

Seneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P., & Zue, V. (1998) Galaxy-II: A reference architecture for conversational system development. *Proceedings of 5th International Conference on Spoken Language Processing, ICSLP 1998* (Sydney, Australia). ASSTA.

Shneiderman, B. (2000) The limits of speech recognition. *Communications of the ACM,* Vol. 43, No. 9, pp. 63–65.

Sidner, C. L. & Forlines, C. (2002) Subset languages for conversing with collaborative interface agents. *Proceedings of the 7th International Conference on Spoken Language Processing, ICSLP 2002* (Denver, CO, USA). ISCA.

Skantze, G. (2003) Exploring human error handling strategies: implications for spoken dialogue systems. *Proceedings of the ISCA Workshop on Error Handling in Spoken Dialogue Systems* (Chateau-d'Oex-Vaud, Switzerland).

Sliski, T. J., Billmers, M. P., Clarke, L. A., & Osterweil, L. J. (2001) An architecture for flexible, evolvable process-driven user-guidance environments. *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Vienna, Austria). ACM Press, pp. 33–43.

Smeele, P. M. T. & Waals, J. A. J. S. (2003) Evaluation of a speech-driven telephone information service using the PARADISE framework: A closer look at subjective measures. *Proceedings of 8th European Conference on Speech Communication and Technology, Eurospeech 2003* (Geneva, Switzerland). ISCA, pp. 1949–1952.

SpeechWorks. Assessing text-to-speech systems quality. White paper. SpeechWorks International, Inc.

Spiliotopoulos, D., Xydas, G., & Kouroupetroglou, G. (2005) Diction based prosody modeling in table-to-speech synthesis. *Proceedings of Text, Speech and Dialogue, TSD2005* (Karlovy Vary, Czech Republic). Springer-Verlag, pp. 294–301.

Stanton, N. (1993) Speech-based alarm displays. In C. Barber and J. Noyes (eds.), *Interactive Speech Technology: Human Factors Issues in the Application of Speech Input/Output to Computers*. Taylor & Francis.

Strik, H., Russel, A., van den Heuvel, H., Cucchiarini, C., & Bowes, L. (1996) Localizing an automatic inquiry system for public transport information. *Proceedings of the 4th International Conference on Spoken Language Processing, ICSLP'96* (Philadelphia, PA, USA). ESCA.

Sturm, J., Bakx, I., Cranen, B., & Terken, J. (2003) Comparing the usability of a user driven and mixed initiative multimodal dialogue systems for train timetable information. *Proceedings of the 8th European Conference on Speech Communication and Technology, Eurospeech 2003* (Geneva, Switzerland). ISCA.

Suhm, B. (2003a) Improving speech interface design by understanding limitations of speech interfaces. *Proceedings of AVIOS Speech Technology Symposium conference, AVIOS 2003*. TMA Associates.

Suhm, B. (2003b) Towards best practices for speech user interface design. *Proceedings of the 8th European Conference on Speech Communication and Technology Eurospeech 2003* (Geneva, Switzerland). ISCA, pp. 2217–2220.

Swartz, L. (2003) *Why People Hate the Paperclip: Labels, Appearance, Behavior, and Social Responses to User Interface Agents.* Honors Thesis for Symbolic Systems Program, Stanford University.

Sweetser, P. & Wyeth, P. (2005) GameFlow: A model for evaluating player enjoyment in games. *ACM Computers in Entertainment,* Vol. 3, No. 3, pp. 1–24.

Terken, J. & te Riele, S. (2001) Supporting the constructions of a user model in speech-only interfaces by adding multi-modality. *Proceedings of the 7th European Conference on Speech Communication and Technology, Eurospeech 2001 Scandinavia* (Aalborg, Denmark). ISCA, pp. 2177–2180.

Tomko, S. & Rosenfeld, R. (2006) Shaping user input in speech graffiti: a first pass. *CHI '06 extended abstracts on Human factors in computing systems* (Montréal, Québec, Canada). ACM Press, pp. 1438–1444.

Turunen, M. (2004) *Jaspis - A Spoken Dialogue Architecture and its Applications*. Ph.D. Thesis, University of Tampere, 2004.

Turunen, M. & Hakulinen, J. (2000a) Jaspis – a framework for multilingual adaptive speech applications. *Proceedings of the 6th International Conference on Spoken Language Processing, ICSLP 2000* (Beijing, China). ISCA, pp. 719–722.

Turunen, M. & Hakulinen, J. (2000b) Mailman – a multilingual speech-only e-mail client based on an adaptive speech application framework. *Proceedings of Workshop on Multi-Lingual Speech Communication, MSC 2000* (Kyoto, Japan). pp. 7–12.

Turunen, M. & Hakulinen, J. (2001) Agent-based error handling in spoken dialogue systems. *Proceedings of the 7th European Conference on Speech Communication and Technology, Eurospeech 2001* (Aalborg, Denmark). ISCA, pp. 2189–2192.

Turunen, M., Hakulinen, J., Salonen, E.-P., Kainulainen, A., & Helin, L. (2005) Spoken and multimodal bus timetable systems: Design, development and evaluation. *Proceedings of the 10th International Conference on Speech and Computer, SPECOM 2005* (Patras, Greece). pp. 389–392.

van der Meij, H. (1992) A critical assessment of the minimalist approach to documentation. *Proceedings of the 10th Annual International Conference on Systems Documentation* (Ottawa, Ontario, Canada). ACM Press, pp. 7–17.

Venkatagiri, H. S. (2003) Segmental intelligibility of four currently used text-to-speech synthesis methods. *Journal of Acoustic Society of America*, Vol. 113, No. 4, pp. 2095–2104.

Vredenburg, K., Mao, J.-Y., Smith, P. W., & Carey, T. (2002) A survey of user-centered design practice. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI 2002* (Minneapolis, MN, USA). ACM Press, pp. 471−478.

Walker, M. A. & Whittaker, S. (1990) Mixed initiative in dialogue: An investigation into discourse segmentation. *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics, ACL 1990* (Pittsburgh, PA, USA). ACL, pp. 70–78.

Walker, M. A., Hirschman, L., & Aberdeen, J. (2000) Evaluation for DARPA communicator spoken dialogue systems. *Proceedings of the 2nd International Conference on Language Resources and Evaluation, LREC 2000* (Athens, Greece).

Walker, M. A., Kamm, C. A., & Boland, J. (2000) Developing and testing general models of spoken dialogue system performance. *Proceedings of the 2nd International Conference on Language Resources & Evaluation, LREC 2000* (Athens, Greece).

Walker, M. A., Passonneau, R., & Boland, J. E. (2001) Quantitative and qualitative evaluation of DARPA communicator spoken dialogue systems. *Proceedings of the 39th Meeting of the Association of Computational Linguistics, ACL 2001* (Toulouse, France). ACL, pp. 515–522.

Walker, M. A., Litman, D. J., Kamm, C. A., & Abella, A. (1997) PARADISE: A framework for evaluating spoken dialogue agents. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, ACL 1997* (Madrid, Spain). ACL, pp. 271–280.

Walker, M. A., Litman, D. J., Kamm, C. A., & Abella, A. (1998) Evaluating spoken dialogue agents with PARADISE: Two case studies. *Computer Speech and Language,* Vol. 12, No. 4, pp. 317–347.

Ward, K. & Heeman, P. A. (2000) Acknowledgments in Human-Computer Interaction. *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics, NAACL 2000* (Seattle, WA, USA). ACL, pp. 280–287.

Weegels, M. F. (2000) Users' conceptions of voice-operated information services. *International Journal of Speech Technology,* Vol. 3, No. 2, pp. 75–82.

Wiedenbeck, S. & Zila, P. L. (1997) Hands-on practice in learning to use software: A comparison of exercise, exploration, and combined formats. *ACM Transactions on Computer–Human Interaction,* Vol. 4, No. 2, pp. 169–196.

Wiedenbeck, S., Zila, P. L., & McConnell, D. S. (1995) End-user training: An empirical study comparing on-line practice methods. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '95* (Denver, CO, USA). ACM Press, pp. 74–81.

Wilensky, R., Chin, D. N., Luria, M., Martin, J., Mayfield, J., & Wu, D. (1988) The Berkeley UNIX consultant project. *Artificial Intelligence Review,* Vol. 14, No. 1–2, pp. 43–88.

Williams, T. R. & Farkas, D. K. (1992) Minimalism reconsidered: Should we design documentation for exploratory learning? *SIGCHI Bulleting,* Vol. 24, No. 2, pp. 41−50.

Williams, J. D., Shaw, A. T., Piano, L., & Abt, M. (2003) Preference, perception, and task completion of open, menu-based, and directed prompts for call routing: A case study. *Proceedings of the 8th European Conference on Speech Communication and Technology, Eurospeech 2003* (Geneva, Switzerland). ISCA.

Witt, S. M. & Williams, J. D. (2003) Two studies of open, vs. directed dialog strategies in spoken dialog systems. *Proceedings of 8th European Conference on Speech Communication and Technology, Eurospeech 2003* (Geneva, Switzerland). ISCA.

W3C (2004a) Voice Extensible Markup Language (VoiceXML) Version 2.0, W3C Recommendation, 16 March 2004: http://www.w3.org/TR /2004/REC-voicexml20-20040316/.

W3C (2004b) Speech Recognition Grammar Specification Version 1.0, W3C Recommendation, 16 March 2004: http://www.w3.org/TR/2004/REC-speech-grammar-20040316/.

Yankelovich, N. (1996) How do users know what to say? *Interactions*, Vol. 3, No. 6, pp. 32–43.

Yesilada, Y., Stevens, R., Goble, C., & Hussein, S. (2003) Rendering tables in audio: The interaction of structure and reading styles. *Proceedings of the 6th International ACM SIGACCESS Conference on Computers and Accessibility, ASSETS '04* (Atlanta, GA, USA). ACM Press, pp. 16–23.

Zajicek, M., Powell, C., & Reeves, C. (1998) A Web navigation tool for the blind. *Proceedings of the 3rd international ACM conference on Assistive technologies, ASSETS '98* (Marina del Rey, CA, USA). ACM Press, pp. 204–206.

Paper I

Markku Turunen, Jaakko Hakulinen, Kari-Jouko Räihä, Esa-Pekka Salonen, Anssi Kainulainen, & Perttu Prusi. An architecture and applications for speech-based accessibility systems. *IBM Systems Journal*, 44(3), pages 485-504.

An architecture and applications for speech-based accessibility systems

M. Turunen
J. Hakulinen
K.-J. Räihä
E.-P. Salonen
A. Kainulainen
P. Prusi

Speech can be an efficient and natural means for communication between humans and computers. The development of speech applications requires techniques, methodology, and development tools capable of flexible and adaptive interaction, taking into account the needs of different users and different environments. In this paper, we discuss how the needs of different user groups can be supported by using novel architectural solutions. We present the Jaspis speech application architecture, which introduces a new paradigm for adaptive applications and has been released as open-source software to assist in practical application development. To illustrate how the architecture supports adaptive interaction and accessibility, we present several applications that are based on the Jaspis architecture, including multilingual e-mail systems, timetable systems, and guidance systems.

INTRODUCTION

Accessibility is often mentioned as one of the major motivations for the development of speech applications. For example, there has been much work in speech-based and auditory interfaces to allow visually impaired users to access existing graphical interfaces.[1] In general, multiple modalities have been used to make human-computer interaction accessible for people with disabilities.[2]

In the ideal case, applications should take the needs of different users and usage conditions into account in the first place: interaction should be adapted to each usage situation. The goal of this approach is universal access to services. Current application development architectures tend to lack the flexibility necessary to adapt to a variety of users and usage conditions. In this paper, we present a system architecture capable of supporting the development of accessible interactive systems.

SPEECH SYSTEM ARCHITECTURES

A software architecture defines applications in terms of components and interactions among them.[3] A framework suitable for practical speech applications must provide components for a variety of application requirements, including dialog management, speech recognition, and natural language processing. High-level components (modules) usually contain multiple subcomponents having their own

■ Applications should take the needs of different users and usage conditions into account ■

internal organization and relations. The challenge lies in finding ways for all of these components to be organized and selecting the functionality that the underlying system architecture should offer. This includes the development of principles for interaction and management of information flow among the system components.

When we consider speech architectures from the human-computer interaction point of view, an interesting issue is how the system can be made to support more intuitive and natural speech-based interaction to allow universal access to services. The needs of different user groups vary considerably. Natural interaction requires flexible interaction models supported by the system architecture.[4]

In most speech systems, components are structured in a pipeline fashion; that is, they are executed in a sequential order, although this kind of "pipes-and-filters" model is considered suboptimal for interactive systems.[3] In order to facilitate the development of advanced speech applications, we need more advanced techniques, models, methodology, and tools. In particular, we need system frameworks that support the requirements mentioned because, as with all technical variation, the key issue is the integration of components into a working system.[5]

Earlier work in advanced speech system architectures includes client-server approaches and systems based on agent architectures. Probably the best-known speech-specific client-server architecture is Galaxy-II.[6] The Open Agent Architecture[7] is a general agent architecture that has been used in the construction of many speech applications. These architectures offer the necessary infrastructure components for applications, but they do not support human-computer interaction tasks or adaptation in any particular manner.

A great deal of work has been done in the field of dialog management. Three particularly interesting recent examples include the agenda-based dialog management architecture[8] and its RavenClaw extension,[9] Queen's Communicator,[10] and SesaME.[11] The purpose of these approaches is not to provide a complete speech architecture but instead, a model for dialog management.

The Jaspis architecture addresses many of the same features as the dialog management architectures mentioned but aims for general applicability in a variety of task settings, one of which is dialog management. It introduces a new paradigm for interactive systems that focuses on speech-based applications. In our previous papers[12] we have presented technical and functional aspects of the architecture. In this paper, the architectural principles—and in particular their novel support for adaptive interaction—are described in the context of human-computer interaction. We demonstrate how it is possible to construct highly adaptive systems suitable for different user groups and to support accessibility by using the Jaspis architecture and its principles.

The remainder of the paper is organized as follows. In the next section we introduce architectural foundations for adaptive human-computer interaction. The Jaspis architecture and its novel interaction paradigm based on *agents, managers* and *evaluators* is introduced with examples. After the architecture presentation, we introduce several Jaspis-based applications for various domains. Examples of multilingual e-mail systems, timetable services, and pervasive computing applications are given. Special focus is given to interaction-level issues, such as error management, help, and guidance. In the next section we report experiences and results from user-centered design, "Wizard of Oz"** studies, and evaluations of Jaspis applications. Accessibility issues, such as those raised in design sessions with users who are visually impaired, are discussed. The paper closes with conclusions and discussion.

JASPIS ARCHITECTURE

Jaspis is a general speech-application architecture designed for the challenges of advanced speech applications, especially adaptive and multilingual speech applications. It provides support for human-computer interaction tasks, such as error handling, "Wizard of Oz" studies (i.e., those in which some parts of the system are simulated with a human operator), and corpora collection. While Jaspis is a general conceptual architecture, it is also a concrete framework which provides components for application development. In this section, we present the principles of the Jaspis architecture, focusing on human-computer interaction tasks, in particular on dialog management, output generation, and input management tasks. In addition, application development aspects are briefly discussed.

Architecture requirements

In order to support more flexible interaction, we have identified requirements for speech application architectures. First, speech applications need *adaptive interaction methods* in all system modules. For example, outputs and inputs should be tailored to the language of the users, and dialog management should adapt to the situation at hand. Second, systems should be *modular* because modular components support reusability, are easy to maintain and extend, can be distributed efficiently, and make adaptivity easier to achieve. The other requirements concern *collaborative and iterative* application design and development, the need for an *extensible and practical* infrastructure for application development, and support for standards. These principles are motivated by the technology, human-computer interaction, and application development viewpoints, all of which should be taken into account. For a more comprehensive description of the Jaspis architecture requirements see Reference 13.

Architecture overview

In order to support the architectural requirements mentioned, the Jaspis architecture uses a *modular and distributed system structure, an adaptive interaction coordination model* and *a shared system context*. These form the basic infrastructure on which other features and components of the system are based.

Figure 1 presents a typical Jaspis-based system setup. The top-level structure of the system is based on *managers*, which are connected to the *central manager* with a star topology. Communication between components is organized according to the client-server paradigm. Local subsystems are located inside the system modules. The interaction coordination model of the Jaspis architecture is based on the agents-managers-evaluators paradigm. *Agents* are interaction components which imple-

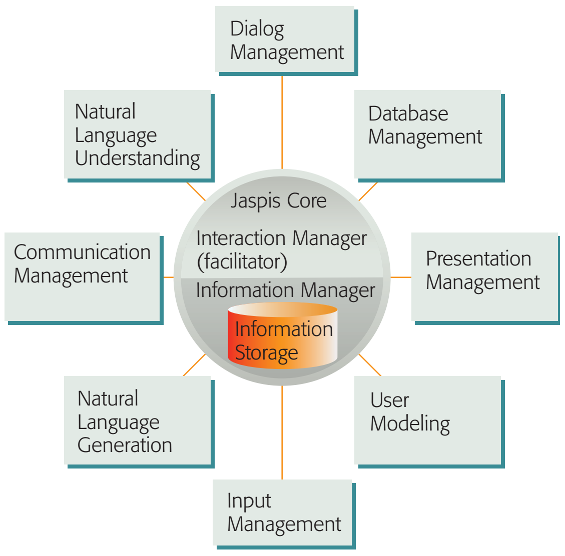


Figure 1
Overall system structure for Jaspis applications

ment different interaction techniques, such as speech output presentations and dialog decisions. *Evaluators* are used to evaluate different aspects of the agents, in order to determine how suitable the agents are for different tasks. *Managers* are used to coordinate these components. All information in Jaspis-based systems is stored in the shared information storage. All components of the system may access the content of the information storage by using the information manager. These are the key features enabling architecture-level adaptation.

Shared information management

Information management is a crucial element of adaptive, modular, and distributed applications. The repository approach (i.e., using "blackboards" and databases) provides several advantages for adaptive and distributed applications. The term "blackboards," in the context of speech applications, refers to shared information resources, or specifically, a shared knowledge base. Most importantly, the repository approach allows the use of shared information by all system components. The main drawback of this approach is the lack of control.[7] In Jaspis, the coordination and control are performed by a separate component (the interaction manager) to achieve architecture-level coordination.

The Jaspis information management architecture consists of four layers. In this way, the actual storage (the *information storage*), the application interface (the *information manager*), and the communication interfaces (the *information access protocol* and *communication protocols*) are separated to maximize flexibility. In this section we focus on information storing and application layers, omitting the communication layers.

The information storage holds all the shared system data, that is, the shared system context. The Jaspis architecture assumes that individual components do not store any high-level information inside them, but instead use the information storage for that purpose. This makes the interaction components *stateless*, and the system is able to adapt to each interaction by choosing proper components for each situation. To make this possible, the system assumes that every component updates its knowledge from the information storage when activated and writes modified information back to the information storage when deactivated. In the ideal case, the shared data should be represented at a conceptual level such that it can be used by other components as well. This is one of the main features used to adapt the systems for different users.

The reference implementation of the information storage uses XML (Extensible Markup Language) for its internal information representation. The content or the structure of information inside the information storage is not defined by the system architecture because this is specific to the particular application and domain. The definition of the shared knowledge is an important phase of the application development process.

The programming interface for the information storage is straightforward: it takes XML requests and produces XML results. The information storage offers only the minimal set of operations needed to manipulate its contents. In addition to the shared information storage, direct information exchange between certain I/O components is supported for efficiency reasons. Most notably, the raw audio streams should be passed between components in a cost-efficient way to minimize system overhead and delays.

The information manager provides an application interface to the information storage. It uses the information access protocol to access the information storage and provides a programming interface for other components to access the shared system context. For example, system inputs and outputs may be modified by their own set of methods. Application developers may write new, application-specific methods when needed.

Flexible interaction management

The interaction management model of Jaspis is focused on the key design principles of the architecture: adaptivity and modularity. Interaction management in this context means both the overall coordination of system components and the coordination of those components that implement interaction techniques to be used in human-computer interaction tasks. In practice, this means input, output, and dialog management components in their various forms.

Interaction techniques are implemented by agents, which are software components specialized for certain tasks. Evaluators are used to make selections among different agents. Managers are used to coordinate agents and evaluators. Components specialized for related tasks are organized into modules. An overview of the interaction management model is presented in ***Figure 2***.

As illustrated in the figure, each system module contains one local manager and several agents and evaluators. It is up to the local managers to decide which agents are used in different situations. Instead of centralizing this decision (by assigning it to the managers), evaluators are used to evaluate agents and their suitability for different tasks. Thus, there is no central component which makes these decisions. This makes it possible to construct highly adaptive and modular systems because all functionality is divided into specialized components that have no predefined execution order and relations among them. The principles governing how managers, agents, and evaluators are used are presented next in more detail.

Managers: coordination

The interaction manager is a central component in Jaspis-based systems. It manages other components and is responsible for the overall coordination of the interaction. The interaction manager is similar to some central components found in other speech architectures. Such components include the hub in the Galaxy-II architecture[6] and the Facilitator in the Open Agent Architecture.[7] Similar components can

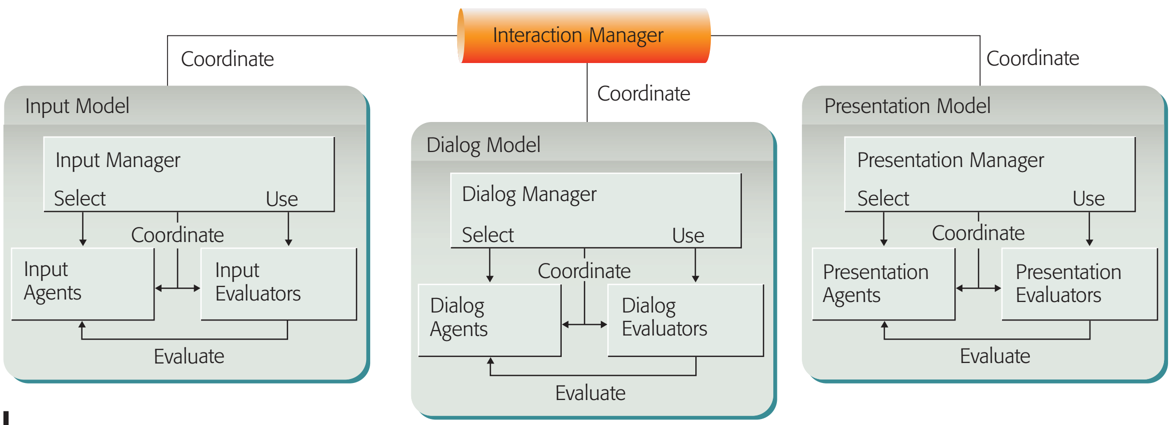


Figure 2
General interaction management architecture

also be found in other distributed architectures. The main differences include the division of labor between the information manager and the interaction manager and the fact that Jaspis is a layered (hierarchical) architecture.

Coordination is one of the main issues in agent-based systems.[7,14] In order to coordinate interaction in a flexible way, it is better to use decentralized control logic; managers and their components may be more autonomous, and the system may adapt to interaction when needed. This is crucial in applications that are targeted for various user groups, and in particular, when applications need to be modified afterwards, as is the case in many accessibility systems which need to be customized for different users. In Jaspis, local managers are quite independent. The interaction manager does not know the internal structure of modules, nor does it coordinate interaction inside modules. Layered systems are easier to maintain, and they can be more efficient. This is because local managers need to coordinate only their own components. Evaluations in particular are more efficient when only those agents that belong to the same module are evaluated.

Jaspis uses a "round robin" interaction management approach, in which one manager is active at a time. The interaction manager enables the managers to take turns, based on a prioritized list. When the option to become active is offered to a manager, it will check if it is able to handle the situation by using its own reasoning algorithm or by consulting evaluators.

In order to support application development, the architecture includes a general manager class that can be customized for different purposes. The general manager asks evaluators to assign a score for each agent in the module. If there is an agent that receives a score above zero, the manager becomes active and activates the agent with the highest score. In the basic algorithm, managers multiply evaluation scores, but more elaborate methods may be used as well.[13]

In addition to the default behavior presented, local managers are able to extend this functionality, or in general, implement their own functionality. For example, some of the Jaspis reference implementation managers (e.g., the presentation manager and the communication manager) have extended the basic functionality. This is because they have different needs when using agents. The agents–managers–evaluators paradigm is used in all system modules, but in various ways.

Agents: interaction tasks

Agents in the Jaspis architecture are software components that handle various interaction tasks. Agents are often used to handle tasks that are fairly complex. For example, in the WITAS (Wallenberg Laboratory for Research on Information Technology and Autonomous Systems) multimodal dialog system, there are six agents.[15] Jaspis has been designed with *compact agents* in mind. Because agents are meant to handle single, well-defined tasks, they are fairly small, and typical systems contain many of them. For example, in the AthosMail application (presented in the following section), there are over

50 basic dialog agents alone (e.g., agents for reading e-mail messages, navigating among folders and messages, and providing context-sensitive help).

Typical agents in Jaspis-based systems are *specialized for certain tasks*, which may be domain-specific or general interaction management tasks. Agents may implement interaction techniques for tasks such as generation of welcome prompts or handling of speech inputs. Some of the agents may be application-independent, while some may be closely tied to the application domain. Application-independent behavior is usually preferred so that components may be used across applications. Usually, application-specific components are encapsulated in their own agents, and inheritance is used to maximize runtime efficiency.[16–18]

Specialized agents make it possible to implement modular, reusable, and extensible interaction components that are easy to implement and maintain. For example, we have constructed general interaction techniques, such as error correction methods, to take care of error situations in speech applications.[19] Another example is that of tutoring agents, which provide guidance for the users in a real context.[20]

In addition to the collaborative approach, where different agents implement different functionality, multiple agents may be specialized for the same tasks with varying behavior. In other words, agents may provide alternative solutions to the same problems. Based on our experiences, we have concluded that different user groups prefer different types of interaction (described in the following sections in more detail). Using modular agents, we can support different interaction strategies within an application and adapt the interaction dynamically to the user and the situation. For example, we may have different agents to take care of speech outputs for various user groups and languages. We have constructed specialized presentation agents for different languages and with varying verbosity levels without modifying the original agents.

Jaspis agents are *stateless*, as stated previously, but they are *autonomous* in the sense that they know when they are able to act. Each agent has a self-evaluation method that checks the current situation (for example, the dialog context) and gives a local estimate that defines how well it can handle the situation at hand. The self-evaluation method is only one part of the agent selection process. All dependencies among agents, including the overall suitability of competing agents for different situations, are modeled using evaluators.

Evaluators: system-level adaptation

Evaluators are the key concept in making applications adaptive and interaction flexible. They determine which agents should be selected for different interaction tasks. It is up to evaluators to compare different agents and their capabilities and assign an evaluation score for every agent in any given situation. Like agents, evaluators are specialized for certain tasks. In practice, this means that different evaluators evaluate different aspects of agents from different system viewpoints. For example, an evaluator may use the dialog history to determine which dialog strategy should be used (i.e., which kind of dialog agent should be selected), and another evaluator may use the user model to select verbose or brief system output formats.

When Jaspis evaluators are employed at the human-computer interaction level, they can monitor interactions and give guidance as to how an interaction is progressing and how it should continue. One example is dialog strategy: if the interaction is not going smoothly when the dialog strategy is based on user initiative, a specialized dialog evaluator may give better scores for dialog agents based on system initiative. Similarly, if the user has problems, presentation evaluators may prefer presentation agents that use more detailed and helpful prompts. Concrete examples are presented in the following sections.

Evaluation is applied in the system modules. When one of the agents inside a module is to be selected, each evaluator in the module assigns a score to every agent in the module. These scores are then multiplied by the local manager. The manager assigns the final score, a suitability factor, to every agent (see Reference 13 for a more comprehensive description). It is noteworthy that there is no single evaluator, nor any single component in general, which selects agents for each situation; instead, the selection is always both dynamic and distributed. This makes it possible to keep the program control and interaction flow highly dynamic and adaptive on the architectural level. This is a major improvement to the "black box" type of adaptivity that most systems offer (see, for example, Reference 21).

Evaluators may use different strategies and techniques to evaluate agents. They may use information such as the current dialog context, user model, and interaction history. Presentation evaluators, for example, may use dialog history to determine (e.g., using heuristic rules) which confirmation agent is most suitable for the current dialog state. Other evaluators may use methods from machine learning, for example, to evaluate agents and their usefulness in a given context.

Evaluators know the properties of the agents, which are expressed in the form of attributes. Attributes are presented in XML-based configuration files and can be configured with domain-specific parameters. In more complex systems, the parameters can be learned automatically. To be generic, the architecture does not force application developers to follow any predefined approach to this process. In conventional dialog systems, all dialog decisions are made by the dialog manager. Jaspis evaluators allow more flexible interaction management to take place because the reasoning is both distributed and dynamic. Evaluators may nevertheless be used for more global evaluation. This can be done with special evaluators to monitor interactions and to favor consistent interaction. For example, the evaluators may favor agents which use a language similar to that used in previous dialogs for inputs and outputs and which follow the same dialog strategy.

In the AthosMail system (to be presented in the following sections), the presentation agents are evaluated with five different evaluators. Each evaluator uses specific information to assign a score to each agent. The presentation evaluators use the following information: (1) dialog data, (2) language of the output, (3) user expertise, (4) characteristics of the mailbox, and (5) self-evaluation results of the agents. Evaluation scores from these five evaluators are combined, and the agent with the highest overall score is selected to generate the output.

We next present how the general agents–managers–evaluators paradigm is applied to human-computer interaction tasks, including dialog, presentation, and input management tasks.

Human-computer interaction management

The Jaspis architecture provides several general interaction modules that can be customized for specific tasks and applications. These are input, dialog, presentation, and communication modules. These modules contain the components used to implement interaction techniques for human-computer interaction tasks. All of these modules are based on the general agents–managers–evaluators paradigm but use it in slightly different ways. Developers are free to introduce additional modules when needed (e.g., for user modeling or handling modality-specific issues).

■ Natural interaction requires flexible interaction models supported by the system architecture ■

Dialog and error management

The task of the dialog management module is to update the dialog state, that is, to move the dialog from one state to another. This abstract task representation may be modeled in various ways and should not be restricted to any single dialog control model, such as the finite-state model or the form-based model.

Like other agents, dialog agents are specialized for different dialog tasks or provide alternative solutions for the same dialog tasks. Unlike many other agents, such as presentation agents and input agents, in most cases there is usually only one dialog agent active at any given time. The dialog control model is one way to categorize dialog agents. For example, in the *state-based* dialog control model, each dialog state can correspond to one dialog agent, whereas in the *concept-based* model, each concept may form its own agent. In the *form-based* dialog model, every form field may correspond with a dialog agent. In addition, it is possible to use multiple dialog control models, such as those based on state machines and forms, in the same application. The combination of different approaches is especially useful when subdialogs are implemented with different dialog control models.[17]

Dialog evaluators may be specialized for aspects such as general dialog-level issues, functional aspects of agents, domain-specific issues, or user preferences. For example, one dialog evaluator monitors the dialog flow and checks that the interface is consistent, ensuring that the dialog management strategy is not changed for every dialog instance. Another evaluator may perform the mapping of dialog tasks to the functionality of agents, while other, more specialized evaluators monitor error situations and the user's need for help and guidance.

■ Information management is a crucial element of adaptive, modular, and distributed applications ■

Dialog agents and evaluators can be used to implement different dialog management strategies to adapt the dialog to the user. Different dialog management strategies, such as the system-initiative approach and the mixed-initiative approach, have different benefits and drawbacks,[22] which makes it useful to use a multiplicity of them in an adaptive way. For example, we have used tutoring agents to provide system-initiative features to the user-initiative interface.[20] In the next section we present a concrete example of such usage in the context of a multilingual e-mail application.

Similarly, alternative solutions for the same dialog tasks can be implemented by using modular agents and evaluators. Alternative dialog agents can be used to provide adaptivity to the interface in special situations. One example is error management. Errors are unavoidable in speech-based applications, and their proper handling is a challenging task. In the agent-based error management model,[19] the task is divided into seven phases, and the actual error correction techniques are implemented using the Jaspis agents and evaluators. For example, there can be agents and evaluators for error detection (e.g., monitoring of speech-recognition confidence scores and dialog history), actual error correction techniques (such as selections from a list and different kinds of confirmations), and decisions regarding how to continue after the error has been corrected.

The agent-based approach for dialog management is demonstrated in the following dialog example (translated from Finnish) from the Busman timetable system.

AGENT U_1:	Welcome to Busman, the Tampere bus information service! How may I help you?
USER:	When does the next bus leave from Central Square to Her ehhh correction from Central Square to Hervanta?
AGENT E_1:	Sorry, I didn't recognize that. Please speak clearly.
USER:	What now?
AGENT H_1:	Busman understands questions such as "When does the next bus to Hervanta depart?", ... [help continues]
USER:	I'm leaving from Central Square, and I'm going to Hervanta.
AGENT E_2:	Sorry, I still didn't recognize that.
AGENT S_1:	Please tell me where you are leaving from.
USER:	From Central Square.
AGENT G_1:	What is your destination?
USER:	Hervanta.
AGENT G_2:	From Central Square to Hervanta use lines 13, 20, 23 and 30.
AGENT S_2:	Do you want to hear timetables?
USER:	Yes.
AGENT G_3:	From Central Square to Hervanta on weekdays at 11:15 bus in line 23...

In this example, five types of dialog agents are present: user-initiative dialog agents (U_1), system-initiative dialog agents (S_1, S_2), error management agents (E_1, E_2), help agents (H_1) and general dialog agents (G_1, G_2, G_3). All of these agents operate on the basis of the shared form and concept definitions that are stored in the information storage. In the beginning, the dialog evaluators select a user-initiative agent (U_1). Recognition errors occur due to the complex recognition grammars, and the first error management agent (E_1) is selected. After that, the user requests help by using a universal command (its usage will be presented in the following sections), and a help agent (H_1) is selected. After the second recognition error, another error management agent (E_2) is chosen, and the dialog evaluators select a system-initiative dialog agent (S_1) to continue the dialog because of the recent errors (the recognition grammars are changed at the same time to more compact ones by the input agents). The recognizer is performing better, and two generic dialog agents (G_1, G_2) are selected to complete the task. Finally, the dialog evaluators select a system-initiative dialog agent (S_2) again, and a generic agent (G_3) provides the results. In the Busman application every dialog agent has a floating-point value (*initiative attribute*) to determine its suitability for system-initiative and user-initiative dialog. This will be presented in the section "Bus timetable systems."

Presentation management

The presentation management module is responsible for generating system outputs suitable for the current dialog state. Usually, this is done on the basis of conceptual messages that the dialog management module produces. Presentation agents produce a representation of speech outputs, which are synthesized by the communication management module. They perform natural language generation, add prosodic information to sentences, decide which (synthesized) voices should be used, and add other modality-specific features into outputs.

Real-world metaphors for presentation agents are actors, who perform their roles as efficiently as they can. Different agents have different capabilities, and they are chosen for different roles. Unlike the real world, the choice of agents can be tailored for each listener, that is, for each user. A real-world metaphor for presentation evaluators is a casting agency, which tries to find the most suitable actors to perform the roles in a play. Similarly, presentation evaluators try to find which presentation agents are the most suitable for the presentation of speech outputs. Several evaluators can be used in this process: in the e-mail domain, we have used five evaluators to choose among the presentation agents.

Multiple presentation agents may be active at a time. First, there may be multiple output requests in information storage, and all of these will be processed. Second, presentation agents may produce several outputs from a single output request. For example, multiple agents may contribute to the output, and a message may be rendered by using multiple modalities. Furthermore, the output generation process can be modular and involve several steps. This could be used when outputs for different user groups are produced. For example, additional information can be added with separate agents for those users who prefer more informative outputs.

Aside from speech outputs, the presentation management module handles the generation of outputs in other modalities. The presentation management module performs *fission*; that is, it decides which modalities should be used for outputs. In this way, it is easy to use different modalities in the system and to take user preferences into account both because dialog management components do not need to be modified when available modalities are changed and because user preferences are learned or given by the user. The presentation management module can be used to support multimodal outputs for different user groups to make single-mode systems more accessible. We have introduced new modalities in this way without the need to modify existing output generation components. Examples of multimodal timetable systems are given in the following sections.

Multilingual outputs can be hard to handle in speech applications. The modular structure of the presentation management module helps to separate language-specific issues into their own agents, and the developers of a new language version are able to translate the system in an iterative way and keep the process manageable. This supports localization, which is one of the key features in making applications more accessible for older users, for example. We used this approach in the case of the Mailman and AthosMail applications, which are presented in the next section. Finally, speech outputs have a strong influence on the user's choice of words, and therefore on the interaction as a whole. For this reason, presentation agents should produce a consistent speech interface and use knowledge from dialog management, user inputs, and the user model in this process.

Input management

Input agents and evaluators are similar to presentation agents and evaluators. They take conceptual input requests from information storage, select appropriate modalities, and add the necessary control information, such as device configuration parameters. The resulting *control messages* are then processed by the communication module. In this way, conceptual input requests, modality selection, and device control are separated from each other.

Input management components take care of the selection and creation of the input vocabulary (recognition grammars or language models in speech applications). Context-sensitive grammars may be selected or generated in system-initiative dialogs, meta-dialogs, and other situations where users are expected to speak in various ways. Recognition grammars should be consistent with the way that the system speaks to the user. Even in simple cases, such as when synonyms and yes/no dialogs are used, a wrong lexical selection may lead to problems if output and input languages are not consistent.[23] Because the order in which Jaspis managers are executed can be freely customized, application developers may decide how they model topics, speech styles, and lexical selection in different modules, that is, in which order the corresponding modules will be executed.

Generation of control information is modality-specific, and a highly application-specific and iterative process. By selecting input modalities for specialized agents, dialog-level components can be used without modification. It should be noted, however, that modality change is not just a configuration parameter, but instead may involve complex relations between input, output, and dialog modules. Input management components do not perform actual multimodal fusion (i.e., the process in which the results of multiple input modalities, generally multiple information sources, are combined). This is the task of the communication module.

Communication management and application development

Communication, that is, low-level input and output management, differs in several ways from other parts of the architecture, although it still shares the agents–managers–evaluators paradigm. In addition to agents and evaluators, communication management includes a layered and concurrent architecture for handling communication between devices and other external resources. This architecture consists of *devices, clients, connections, servers, handlers*, and *engines*. Devices and engines are I/O-specific components; clients, connections, servers and handlers can be used in other parts of the system for distribution of components.

In order to create intuitive and rich interfaces, we need flexible models for handling interaction.[4] In Jaspis, the communication manager coordinates input and output devices, using I/O agents to interpret and conceptualize inputs (including natural language understanding and multimodal fusion), and I/O evaluators to coordinate devices. This process is carried through in a timely and concurrent manner, and feedback may be provided to the devices while they are still gathering inputs or presenting outputs.

With respect to technology, the Jaspis framework is based on dynamic objects, XML documents, a set of core infrastructure classes, and more than 20 extensions. XML is used in all parts of the architecture, and the system supports several standards and markup languages. For example, it includes support for several synthesis markup languages, and we are currently implementing VoiceXML support for distributed dialogs and various mobile devices. Because Jaspis-based systems are modular and distributed, an important part of practical application development is configuration, which is performed using XML-based configuration files.

The reference implementation of the architecture is released as open-source software (under the LGPL [Lesser General Public License] license) to support practical application development and can be downloaded from its home page (http://www.cs.uta.fi/hci/spi/Jaspis/). Further details about the communication management and application development are outside the scope of this paper, but can be found in Reference 13.

JASPIS APPLICATIONS

In this section, we present several applications using the Jaspis architecture. These applications cover many areas, including information services, multilingual systems, and pervasive computing applications. Many human-computer interaction aspects are involved, and in all of them accessibility issues are taken into account. In particular, we have worked with special user groups to design, iteratively develop, and evaluate these applications.

E-mail applications

The e-mail domain is an especially suitable area for speech applications. E-mail itself is one of the most successful applications in the history of computing and involves many issues that are relevant to other services and to universal access for various users in different settings.

Our research group has been involved in the development of several multilingual (Finnish, English, and Swedish) speech-based e-mail applications. The first version of the Mailman application was developed for a Finnish national research project.[24] In the EU-funded DUMAS project, the existing Mailman application served as a basis for the AthosMail application.[25] AthosMail supports more advanced functionality, including alternative solutions for input processing, dialog management, and output generation, to bring more robustness and adaptivity to the interaction.[17]

These e-mail applications allow the user to access his or her mailbox by using a standard mobile or fixed-line phone. They offer functionality for most e-mail reading tasks. The systems provide both speech input (speech recognition) and DTMF (Dual-Tone Multi-Frequency, also known as "touch-tone") interfaces which may be used in a multimodal fashion. We have worked together with visually impaired users and learned in the user studies that their preferences are quite different from those of other users, especially those of normally sighted novice users. In order to help different users, we have implemented several features that provide adaptive interaction supporting accessibility for various user groups.[26]

Flexible interaction management

In the AthosMail application, two alternative dialog management approaches are used to bring more flexibility to the interaction.[18] All dialog components use the same dialog history, which is represented as a discourse tree in the information storage. In addition, the tutoring agents bring system-initiative features to the user-initiative interface,[20] complementing the main dialog components.

When the interface is adapted for different users, it is not possible for all functionality to be used by all user groups in all cases (e.g., for technological, economical, or interface reasons). To share components and maintain portability, a common core of functionality is needed. More advanced interaction methods can be built on top of this core, taking into account the specific needs and capabilities of each user group. In the Mailman and AthosMail systems, the different configurations of the system share the same basic functionality. In addition to speech inputs, all functionality can be accessed by the DTMF interface. In small mailboxes, elementary numbers (1–9) are used to select messages and folders. In this way, both the speech and the DTMF interfaces are robust and simple to use. If the user has a large number of messages, more complex recognition grammars are selected by input evaluators to access the messages, and the DTMF interface uses multiple keys to access messages. In addition, we have adapted the recognition grammars to contain the names and keywords found in each user's e-mail messages. We have found the DTMF interface to be important for users who are visually impaired, who have usually learned to use it in other applications and often prefer it over the speech recognition interface. However, when we designed DTMF layouts with these users and tested them with sighted novice users, the novice users found them hard to use. We discuss this further in the following sections.

■ In order to coordinate interaction in a flexible way, it is better to use decentralized control logic ■

Adaptive system outputs

In the e-mail domain, the main challenge is how to read messages and present other information so that the outputs are both intelligible and pleasant for the user. In the presentation components, different prompts are used for different users (novices, regular users, experts). This is implemented in such a way that each type of prompt (for example, a brief and a verbose version of a welcome prompt) has its own agents, and the appropriate one is selected at runtime, based on dynamically calculated user profiles. Presentation agents for multiple languages are implemented in the same way, thus making it easy to port the applications to different languages. Overall, multilinguality is one of the key issues in e-mail systems.[24] Users have different ways to approach multilingual systems: for example, some of the visually impaired users prefer to use a Finnish synthesizer for English messages as well because they are accustomed to how the synthesizer pronounces English messages. This kind of user preference can be modeled using Jaspis evaluators to dynamically select proper agents for reading the messages.

In order to present large lists of messages efficiently and adapt them for each user, the messages are categorized into meaningful groups by using an algorithm that tries to create both meaningful and easily manageable groups. Information about the messages, such as header information, is used for this purpose. The number of messages within each group is kept within the limits of human capabilities to handle spoken data.[27] This helps the user to get an overview of the mailbox contents even with a large number of messages. By dynamically adapting the groupings for each situation, it is possible to use alternative input methods and strategies, such as the DTMF keys. In particular, users who are visually impaired often prefer these communication methods. In addition, tapering is used to reduce the length of the lists, by including only the meaningful information in them.

■ Usually, application-specific components are encapsulated in their own agents, and inheritance is used to maximize runtime efficiency ■

Help and guidance

In order to provide help and guidance for novice users, AthosMail supports a set of universal commands[28] that adapt to the interaction. With the "tell me more" command, the user can get a detailed presentation of the current dialog situation. For example, when a message is selected, only the main information is spoken. With the "tell me more" command, the user can access additional information, such as length of the message, date, and time. Another universal command is the "what now?" command, which triggers context-sensitive help, in which only relevant guidance is given to the user, taking into account the current dialog situation and history. These features are demonstrated in the following dialog (translated from Finnish):

SYSTEM:	Message number one from Esa-Pekka Salonen, subject is "AthosMail review"
USER:	Tell me more.
SYSTEM:	Message number one from Esa-Pekka Salonen, subject is "AthosMail review, urgent." Message was sent on Wednesday at 14.24. It has 6 lines. Summary of message: "We should prepare for the review. How about meeting tomorrow?"
USER:	What now?
SYSTEM:	You can make the system read the message by saying "read", or you can select another message by saying, for example, "second message", ...

Another form of guidance is provided by tutoring agents.[20] They introduce the system to the user, monitor how well the user interacts with the system, and provide guidance accordingly. Tutoring components take the initiative in certain situations. They use the dialog history, the user model, and their own tutoring plan in this process. Technically, the Jaspis agents and evaluators make it possible to add the tutoring agents to the system without modifications to other components. Furthermore, the tutoring feature can be turned dynamically on and off. The following dialog (translated from Finnish) gives an example of the tutoring feature:

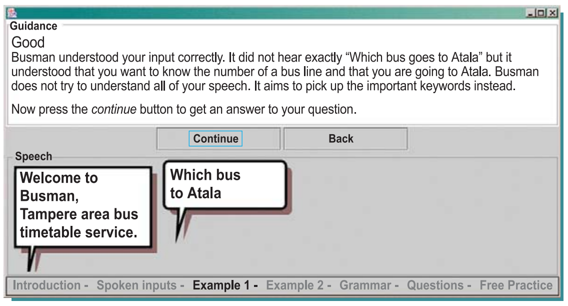
SYSTEM:	Please wait, connecting to your mail server ... Hi, Test user. No new messages. 17 old messages. You have 3 groups. Group one, ...
TUTOR:	Hi, I'm your tutor. I'll teach you how to use the system. Next, choose one of the available groups. You can do this by saying, for example, "third group." So, please use the group number you wish.
USER:	Go to the third group
TUTOR:	Good. Now you are in the third group. Next AthosMail will list messages in the group.
SYSTEM:	[the system lists the messages in the selected third group]

Bus timetable systems

In addition to e-mail systems, we have been working with several bus timetable systems. As with the e-mail domain, the timetable domain is practical, provides plenty of research challenges, and the results can be used in other information services. Similar human-operated systems are widely available, and there is a need to replace existing systems with automated systems. Furthermore, with automated systems it is possible to produce new services that are either not possible or not economically viable with human operators. For special user groups, they may represent the only possibility to access information, as has been noted recently in many areas.

The Busman and Interact systems provide bus route and timetable information for the Tampere and Helsinki areas, two major cities in Finland. The functionality of these systems is similar to other timetable services, such as MALIN.[29] The user may request information such as bus routes (e.g., "Which bus goes to the university hospital?") and timetables (e.g., "When does the next one leave?"). Like the e-mail applications, these systems have a speech interface and can be used with mobile and fixed-line telephones. In contrast, a third timetable system (Stopman) and its ongoing continuation (PUMS), are based more on the system-initiative approach and can be used with a multimodal personal digital assistant (PDA) interface as well. These applications are designed in close collaboration with users who are visually impaired. Currently, we are implementing mobile clients for the Stopman system. Wireless connections, such as GPRS (General Packet Radio Service) or UTMS (Universal Mobile Telecommunications System), provide speech recognition and synthesis services on PDAs and cellular phones for users who are visually impaired. Jaspis components are used both on the server and the client side, and the interaction, such as menu selection, is adapted by the agents and evaluators to the capabilities of each device. In addition to speech and nonspeech audio, we have plans to use touch-sensitive displays. In the Interact system, multimodal extensions such as a graphical touch-screen interface for information kiosk usage and a short-message interface for mobile nonspeech usage were implemented for users who are deaf or hard of hearing.

In the Busman system, we have experimented with a truly mixed-initiative approach to dialog management, which we have found to be the key aspect in making dialogs work in this domain. In this approach, each dialog agent has an initiative attribute, which indicates its suitability for system-initiative and user-initiative dialogs. In the current implementation, one dialog evaluator is used to give preference to system-initiative agents when the interaction proceeds normally, and to user-initiative agents when errors are present. In this way, the system tries to adapt to the interaction by changing the dialog strategy on the basis of the success of the


Good

Busman understood your input correctly. It did not hear exactly "Which bus goes to Atala" but it understood that you want to know the number of a bus line and that you are going to Atala. Busman does not try to understand all of your speech. It aims to pick up the important keywords instead.

Now press the *continue* button to get an answer to your question.
Guidance
Continue
Back
Speech
Welcome to Busman, Tampere area bus timetable service.
Which bus to Atala
Introduction - Spoken inputs - Example 1 - Example 2 - Grammar - Questions - Free Practice

Figure 3
Multimodal tutoring in the Busman system

interaction. An example dialog with this feature was given in the section "Dialog and error management." In addition, the Busman application includes a set of tutoring agents[20] to provide multimodal guidance and assistance in error situations. ***Figure 3*** illustrates this guidance. It also demonstrates how spoken dialogs can be visualized automatically. In terms of technology, the tutoring is implemented as a set of agents and evaluators providing GUI outputs and inputs, while the Busman agents and evaluators provide speech inputs and outputs. Again, the tutoring component could be included in the system without any modifications to the Busman code. All the information necessary for the tutor agents is already available in the information storage.

Pervasive speech applications

Doorman is a pervasive computing system that uses speech and audio as its main modalities. It has been designed with both sighted users and users who are visually impaired in mind. We have used it to experiment with implementations of speech interfaces in pervasive-computing settings. In this field, many new challenges have been encountered. Experiments from this domain have greatly influenced the development of the new Jaspis-2 architecture,[30] which supports multiple concurrent users and dialogs.

The Doorman system serves staff members and visitors in an office environment. The system controls access to this environment by identifying staff members and helping visitors to find the place or person that they are looking for. The system gives guidance to visitors about how to reach their destinations. Several extensions, such as an auditory awareness information service, have been added to the system. The system has features similar to the Office Monitor[31] and PER[32] systems.

The Doorman system uses speech recognition and speaker identification to recognize users from their speech. Synthesized speech, nonspeech audio, and pointing gestures are used for multimodal outputs. For visually impaired users, the speech outputs, augmented with nonspeech audio (including auditory icons) contain the necessary information. The presence, location, and state of users can be tracked from different signals and their fusion. People on the move can be followed using infrared and pressure-sensitive EMFi (Electro-Mechanical Film) technologies. Other information (such as keyboard, mouse, or application activity) is received from desktop computers and mobile devices.

■ Tutor-based guidance is a promising approach to making systems accessible for different users ■

The main feature of the system is guidance. The office layout is modeled in such a way that the system is able to generate various alternative guidance descriptions with varying levels of detail. The selection of details is highly dependent on the particular user group process. Route instructions given to users who are visually impaired need to be founded on different criteria from those for sighted users.[33] For example, the visual landmarks that are useful for sighted users can be replaced with auditory landmarks for visually impaired users. This can be understood by comparing the following system outputs:

For sighted users:

> Follow the hallway until you come to a crossroad with a sign “copy machine.”

For users who are visually impaired:

> Continue ahead until you can hear the following sound [sound of a copy machine].

Communication with the user is handled through communication points located in strategic places in the facility, such as intersections. This enables the distribution of route instructions in small units, which are easier to understand and memorize, hence easing the cognitive load on the user.[34] These smaller units are presented according to the user’s progress along the route. Distributed route instructions are supported by auditory cues, which function as beacons along the route. An auditory cue played from the next communication unit on the route functions as a target which the user can aim at. This serial component in multimodal spatial information is very important for visually impaired users.[33] The user can ask for more detailed information from the system when passing communication units.

In addition to guidance related to navigation, the system is capable of giving added information in the manner of an exhibition guide. The guide gives a tour of the premises, showcasing physical artifacts. The same identification and positioning information enables cross-references to earlier events and personalized content generation with temporally as well as spatially dynamic modality choices. Such choices are helpful for various accessibility problems, for example, as demonstrated in the following system outputs:

For sighted users:

> As you can see, the artist uses the same technique of contrasting colors when choosing materials, as in the three previous sculptures.

For users who are visually impaired:

> If you touch the sculpture, you can feel the contrasting textures of soft wool and unfinished cast bronze, a similar method to that used in the artist’s two previous sculptures.

One group of extensions to the Doorman system is applications for supporting awareness and group communication. In these applications, auditory information is presented in a way which helps to keep users aware of events occurring in their surroundings. For example, an application monitors the presence of group members and informs other group members about the activity in the group (e.g., usage of desktop computers or tracking of people using floor sensors). The presentation of the information is done so that auditory icons and other auditory information indicate important changes in the environment, such as arrival of other people and incoming messages. Through ambient soundscapes (i.e., acoustic environments that surround the user), information is made available in the periphery of the senses.[35] Both continuous and temporal information are presented. Natural sounds, such as walking sounds and birds singing, are used in auditory presentations.[36] From the architectural point of view, creating and controlling such soundscapes is not trivial. One has to map the information to specified qualities of the sound, as well as manage and adjust the whole composition dynamically according to preset rules, in order to assure the continuity and harmony of the presentation.

The guidance and awareness applications can be very useful for users with special needs. For example, navigation in an unfamiliar environment can be a challenging task for users who are visually impaired and cannot rely on visual landmarks. Similarly, a good deal of tacit awareness information can be sensed, fused together, processed, and made available through several modalities and levels of abstraction. Our goal is to provide methods to express meaningful information from the environment to help different users in their everyday tasks. We have addressed this challenge by implementing alternative interaction techniques and applied them in the various applications using the agents–managers–evaluators paradigm. In order to better understand the needs of different users, we plan to install the Doorman system in places where special user groups work and live.

USER STUDIES

We arranged several user experiments at different stages of the development of the Jaspis applications. We have used various representative user groups in these studies, focusing on users who are visually impaired in particular. In addition, user feedback has been received from the public use of the applications. Mailman has been available to the public since 1999, and the bus timetable service Stopman has been in public use since 2003. In these studies, various Jaspis evaluation tools have been used.

User-centered design

To take representative users into account in the development of the Mailman application, we arranged a design session with the Finnish Federation of the Visually Impaired. The purpose of the session was to introduce the components that had been implemented and to design speech inputs and outputs by experimenting with different alternatives. Similarly, we designed bus timetable systems with representative users. We found it useful to conduct early user tests with partially implemented applications. In particular, we managed to set up design sessions with Jaspis tools without any modifications to the application code. This significantly accelerated the process and allowed a wide range of experiments to be conducted interactively.

In order to identify usability problems, we collected feedback by asking specific questions and requesting free-form comments on the usability of the applications. These are good ways of identifying specific problems, such as poor rendering of e-mail contents. Users who are visually impaired, especially those working in the information technology industry, may be considered to be expert users of speech applications, and they are also, in many ways, a very demanding user group. Other expert opinions were gathered from students taking a usability course with no prior experience of speech applications, but with an ability to analyze user interfaces.

The results obtained with representative end users have been used in implementations using Jaspis agents and evaluators in practical applications. For example, we have used and evaluated different DTMF layouts for e-mail applications designed with users who are visually impaired. Similarly, we have implemented new agents to read e-mail messages more fluently on the basis of the comments from users who are visually impaired. With our modular and adaptive approach, these components can be included in applications and selected dynamically when needed. Furthermore, the resulting agents can be used in other similar applications as well. For example, a telephone-based document-reading system for users who are visually impaired has been implemented by other developers on the basis of the Jaspis architecture and the AthosMail application.[37]

Wizard of Oz studies

"Wizard of Oz" (WOZ) experiments, in which some parts of the system are simulated with a human operator, are useful for speech application development as a usability-testing and data-collection method. There are several tools to aid such experiments, such as SUEDE.[38] One particular limitation of such external tools is that they simulate the whole system, and as a result, the software used in the experiment is different from the software to be used in the actual system. In most cases, however, at least a partly implemented system is needed or wanted. Thus, it would be useful to run the actual system in a WOZ mode. This is the purpose of the Jaspis WOZ tools. For building WOZ interfaces, Jaspis includes a generic WOZ GUI (graphical user interface), which can be customized for different purposes by using an XML-based description language. This way, setting up a WOZ GUI requires a definition of the interface, and no programming is necessary.

■ The need for adaptivity and supporting architectures is identified to be one of the key elements in the next generation of speech applications ■

We used the Jaspis tools to set up a WOZ experiment to test the Doorman and AthosMail applications. The usability of the Doorman system was evaluated as part of the development process by performing a WOZ experiment in which speech recognition and speaker recognition were simulated. This was done by using a Jaspis WOZ applet. Otherwise, the guidance system was functional, and there were no modifications to the Doorman components. Aside from evaluating the system functionality and multimodal guiding instructions, we wanted to collect data about how people speak to the system.

With the Doorman WOZ experiment, we were able to find many ways to improve the system. The main findings from the user study concerned the way that the users spoke to the system and the guidance given by the system. For example, the guidance algorithm was modified to use landmarks and relative measures, and the dialog was divided to consist of smaller dialog units. There were several user groups (staff members, students, and visitors), who had very different needs for the interaction. The experiment and the results are described in detail in Reference 39. In the future, we will implement more adaptivity in the guidance to customize it for more user groups and individuals, making the system more accessible.

We also used the WOZ method with a similar setup to test the AthosMail application. We used the WOZ study as a starting point for the AthosMail application design, together with experiments in the usage of the Mailman system.

Evaluation of working applications

We have done several experiments with working versions of Jaspis applications. The e-mail systems have been evaluated with several approaches. In these studies, we have found several interesting phenomena from the user diversity point of view; for example, males and females have quite different expectations and perceptions of the speech interface;[40] tutor-based guidance is a promising approach to making the systems accessible for different users;[41] and sighted and visually impaired users prefer different DTMF layouts. When we tested various DTMF layouts, some of which had been designed together with users who were visually impaired, sighted novice users found the layouts to be randomly assigned to the application functionality. Users who were visually impaired, on the other hand, had a strong mental model (learned from the use of text-based applications and screen readers) concerning how the DTMF keys should be used for navigation in speech interfaces. In addition, this showed that novice users need support and guidance when DTMF interfaces are used. In the case of the Jaspis applications, customized interfaces and their dynamic selection are modeled using agents and evaluators, and we have implemented several context-sensitive features for help and guidance, as discussed earlier.

We have conducted user studies with the working versions of the timetable systems as well. Several problems were found after the tests. Based on the findings, Jaspis agents were used to improve usability and add new functionality to the system. For example, the users' language in the experiments was different from the language in the recorded conversations between human operators and users. In the human-to-human recordings, conversational language was used, but in the human-to-computer interaction, shorter sentences, terser style, and different words and structures overall were used. Most interestingly, in the human-to-computer conversations, some people used nongrammatical sentences similar to universal commands[28] and tried to adapt their language to the system. A representative example is "departure: railway station [pause] destination: Arabia." The lesson learned here is that the language model should not be acquired directly from human-to-human conversations because the language used in human-to-computer communication is quite different. We solved this problem by adding a set of new Jaspis agents to take care of those inputs that the original, corpus-based input-interpretation components were not able to handle.

In addition, we concluded that it is more efficient to use a system-initiative dialog strategy as a default. We used this approach in the more recent Stopman application, which has been evaluated by analyzing the calls from the public use of the system and has been found usable in real-life practical situations. In this context as well, it seems that users who are visually impaired have quite different ways to interact with the system. For example, they seem to interrupt the system prompts immediately in the beginning of the call, whereas most users do not interrupt the greeting prompts (however, because the calls are anonymous, we cannot identify the users accurately).

In summary, the tests and experiments with the applications have provided essential information for iterative application development. In particular, we have found the needs of different user groups to be quite different. From the technical perspective, we have found it efficient to work with the Jaspis agents–managers–evaluators paradigm, which has allowed us to improve the accessibility of the systems by implementing alternative components for the same tasks. Jaspis agents have made it possible to efficiently realize results in terms of software components and to construct sophisticated tools to aid the evaluation process.

CONCLUSION

We have presented a number of ways in which the needs of different users can be taken into account with flexible and adaptive architecture solutions. The Jaspis architecture includes the generic agents–managers–evaluators paradigm, which has proven to be successful when adaptive applications and interaction techniques have been built on top of the architecture. When applications are constructed with adaptivity in mind, it is easy to support different user groups, as demonstrated in several Jaspis applications.

Overall, the need for adaptivity and supporting architectures is identified to be one of the key elements in the next generation of speech applications.[42] We have found the compact agents to be very useful when alternative interaction methods are included in existing applications. Jaspis agents together with the built-in adaptation mechanism of Jaspis make it possible to include additional and alternative functionality for different needs without modifications to the existing components. An example of this is the customized DTMF interfaces for telephone systems and spoken guidance in indoor environments that takes the needs (e.g., visual versus nonvisual landmarks) of different users into account.

In order to better support sophisticated speech-based systems, the Jaspis architecture has been extended in several ways. The main new feature of the Jaspis-2 architecture[30] is the support for concurrent interaction in a coordinated and synchronized way. In the Jaspis-2 architecture, indirect messaging (using triggers and transactions) between system components (i.e., agents) is used to support concurrent interaction, while the coordinated adaptation mechanism of the original architecture (using managers and evaluators) is preserved.

At the same time, the architecture is more blackboard-oriented, and the distribution of components is extended to support sharing of components among modules. All the changes have been made in a way that makes it possible to use the adaptive features of the original architecture. This is still preliminary work, and in the future we will continue the development of the Jaspis-2 architecture in close relation to pervasive-computing systems in order to find out how new application areas can be better supported by the system architecture.

The solutions offered by the Jaspis architecture are targeted mainly for speech-based and auditory applications. Similar challenges can be found in other application domains as well. In particular, non-speech-based approaches to support accessibility are among the issues that will be part of the future development of the Jaspis architecture and its applications.

**Trademark, service mark, or registered trademark of Turner Entertainment Corporation.

CITED REFERENCES

1. E. Mynatt and G. Weber, “Nonvisual Presentation of Graphical User Interfaces,” *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI’94)*, Boston, MA (1994), pp. 166–172.
2. A. Edwards, “Multimodal Interaction and People with Disabilities,” in *Multimodality in Language and Speech Systems*, B. Granström, D. House and I. Karlsson, Editors, Kluwer Academic Press, Dordrecht, The Netherlands (2002), pp. 73–92.
3. D. Garlan and M. Shaw, “An Introduction to Software Architecture,” in *Advances in Software Engineering and Knowledge Engineering, Series on Software Engineering and Knowledge Engineering, Volume 2,* V. Ambriola and G. Tortora, Editors, World Scientific Publishing Company, Singapore (1993), pp. 1–39.
4. J. Allen, G. Ferguson, and A. Stent, “An Architecture for More Realistic Conversational Systems,” *Proceedings of the International Conference on Intelligent User Interfaces 2001 (IUI-01)*, Santa Fe, New Mexico (2001), pp. 14–17.
5. M. McTear, “Spoken Dialogue Technology: Enabling the Conversational Interface,” *ACM Computing Surveys* **34**, No. 1, 90–169 (March 2002).
6. S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, “Galaxy-II: A Reference Architecture for Conversational System Development,” *Proceedings of the Fifth International Conference on Spoken Language Processing (ICSLP98)*, Sydney, Australia (1998), pp. 931–934.
7. D. L. Martin, A. J. Cheyer, and D. B. Moran, “The Open Agent Architecture: A Framework for Building Distributed Software Systems,” *Applied Artificial Intelligence: An International Journal* **13**, No. 1–2 (January–March 1999), pp. 91–128.
8. A. Rudnicky and W. Xu, “An Agenda-based Dialog Management Architecture for Spoken Language Systems,” *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop* (1999), pp. 337–340.
9. D. Bohus and A. Rudnicky, “RavenClaw: Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda,” *Proceedings of the Eighth European Conference on Speech Communication and Technology (Eurospeech 2003)*, Geneva, Switzerland (2003), pp. 597–600.
10. I. O’Neill, P. Hanna, X. Liu, and M. McTear, “The Queen’s Communicator: An Object-Oriented Dialogue Manager,” *Proceedings of the Eighth European Conference on Speech Communication and Technology (Eurospeech 2003)*, Geneva, Switzerland (2003), pp. 593–596.
11. B. Pakucs, “Towards Dynamic Multi-Domain Dialogue Processing,” *Proceedings of the Eighth European Conference on Speech Communication and Technology (Eurospeech 2003)*, Geneva, Switzerland (2003), pp. 741–744.
12. M. Turunen and J. Hakulinen, “Jaspis—A Framework for Multilingual Adaptive Speech Applications,” *Proceedings of the Sixth International Conference on Spoken Language Processing (ICSLP 2000)*, Beijing, China (2000), pp. 719–722.
13. M. Turunen, *Jaspis—A Spoken Dialogue Architecture and Its Applications*, Ph.D. dissertation, University of Tampere, Finland, Department of Computer Sciences, Report A-2004-2, ISBN 951-44-5896-6, ISSN 1459-6903 (2004).
14. N. Blaylock, J. Allen, and G. Ferguson, “Synchronization in an Asynchronous Agent-Based Architecture for Dialogue Systems,” *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialog*, Philadelphia, PA; Association for Computational Linguistics (2002), pp. 1–10.
15. O. Lemon, A. Bracy, A. Gruenstein, and S. Peters, “The WITAS Multi-Modal Dialogue System I,” *Proceedings of the Seventh European Conference on Speech Communication and Technology (Eurospeech 2001)*, Aalborg, Denmark (2001), pp. 1559–1562.
16. I. O’Neill and M. McTear, “Object-Oriented Modeling of Spoken Language Dialogue Systems,” *Natural Language Engineering* **6**, No. 3–4, Cambridge University Press (2000), pp. 341–362.
17. M. Turunen, E.-P. Salonen, M. Hartikainen, and J. Hakulinen, “Robust and Adaptive Architecture for Multilingual Spoken Dialogue Systems,” *Proceedings of the Eighth International Conference on Spoken Language Processing (INTERSPEECH 2004-ICSLP)*, Jeju Island, Korea (2004), pp. 3081–3084.
18. E.-P. Salonen, M. Hartikainen, M. Turunen, J. Hakulinen, and A. Funk, “Flexible Dialogue Management Using Distributed and Dynamic Dialogue Control,” *Proceedings of the Eighth International Conference on Spoken Language Processing (INTERSPEECH 2004-ICSLP)*, Jeju Island, Korea (2004), pp. 197–200.
19. M. Turunen and J. Hakulinen, “Agent-Based Error Handling in Spoken Dialogue Systems,” *Proceedings of the Seventh European Conference on Speech Communication and Technology (Eurospeech 2001)*, Aalborg, Denmark (2001), pp. 2189–2192.
20. J. Hakulinen, M. Turunen, and E.-P. Salonen, “Agents for Integrated Tutoring in Spoken Dialogue Systems,” *Proceedings of the Eighth European Conference on Speech Communication and Technology (Eurospeech 2003)*, Geneva, Switzerland (2003), pp. 757–760.
21. J. Chu-Carroll, “MIMIC: An Adaptive Mixed Initiative Spoken Dialogue System for Information Queries,” *Proceedings of the 6th ACL Conference on Applied Natural Language Processing*, Seattle, WA (May 2000), pp. 97–104.
22. M. Walker, J. Fromer, G. Fabbrizio, C. Mestel, and D. Hindle, “What Can I Say?: Evaluating a Spoken Language Interface to Email,” *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI 98)*, Los Angeles, CA (1998), pp. 582–589.
23. B. Hockey, D. Rossen-Knill, B. Spejewski, M. Stone, and S. Isard, “Can You Predict Responses to Yes/No Questions? Yes, No, and Stuff,” *Proceedings of the Fifth European Conference on Speech Communication and Technology (Eurospeech 1997)*, Rhodes, Greece (1997), pp. 2267–2270.
24. M. Turunen and J. Hakulinen, “Mailman—a Multilingual Speech-Only E-Mail Client Based on an Adaptive Speech Application Framework,” *Proceedings of the Workshop on Multi-Lingual Speech Communication (MSC 2000)*, Kyoto, Japan (2000), pp. 7–12.
25. M. Turunen, E.-P. Salonen, M. Hartikainen, et al., “AthosMail—A Multilingual Adaptive Spoken Dialogue System for the E-mail Domain,” *Proceedings of the Workshop on Robust and Adaptive Information Processing for Mobile Speech Interfaces*, Geneva, Switzerland (2004), pp. 77–86.
26. E.-P. Salinen, M. Hartikainen, M. Turunen, J. Hakulinen, J. Rissanen, K. Kanto, and K. Jokinen, “Adaptivity in a Speech-Based Multilingual E-mail Client,” *Proceedings of NordiCHI 2004*, Tampere, Finland (2004), pp. 437–440.
27. B. Suhm, “Towards Best Practices for Speech User Interface Design,” *Proceedings of the Eighth European Conference on Speech Communication and Technology*

(Eurospeech 2003), Geneva, Switzerland (2003), pp. 2217–2220.

28. R. Rosenfeld, X. Zhu, S. Shriver, A. Toth, K. Lenzo, and A. W. Black, "Towards a Universal Speech Interface," *Proceedings of the Sixth International Conference on Spoken Language Processing (ICSLP 2000)*, Beijing, China (2000), http://www-2.cs.cmu.edu/~stef/papers/ICSLP00-usi.pdf.
29. N. Dahlbäck and A. Jönsson, "Knowledge Sources In Spoken Dialogue Systems," *Proceedings of the Seventh European Conference on Speech Communication and Technology (Eurospeech 1999)*, Budapest, Hungary (1999), pp. 1523–1526.
30. M. Turunen and J. Hakulinen, "Jaspis^2 - An Architecture For Supporting Distributed Spoken Dialogues," *Proceedings of the Eighth European Conference on Speech Communication and Technology (Eurospeech 2003)*, Geneva, Switzerland (2003), pp. 1913–1916.
31. N. Yankelovich and C. McLain, "Office Monitor," *Proceedings of the Conference on Human Factors in Computing Systems (CHI '96)*, Vancouver, British Columbia, Canada; ACM Press, New York (1996), pp. 173–174.
32. B. Pakucs and H. Melin, "PER: A Speech Based Automated Entrance Receptionist," *Proceedings of the 13th Nordic Conference of Computational Linguistics (NoDaLiDa '01)*, Uppsala, Sweden (2001).
33. S. Werner, B. Krieg-Brückner, H. A. Mallot, K. Schweizer, and C. Freksa, "Spatial Cognition: The Role of Landmark, Route, and Survey Knowledge in Human and Robot Navigation," in *Informatik '97*, M. Jarke, K. Pasedach, and K. Pohl, Editors, Springer-Verlag, Vienna, Austria (1997), pp. 41–50.
34. K. Hook, "An Approach to a Route Guidance Interface," Licentiate Thesis, Stockholm University, Department of Computer and Systems Sciences, ISSN 1101 8526 (1991).
35. M. Weiser and J. S. Brown, "The Coming Age of Calm Technology," in *Beyond Calculation: The Next Fifty Years*, Copernicus, New York, NY (1997), pp. 75–85.
36. K. Mäkelä, J. Hakulinen, and M. Turunen, "The Use Of Walking Sounds In Supporting Awareness," *Proceedings of the International Conference on Auditory Display (ICAD 2003)*, Boston, MA (2003), pp. 144–147.
37. P. Heiskari, "The Finnish AthosNews: A News-Reading Application for Visually Impaired Users," *Proceedings of the 20th International Conference on Computational Linguistics*, Geneva, Switzerland (2004), p. 43.
38. S. Klemmer, A. Sinha, J. Chen, A. Landay, N. Aboobaker, and A. Wang, "SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces," *CHI Letters, The 13th Annual ACM Symposium on User Interface Software and Technology (UIST 2000)* **2**, No. 2 (2000), pp. 1–10.
39. K. Mäkelä, E.-P. Salonen, M. Turunen, J. Hakulinen, and R. Raisamo, "Conducting a Wizard of Oz Experiment on a Ubiquitous Computing System Doorman," *Proceedings of the International Workshop on Information Presentation and Natural Multimodal Dialogue* (2001), pp. 115–119.
40. M. Hartikainen, E.-P. Salonen, and M. Turunen, "Subjective Evaluation of Spoken Dialogue Systems Using SERVQUAL Method," *Proceedings of the Eighth International Conference on Spoken Language Processing (INTERSPEECH 2004-ICSLP)*, Jeju Island, Korea (2004).
41. J. Hakulinen, M. Turunen, E.-P. Salonen, and K.-J. Räihä, "Tutor Design for Speech-Based Interfaces," *Proceedings of Designing Interactive Systems (DIS2004)*, Cambridge, MA (2004), pp. 155–164.
42. M. McTear, "New Directions in Spoken Dialogue Technology for Pervasive Interfaces," *Proceedings of the Workshop on Robust and Adaptive Information Processing for Mobile Speech Interfaces* (2004), pp. 57–64.

Accepted for publication February 16, 2005.
Published online August 5, 2005.

Markku Turunen
University of Tampere, Department of Computer Sciences, Tampere Unit for Computer Human Interaction, Speech-Based and Pervasive Interaction Group, Kanslerinrinne 1, FIN-33014 University of Tampere, Finland (mturunen@cs.uta.fi). Dr. Turunen received a Ph.D. degree in computer science from the University of Tampere in March 2004. He joined the Tampere Unit for Computer-Human Interaction in 1998, and has been coordinating one of its research groups, the Speech-Based and Pervasive Interaction group. His research interests are in speech-based, auditory, and pervasive applications.

Jaakko Hakulinen
University of Tampere, Department of Computer Sciences, Tampere Unit for Computer Human Interaction, Speech-Based and Pervasive Interaction Group, Kanslerinrinne 1, FIN-33014 University of Tampere, Finland (jaakko.hakulinen@cs.uta.fi). Mr. Hakulinen is a researcher in the Speech-Based and Pervasive Interaction Group. He received an M.Sc. degree in computer science from the University of Joensuu in 1998. He joined the TAUCHI (Tampere Unit for Computer Human Interaction) unit in 1998 and has been working since on the area of speech interfaces. Currently, he is working on software tutoring of speech interfaces.

Kari-Jouko Räihä
University of Tampere, Department of Computer Sciences, Tampere Unit for Computer Human Interaction, FIN-33014 University of Tampere, Finland (kjr@cs.uta.fi). Dr. Räihä is a professor of computer science at the University of Tampere. He received a Ph.D. degree in computer science from the University of Helsinki in 1982 and moved to the University of Tampere soon after. He has worked in human-computer interaction for 15 years and is the founder and head of the Tampere Unit for Computer-Human Interaction, a research unit of more than 50 researchers, faculty members, and graduate students. His research interests are in interaction design, particularly the use of nonstandard modalities, such as eye gaze and speech.

Esa-Pekka Salonen
University of Tampere, Department of Computer Sciences, Tampere Unit for Computer Human Interaction, Speech-Based and Pervasive Interaction Group, University of Tampere, Kanslerinrinne 1, FIN-33014 University of Tampere, Finland (esa-pekka.salonen@cs.uta.fi). Mr. Salonen is a researcher and a Ph.D. candidate at the University of Tampere. He received B.Sc. and M.Sc. degrees in computer science from the University of Tampere in June 2002 and in December 2002, respectively. Mr. Salonen has worked in the Speech-Based and Pervasive Interaction Group since 2001 on various projects and applications.

Anssi Kainulainen
University of Tampere, Department of Computer Sciences, Tampere Unit for Computer Human Interaction, Speech-Based and Pervasive Interaction Group, University of Tampere, Kanslerinrinne 1, FIN-33014 University of Tampere, Finland (anssi@cs.uta.fi). Mr. Kainulainen is a researcher in the Speech-Based and Pervasive Interaction Group. He is working on auditory awareness information in ubiquitous systems.

Perttu Prusi
University of Tampere, Department of Computer Sciences, Tampere Unit for Computer Human Interaction, Speech-Based and Pervasive Interaction Group, University of Tampere, Kanslerinrinne 1, FIN-33014 University of Tampere, Finland (prusi@cs.uta.fi). Mr. Prusi is a researcher in the Speech-Based and Pervasive Interaction Group. He received an M.Sc. degree in computer science from the University of Tampere in February 2005. He is working on speech-based and audio-based guidance systems. ■

Paper II

Jaakko Hakulinen, Markku Turunen, & Esa-Pekka Salonen. Agents for Integrated Tutoring in Spoken Dialogue Systems. In *Proceedings of Eurospeech 2003*, ISCA, pages 757-760.

Agents for Integrated Tutoring in Spoken Dialogue Systems

Jaakko Hakulinen, Markku Turunen, Esa-Pekka Salonen

Speech-based and Pervasive Interaction Group, Tampere Unit for Computer-Human Interaction
Department of Computer and Information Sciences
33014 University of Tampere
FINLAND
{jh, mturunen, eps}@cs.uta.fi

Abstract

In this paper, we introduce the concept of integrated tutoring in speech applications. An integrated tutoring system teaches the use of a system to a user while he/she is using the system in a typical manner. Furthermore, we introduce the general principles of how to implement applications with integrated tutoring agents and present an example implementation for an existing e-mail system. The main innovation of the approach is that the tutoring agents are part of the application, but implemented in a way which makes it possible to plug them into the system without modifying it. This is possible due to a set of small, stateless agents and a shared Information Storage provided by our system architecture. Integrated tutoring agents are easily expandable and configurable, and general agents can be shared between applications. We have also received positive feedback about integrated tutoring in initial user tests conducted with the implementation.

Introduction

While some designers believe that a speech system should be so intuitive that one can immediately start using it, at least some teaching and guidance is usually necessary regardless of the type of the application. The main issue is that the user must know what to say to the system. Grammars in speech systems are always limited in some way. Especially, command based systems with a small vocabulary are powerful only after the user has been taught how to use the system. On the other hand, in systems with open grammars and sophisticated natural language understanding, it is necessary to teach the user the limits and capabilities of the system. This information is often on a higher conceptual level, but may be effectively taught in context. In general, the user must know what he/she can say to the system.

Prompt design is considered to be the key issue in user guidance when systems are targeted to a wide audience [1]. However, it has been found that the initial impression of a system affects the users' image of the system also later on. In a research by Kamm et al. [2] users studied a written tutorial on a web page before they started using the system. Another group learned to use the system by following its prompts and using its help features. The latter group had significantly lower satisfaction with the system, even though they reached comparable performance.

For all of the reasons mentioned it is important to give the users guidance on how to use a speech system. Speech systems often include some sort of manuals on web pages and in print. However, users in general are not very interested in reading manuals, but they would rather want to get into action as soon as possible. We introduce integrated tutoring in speech systems as a solution for this problem. These systems have tutoring as part of their functionality, so users can start to use the applications right away and learn as they go. We show that it is possible to implement integrated tutoring in a way that makes it possible to plug the tutoring components into applications without modifying the original applications.

In this paper we focus on the implementation of the tutoring agents and present an example implementation for an existing speech-based e-mail application. We begin by introducing integrated tutoring systems, and especially the dialogue management part of them. Next, we describe the technical implementation of the integrated tutoring system on top of the Jaspis architecture [3]. This is followed by a concrete example with the Mailman application [4]. Finally, we discuss some issues we have confronted with our tutor system.

Integrated Tutoring Systems

By integrated tutoring we mean functionality where a system gives guidance to a user on the operation of the system while the user is using the system. From the user's point of view, we have implemented the tutoring as a separate dialogue participant, not as part of the system.

Integrated tutors are often used in computer games. They teach the user how to play the game by telling the user what to do next while the user is playing the game. This way the tutorial session is more meaningful and pleasant for the user. By introducing such a system to speech applications, we gain the same advantages. It can also be very efficient to teach some features of the system by letting the user use the features with the actual system. We can also give guidance on certain features at moments when the user can or should try them out, i.e. we can select meaningful context for tutoring. Tutoring is also an efficient way to tell about new features or modifications to an already existing system.

Comparing tutoring to using system prompts to teach the user, we would like to point out several differences. Instead of informing the user about different possibilities, the tutor usually gives instructions to the user on what to do next. The tutor can also check that the user is really doing what was asked of him and possibly discard incorrect inputs. This way we can, for example, build safe learning environments (see [5]) where the user cannot make any fatal operations. In speech systems this is often a welcome feature because the possibility of recognition errors increases the occurrence of mistakes.

In tutoring systems the teaching usually has some sort of structure, so that it is more than just a set of random hints. In addition to teaching the user the actual commands, the tutor can teach on a more conceptual level. The tutor can also check if the user has learned the taught subject by giving the user such tasks that the user has to use skills taught earlier. If the user has problems, the tutor can adapt and teach the subject again. We may also have a tutor which allows the user to ask questions from it. In traditional tutoring systems this is an essential part of tutoring, but when the tutor is part of the tutored system, we can also build effective tutors which get all their feedback from the user by following how the user is using the system (for such functionality in traditional tutoring systems see e.g. [6]). Finally, when the tutor is a separate dialogue partner, there is no need to shorten the system prompts, as the user becomes more familiar with the system.

Dialogue Management

An integrated tutoring system brings up interesting design and technical questions. From the user's point of view the tutor may be seen as a separate person in the dialogue, or the system itself may act as the tutor. The same distinction can be made from the technical point of view as well. The tutoring functionality can be implemented as part of normal dialogue management or it can be a separate implementation working in parallel to normal dialogue management.

When the tutor is presented to the user as a separate dialogue partner, the dialogue becomes a three party one. This causes some changes to the dialogue structure as, for example, turn taking becomes more complicated and dialogue design must answer the question of when and how the tutor takes the turn. The tutor may comment on the user input and tell what will happen next or it can speak after the system and comment on what the user just heard. The tutor can also give advice on what can be said next.

We should also separate the dialogue initiative of the user-system dialogue from the initiative of the user-tutor dialogue. The tutor can take the initiative in user-tutor dialogue while the system may still work in user initiative mode. This way we can effectively teach users in user-initiative systems without altering the systems.

An interesting change to the dialogue structure occurs when a tutor modifies the user inputs. The tutor can, for example, discard an input and let the user try to speak again if the input was not what it was expected to be. In a case like this the tutor controls the user-system dialogue. Another step is when the tutor "speaks" to the system, i.e. gives inputs to the system on behalf of the user and acts as an example to the user. The user may also want to speak to the tutor directly. This may lead to complex situations which need to be taken into account in many components of a system, including the speech recognition grammars.

Implementation of Tutoring Agents

When tutoring functionality is implemented into a dialogue system, especially into an existing one, it is desirable that the tutor can be implemented without altering the existing base system. This way the base system can be available without the tutor and without any modifications. This also ensures that the tutored system does not differ from the base system. Such an implementation can be hard or even impossible with monolithic systems. Our tutoring implementation for the Mailman system and our model for tutoring agents are based on the Jaspis architecture [4]. The idea of several small agents controlling the dialogue in Jaspis proved to be very powerful when we implemented the tutor agents. No alterations were made to the actual Mailman program code. Instead, the tutoring implementation is just a set of new agents working in parallel to the agents of the Mailman system. Next, we introduce the principles of the system architecture behind the tutorial agents.

System Architecture

Jaspis is an architecture for spoken dialogue systems [4]. The tutoring agents use three of the main features of Jaspis: the shared Information Storage, evaluators, and small, stateless agents. The shared Information Storage is a component that all the other components in Jaspis based systems have access to. This is the storage where each component places all the information that they need to store. Information Storage can be seen as a kind of blackboard. It contains, for example, the dialogue state and the user model. Because of this, all the other components can be stateless, i.e. they do not store any information about the dialogue state in their internal structures. This way we can always freely choose an appropriate component for each situation.

The other components in the Jaspis architecture are agents and evaluators. Agents are the components that actually implement the system functionality. Evaluators are used to select the best agent for each situation. Each agent and evaluator works under one of the several managers in a system. For example, dialogue agents and dialogue evaluators work under the Dialogue Manager. Every time Dialogue Manager receives a turn, i.e. is able to do anything, all the agents are evaluated to find which one, if any, is the most capable of handling the situation at hand. The idea is for the agents to be very compact so that in each dialogue turn there can be a different dialogue agent, or even several, to take the turn. For example, in the base Mailman system (not including the tutor features) there are 27 different dialogue agents. There is a separate agent for each system function and several small agents for different situations such as the different phases of logging in.

Tutoring Agents

Our general tutorial implementation consists of a set of agents and evaluators. In dialogue management we have tutor agents for providing information to the user. Some give the user general hints and only take care that the hint comes in at an appropriate spot in the tutorial structure. Others give the user information about something concerning the current topic. When agents are evaluated, the state of the dialogue is examined to see if the situation is appropriate for each agent. The user model is also used to see if the information has already been taught to the user. If an agent receives a turn, it updates the user model to avoid receiving a turn again.

The turn taking decisions may include any kind of information available. However, excluding the user model, most of the decisions are based on four rules which all are based on information found in dialogue history. The base system and the tutor agents store their dialogue events in a common dialogue history. The tutor agent evaluation looks at the dialogue history for specific events. An agent may require the previous event to be of a specific type. Specific previous events may also prevent an agent from taking a turn. Furthermore, an agent may require a set of events to be found in the history and some events anywhere in history may also prevent an agent from taking a turn.

With these rules we have been able to make specific tutoring events happen at specific points in the dialogue, making sure that the structure of the dialogue and the tutorial stays coherent. We take care that the tutor does not teach something which requires knowledge not yet taught, does not teach something that the user already knows and makes sure that the turn is given to the user or the system at an appropriate time. It is also possible to write special rules into single agents on a case-to-case basis. The tutoring agents also have different values that they return for their ability to take turn even if all their conditions match. This way we can make an agent to take turn before another if conditions for both are met.

Special agents manage the user inputs. They receive turn when another agent has the asked user to give a specific input. In such a case the agent has stored information about the request to the Information Storage. The agent managing input looks at the user input and if it matches the requested one, it does not alter the input but instead gives the user confirmation about successful recognition and possibly some description of what the system will do next. If the input differs from what was asked, the agents discard the input, inform the user about the recognition result and ask user to try to give the command again. Alternatively, an agent can simply tell the user that the input was not an anticipated one and give advice to the user to listen to what happens next. Input managing agents also take turn when no special input was requested, but the system receives only silence or a recognition error occurs. In such cases the tutor instructs the user to speak at a correct time. An agent could also modify or create inputs. This agent can be used when we want to give the user an example, i.e. the tutor uses the system.

In addition to the aforementioned agents, the dialogue module includes a special tutor agent called "RestoreIOResults". It modifies the input/output results (user inputs etc.) so that other tutorial agents can co-exist with the normal dialogue agents without conflicts, the base implementation does not see the tutor behavior at all. When a tutor agent takes turn, it stores all input/output results available into the shared Information Storage. When the tutor agent has done its actions, "RestoreIOResults" modifies the information storage so that the normal dialogue agents can continue. This way the base system implementation does not need to take the things that are happening within the tutorial into consideration.

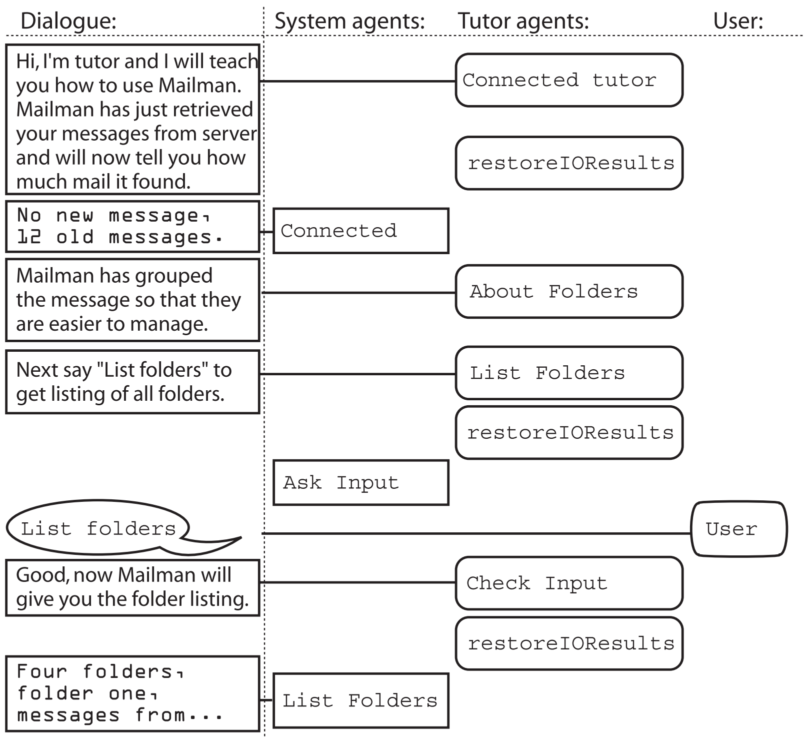
There is also a special tutor mode evaluator under Dialogue Manager. This evaluator takes care of when the tutorial mode is selected that the tutoring agents have priority over the normal agents. It does not block the normal agents from working as the normal system is supposed to work all the time. This evaluator can however block the tutorial agents from taking turn when the tutorial mode is turned off.

Tutorial Implementation in Mailman Application

We have an implementation of the integrated tutor in our Mailman application [3]. Mailman is a telephone operated speech interface to e-mail. It has a user-initiative speech interface that is used to browse the mailbox. The telephone keypad can also be used to access most of the system functionality. Mailman has user profiles and users must login to the system when they call. The user models enable us to use the tutor only during the first calls. Furthermore, Mailman is a user-initiative system and therefore requires explicit training to learn the commands; it is an excellent system for an integrated tutor. Mailman has also some higher-level concepts, like automatic e-mail message grouping that our tutor describes to the user.

The tutor is implemented as a separate dialogue partner identified by a different voice. The tutor also refers to itself as "tutor" and the system as "system" and "Mailman". It gives user guidance and hints during dialogue. In the beginning of the tutoring it requires the user to follow its tutoring plan and discards unwanted inputs, thus controlling dialogue to some extent. The user is not able to speak to the tutor but the tutor monitors the dialogue and uses this to adapt its own behavior.

The implementation includes several agents. There are about 30 tutoring utterances that are generated by the presentation agents. As we implemented the Mailman tutor as a separate person and because the set of different tutor outputs is limited, we decided to use recorded voice as our tutor voice. Therefore, the task of the presentation agents is rather straightforward. Most of the tutorial utterances are simple canned utterances and some context sensitive outputs feature simple slot-filling. For instance, an actual example of currently used grouping is used when the tutor explains the concept of mail folder. Furthermore, a special presentation agent was implemented to inform the user that Mailman understood something else than the tutor asked the user to say.


Dialogue:
System agents:
Tutor agents:
User:
Hi, I'm tutor and I will teach you how to use Mailman. Mailman has just retrieved your messages from server and will now tell you how much mail it found.
Connected tutor
restoreIOResults
No new message, 12 old messages.
Connected
Mailman has grouped the message so that they are easier to manage.
About Folders
Next say "List folders" to get listing of all folders.
List Folders
restoreIOResults
Ask Input
List folders
User
Good, now Mailman will give you the folder listing.
Check Input
restoreIOResults
Four folders, folder one, messages from...
List Folders

*Picture 1: **Dialogue and corresponding agents***

There are 25 dialogue agents in all. Most of them are basic information providing agents. Some provide information in specific situations such as right after login and during reading an e-mail message. Others just provide information whenever it is appropriate according to the state of the tutorial.

At the start of the tutoring, certain tutoring content is given in two sets of tutorial outputs. In a set, tutorial messages follow one another in a specified order. The first set describes to the user the concept of folders and how to browse them. The other set teaches browsing between e-mail messages and reading them. The first set is optional as is presented only if there are folders present. These sets include tutoring turns where the user is requested to give specific inputs. Each of these cases has a matching agent which checks if the given input indeed matches the requested one, gives the appropriate tutoring message and if necessary, discards the incorrect input before the Mailman system has a chance to process it.

Picture 1 contains an example where an excerpt of dialogue and corresponding agents are presented. The dialogue starts when a user has logged into the system and Mailman has retrieved e-mail messages from a mail server. This is also the point when the integrated tutoring comes into play. “ConnectedTutor” takes the turn and explains to the user the situation and what is going to happen. The next agent, “RestoreIOResults” takes the turn and removes the results of the tutor output messages and replaces them with whatever messages were present when the “ConnectedTutor” agent took turn. Next, the normal “Connected” agent of the Mailman system takes turn. It is followed by two tutor agents. The latter one asks the user to give a specific input. The tutor agent turn ends once again with “RestoreIOResults” agent and the system proceeds as usual, i.e. asking input from the user. When the user has given the input, the tutor checks that it was correct, gives approving output and once again “RestoreIOResults” receives turn, this time placing the user input back as the active input and the system reacts to the user’s command in the usual way.

The existing Mailman agents featured in this example are the agents of the original Mailman system. This is the case for the entire system; the original Mailman agents have not been modified in any way.

Discussion

Integrated tutoring as a concept seems to be a promising one. Our initial tests with the first version of the tutor provided positive comments. However, some problems were spotted as well. Mainly these concerned the users’ view of the state of the tutoring. We had to update the tutor to be more informative about the status of the tutorial. The latest version explicitly informs the user when the guided part of the tutoring is over and the user can start using the system freely. This was also a good spot for a summarization of covered subjects. Our plans for future work include usability testing of the Mailman tutoring functionality to find out how well the concept and implementation work in practice.

In the implementation we have encountered one problematic issue. The only signs about the tutoring agents that are left in Information Storage are the entries in the dialogue history. In our current implementation these entries are placed in the same dialogue history as the normal system events. Using the same dialogue history has caused us slight problems. These occur when Mailman uses the dialogue history and the user gives the ”repeat”, ”what next” or ”tell more” command. We had to override the corresponding agents with special tutor agents to avoid repeating the tutor messages. This practice is based on the idea that the user perceives the tutor as separate from the system and that the user can talk only to the system. Therefore, for example, the repeat command should repeat the last system output, never a tutor output. We would like to avoid any alterations to existing systems and therefore these cases are a bit problematic. Overriding existing agents could be avoided by maintaining a separate, yet corresponding dialogue history for the tutor events. Separate dialogue histories should be very handy especially in systems where the user can speak to a tutor and we have two parallel dialogues.

Conclusions

We have developed an integrated tutoring feature into spoken dialogue systems that teaches users how to use the system. Tutoring as a way to teach the system to the users has certain benefits. Users can start using the system right away without a need for reading manuals or similar external activities. We can also provide teaching in a real life context and we are able to check that the user really learns what we teach and adapt the teaching when problems occur.

In this paper we have described the implementation issues of integrated tutoring. The agent based approach of the Jaspis architecture proved extremely efficient from this point of view. We did not have to alter our original system at all. Also, the shared Information Storage played an important role. In all, the Jaspis architecture was a very efficient and easy platform to develop such new functionality into our existing system. This extendibility will enable us to continue our work with the tutoring and make it easy to extend our current tutor. For example, providing the user a possibility to talk to the tutor would require adding some new agents and another recognizer to the system. Once again, the existing components would not need to be altered in any way. The explicit evaluator system of the Jaspis architecture was also used, which makes it easy to turn the tutor on and off, for example.

References

[1] Yankelovich, N., “How Do Users Know What to Say?” *Interactions*, 3(6), 32-43, 1996.

[2] Kamm, C., Litman, D. and Walker, M.A., “From Novice to Expert: The Effect of Tutorials on User Expertise with Spoken Dialogue Systems”, *Proceedings of the International Conference on Spoken Language Processing*, (ICSLP98), 1998.

[3] Turunen, M. and Hakulinen, J., "Mailman - a Multilingual Speech-only E-mail Client based on an Adaptive Speech Application Framework", *Proceedings of Workshop on Multi-Lingual Speech Communication* (MSC 2000), 7-12, 2000.

[4] Turunen, M. and Hakulinen, J., “Jaspis - A Framework for Multilingual Adaptive Speech Applications”, *Proceedings of 6th International Conference of Spoken Language Processing* (ICSLP 2000), 2000.

[5] Nakatami, L.H., Egan, D.E., Ruedisueli, L.W., Hawley, P.M. and Lewart, D.K., “TNT: A Talking Tutor ‘N’ Trainer for Teaching the Use of Interactive Computer Systems”, *Proceedings of CHI’86*, 29-34, 1986

[6] Rickel, J., Ganeshan, R., Rich, C., Sidner, C.L. and Lesh, N., "Task-Oriented Tutorial Dialogue: Issues and Agents" *Working Notes of AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications*, 2000.

Paper III

Jaakko Hakulinen, Markku Turunen, Esa-Pekka Salonen, & Kari-Jouko Räihä. Tutor Design for Speech-Based Interfaces. In *Proceedings of DIS2004*, ACM Press, pages 155-164.

http://doi.acm.org/10.1145/1013115.1013137

Tutor Design for Speech-Based Interfaces

Jaakko Hakulinen, Markku Turunen, Esa-Pekka Salonen, Kari-Jouko Räihä
Tampere Unit for Computer-Human Interaction (TAUCHI)
Department of Computer Sciences
33014 University of Tampere, Finland
+358-3-215 {8558, 8559, 8873, 6952}
{Jaakko.Hakulinen, Markku.Turunen, Esa-Pekka.Salonen, Kari-Jouko.Raiha}@cs.uta.fi

Abstract
Speech-based applications commonly come with web-based or printed manuals. Alternatively, the dialogue can be designed so that users should be able to start using the application on their own. We studied an alternative approach, an integrated tutor. The tutor participates in the interaction when new users learn to use a speech-based system. It teaches the users how to operate the system and monitors user actions to be certain that the users do indeed learn. In this paper we describe our experiences with the design and the iterative development of an integrated tutor. Expert evaluation and two user tests were conducted with different versions of the tutor. The results show that the tutor can effectively guide new users. We identify the six most important lessons learned, the most important being that it is essential to spot problems by monitoring user actions, especially when novice users are tutored.

Categories and Subject Descriptors: H.5.2 [User Interfaces]: Voice I/O, Natural language, Training, help and documentation.

General Terms: Human Factors, Design, Documentation.

Keywords: Software tutoring, guidance, speech user interfaces, spoken dialogue systems, intelligent help systems.

INTRODUCTION

Speech-based interfaces share a common problem with textual, command-based interfaces: the interaction objects and commands are not visible. This creates the need to learn at least the basics of a system before using it. The trial and error approach, one of the advantages of direct manipulation graphical user interfaces, is out of the question.

In the 1970's, when textual interfaces were the dominant interface style, the usual tool for teaching the commands needed was a manual, either printed or electronic. It is well known that users do not like to read manuals, but want to start using a new tool as soon as possible. This often leads to suboptimal ways of using a software application [2].

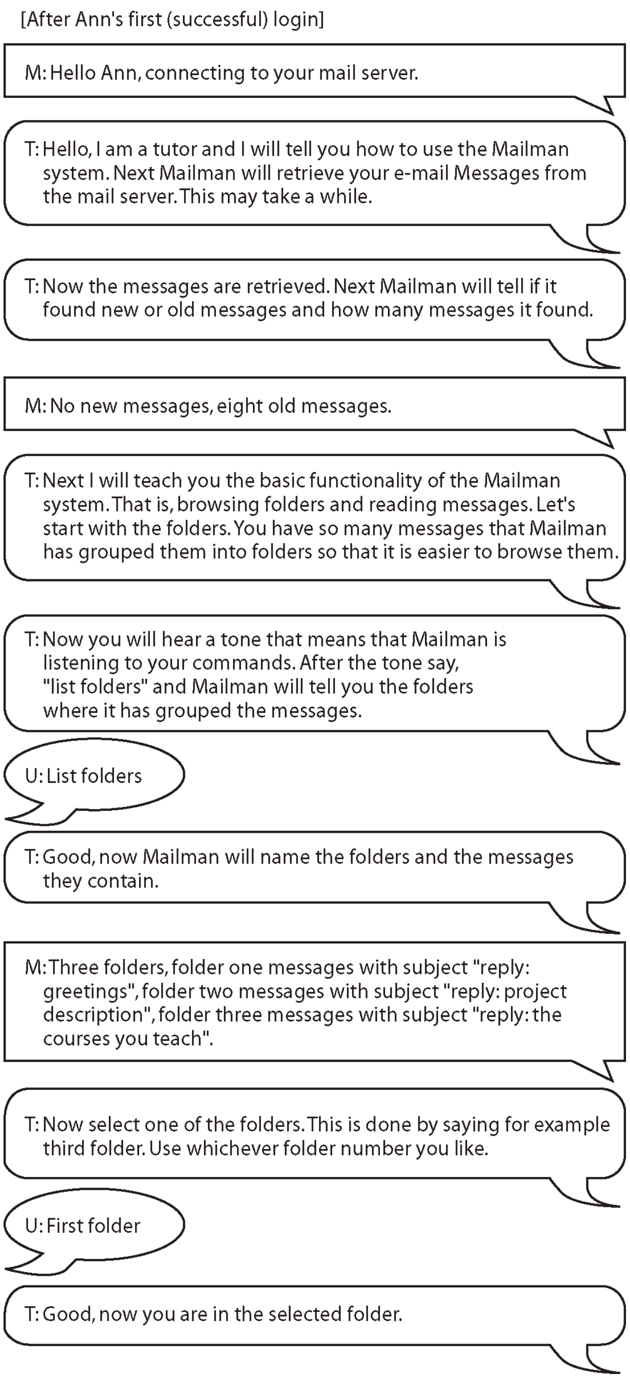
In the speech domain, paper manuals also suffer from an accessibility problem. Many speech-based systems are telephone-based and are used in places where manuals and web pages are not available. Therefore accessing offline documentation is often impossible. In addition, visually impaired users are common with speech-based applications and they may have difficulty accessing print media.

For visual user interfaces, manuals were soon accompanied by other more effective learning tools. Various kinds of embedded assistance [1], like AppleGuide [4], were developed. In addition, a range of tutoring programs, where software components guide new users, has been developed and even attempts to support automatic creation of tutors have been made [3]. Somewhat surprisingly, this approach has not been tried much with speech interfaces. Instead, speech-specific techniques have been developed for introducing the available functionality to users. These include techniques such as giving hints (longish prompts listing several possible commands in the current situation) and tapering (starting with long prompts, which are gradually shortened when the user presumably becomes familiar with the interaction style) [14].

Tutoring, provides potential advantages that cannot be achieved simply by prompt design. For instance, by using system prompts it is generally very hard to explain complex application functionality and the necessary concepts behind the interface. With a tutor we can provide more explanations and still combine these with actual usage.

It has also been shown that a (web based) tutorial can play a very important role in how new users receive a speech-based application. Kamm et al. [7] had two groups using a speech-based system. One group learned the system with only guiding system prompts, while another group received a short tutorial before starting to use the same system. Both groups eventually learned to use the system equally well, but the users in the tutored group had a better perception of the system afterwards.

The line between intelligently guiding prompts and tutoring may sometimes be hard to draw. However, a tutor is more powerful in the sense that it can explicitly tell a user what to do and then monitor if the user is capable of doing what was asked and provide feedback accordingly. Additionally, in tutoring teaching has a structure, an agenda that takes care that teaching proceeds in a meaningful order. Finally, the tutor can be a separate dialogue partner, parallel to the application, which makes it easy to refer to application functionality and features. On the other hand, this gives the interaction a radically new flavor: instead of a dialogue between two partners, a third actor monitoring the process steps into play, making interaction design more challenging.


[After Ann's first (successful) login]

M: Hello Ann, connecting to your mail server.

T: Hello, I am a tutor and I will tell you how to use the Mailman system. Next Mailman will retrieve your e-mail Messages from the mail server. This may take a while.

T: Now the messages are retrieved. Next Mailman will tell if it found new or old messages and how many messages it found.

M: No new messages, eight old messages.

T: Next I will teach you the basic functionality of the Mailman system. That is, browsing folders and reading messages. Let's start with the folders. You have so many messages that Mailman has grouped them into folders so that it is easier to browse them.

T: Now you will hear a tone that means that Mailman is listening to your commands. After the tone say, "list folders" and Mailman will tell you the folders where it has grouped the messages.

U: List folders

T: Good, now Mailman will name the folders and the messages they contain.

M: Three folders, folder one messages with subject "reply: greetings", folder two messages with subject "reply: project description", folder three messages with subject "reply: the courses you teach".

T: Now select one of the folders. This is done by saying for example third folder. Use whichever folder number you like.

U: First folder

T: Good, now you are in the selected folder.

Figure 1: Sample dialogue between the tutor, a user and Mailman.

We implemented an integrated tutor in our existing speech-based e-mail reading application, Mailman. The nature of the tutoring component is best described by an example. Figure 1 is an excerpt from an interaction between a user (U), Mailman (M) and the tutor (T).

Looking at the sample dialogue there are several things worth noticing. First of all, the tutor gives the user an explicit instruction on what to do next (*after the tone say "list folders"*). This is an important difference if tutoring is compared to just adding hints to a speech interface. The tutor also makes sure that the user is able to give the requested command successfully. Had there been, for example, a recognition error, the tutor would have acted differently, told the user about the problem and asked her to try again. It is also noteworthy that in the sample dialogue the tutor is a separate dialogue partner. In our implementation it has a voice different from the system's voice and it clearly draws a line between itself and the Mailman system ("*I am a tutor…*", "*next Mailman will…*" etc.).

In what follows, we first review the basic concepts of integrated tutoring (especially in the domain of speech-based interfaces), and describe the Mailman system and our tutor. Then we discuss the design decisions that were made during the iterative development of the tutor, pointing out pitfalls to watch out for. Our findings are collected into a list of lessons learned that can be used when developing integrated tutors for other speech-based systems. These include the fact that tutoring needs to be given in small units and at the right time and that monitoring users and providing more help when it seems necessary makes tutoring much more successful. We conclude by briefly describing preliminary results from a controlled user study where the integrated tutor was compared to web-based learning materials. The integrated tutor proved effective: all users learnt to use the system and when compared with web-based learning, they were able to start using the system much more fluently.

INTEGRATED TUTORING

As integrated tutoring has not been widely used in speech-based systems, our aim in this study is to gain insights into possibilities and potential problems present when tutoring is integrated into a speech user interface. The main contribution of this paper is a collection of the lessons we learned when iteratively developed a tutor for our Mailman system.

Software tutoring has been studied in the context of text and graphical interfaces. Several potential gains of tutoring have been identified in literature. Based mostly on the benefits of software tutoring as presented by Garcia [5] and the advantages of speech when used in software tutoring as presented by Nakatami et al. [9] we have gathered the following major benefits of software tutoring of speech-based systems:

- Learning happens in a meaningful context.
- Teaching can be synchronized to applications events.
- Users can try things out right away and learn by doing.
- A tutor can monitor user actions and make sure users learn successfully and do not make serious errors.
- Users can get supporting feedback on their performance.

- Teaching can be structured and presented to users in effective order. It is possible to guide new users very carefully in the beginning.

Using speech-only interfaces also involves challenges and obstacles. The nature of speech, being a slow channel, makes it hard to provide users with large amounts of information. Furthermore, the temporal and transient nature of speech is a challenge, as users cannot reread something presented earlier as they can with text.

Definition of tutoring

Before we move on to discuss the design of our own tutor, we define what we consider to be an integrated tutor. Briefly, an integrated tutor tutors a user while he or she is using the new application. Tutoring is context sensitive, the tutor takes the pedagogical initiative and guides the user through the system with a structured agenda, uses some explicit instructions and makes sure the user is learning. These features are discussed in more detail below.

The user is tutored to use an application while (s)he is using it. The actual application is running simultaneously with the tutor and the user is using the application. This way the user learns the system in a real environment and there is no need to transfer knowledge from a training platform to the actual application or switch between the application and the training material. While learning the system, the user can accomplish his or her real tasks. This way the learning is constantly anchored in real usage and we avoid the problem of lacking motivation due to irrelevant training tasks.

Guidance given according to what the user is doing. The basics are usually taught in a more strictly guiding tutoring agenda but soon the tutor can move into the background and give the user information only in the event of a relevant context. The tutor also takes advantage of a user model, tracking what the user has been taught and what the user has been able to do with the system. The same things are not retaught, unless the tutor thinks it is necessary.

The tutor can explicitly instruct the user in what to do. While system prompts may just tell the user what he or she *can* do next, the tutor can instruct the user *exactly* what to do. At first a tutor can guide a user through just one of many possible alternative ways of doing things to keep things simple. This way the user gets an easy start and knows how to begin.

The tutor can monitor user actions and adjust its behavior. The tutor can follow the user behavior and see if the user is having problems. When the tutor has asked the user to do something it is easy to spot problems, as the tutor knows what should happen. If the tutor asks the user to say something to the application and speech recognition returns something else with low recognition probabilities, we can assume e.g. that the user does not know when to talk to the system. In traditional, dialogue based intelligent tutoring systems the dialogue often consists of tasks that the tutor sets the student. In software tutoring the interaction can be working with user's own tasks, doing the things he or she wants to do. However, it is possible, and in more complex system possibly very efficient, to let the tutor set tasks for the user. The tutor can then follow what the user is doing and give the necessary help.

Teaching in integrated tutoring has a structure. Tutored subjects may be arranged into an agenda that defines their order. The order may not be fully specified but may, for example, contain threads of subjects taught in a certain order. This way we can make sure information is taught to the user in such an order that the user can build on previously acquired knowledge.

Content of tutoring

There are several things that we may need to teach to a user about a speech-based application. First of all users need to know the functionality of the system. They should know all the necessary functions and how to access them. Often systems have some restrictions that the user should be aware of. There may also be some higher-level concepts that the user should be familiar with to understand how the system really works. With tutoring we can possibly teach both the restrictions and the higher-level concepts in an appropriate context, for example point out a restriction when it is effective. The tutoring can make an application more transparent, and therefore easier for the user. This may also increase users' faith in a system.

In addition to the above mentioned, higher-level knowledge, we may need to teach the user the technical details on how to operate the system. We need to tell the user the actual commands that he or she can use. In speech systems vocabularies and natural language understanding are often somewhat limited and examples of valid input utterances are usually the most effective way to teach what one can say to a system. We may also need to teach when to speak. If a system does not support so called barge-in functionality, i.e. the system is listening to the users only when it is not speaking itself, this may be an important part of the tutoring. The tutor can also give examples on how to speak, e.g. not to hyperarticulate. The tutoring of how and when to speak can be made context sensitive, i.e. we give instructions only if the inputs we receive from the users are not what were expecting. Especially in error situations, the tutor can give the user very detailed help.

Next we will describe Mailman and the technical details of our tutor before moving to our experiences of tutor design.

MAILMAN

We decided to implement our tutor in the context of the Mailman system. We have developed several speech-based applications in our research group and Mailman [12] was selected since it is a fairly stable system, designed so that it requires users to get some training before they start to use the system and has a user model for each user.

Mailman is a telephone based e-mail reading application. Both speech input and telephone keys can be used to control it. Mailman has a recognition grammar of approximately 30 words and does not support barge-in. However, telephone keys can be used to interrupt Mailman's utterances. When a user calls Mailman, he or she is first asked to identify him or herself by entering a user code and a pass code with telephone keys. Mailman retrieves the copies of the user's new e-mail messages or old messages if there are no new messages in the user's mailbox. The mailbox in the server is left untouched so that Mailman works as a secondary e-mail reading method to a GUI e-mail client. If there are more than six messages in the current mailbox, Mailman automatically groups the messages into folders to make it easier to browse them.

Mailman is, after the login phase, a user initiative system. It does not ask the user any questions but the user must know the system functionality and give commands to the system.

Mailman functionality includes browsing the automatically generated message folders. Folders can be selected by referring to them by their ordinal numbers or with the words "previous" and "next". Inside the folders, messages can be selected and read in similar manner. Active messages can be read with plain "read" command. When mail is being read, the user can move back and forth inside messages using telephone keys. Most of the other commands are also available with telephone keys. Mailman functionality also includes the commands "help", "what next", "tell more", "repeat" and a key command to end the session with Mailman. Some synonyms are included in the recognition grammar so some commands can be accessed with two or more different phrases.

When messages are read, Mailman interprets some elements, like web addresses, and modifies them into a more comprehensible form before reading them to the user. The language of e-mail messages is automatically identified per paragraph and an appropriate speech synthesizer is used.

There are both English and Finnish language versions of the Mailman system. The differences between the versions are in the system prompts, recognition grammars, language understanding and language identification. Both language versions were used in the different phrases of the development and evaluation of the integrated tutor.

TUTOR

The tutor was implemented by adding new components for the Mailman system. Mailman runs on top of the Jaspis architecture [11]. The tutor is a collection slightly over 30 Jaspis agents added to the Mailman system next to the existing agents [6]. The existing code of Mailman did not need to be modified: the addition of new agents was enough.

The tutor can participate in a dialogue at any point, before and after any system output and user input. The tutor can also be aware of the system status. This is possible because of the shared information storage used in Jaspis. The tutoring components can monitor the inputs the system has received, modify or discard these inputs as necessary and even create new ones. The tutor can also monitor the system state, for example the dialogue history. Tutor components also update the dialogue history and in this way different tutor components can coordinate their actions and follow a tutoring plan.

In practice the tutor works by sending output messages at appropriate times. These outputs are recorded speech, possibly concatenated from several segments. When the tutor has asked the user to give a specific input it also monitors user inputs. The tutor discards erroneous inputs before Mailman can react to them and generates an appropriate tutoring message to the user. The tutor also makes sure that the tutoring messages are given so that they match the tutoring already given and the dialogue state in general. This way the structure and agenda of tutoring are maintained.

The Mailman implementation acts normally all the time when the tutor implementation is not overriding it. When the tutor is turned off or does not have anything to say, Mailman works as usual.

ITERATIVE DEVELOPMENT

We iteratively developed the Mailman tutor in three major phases, improving the system in each step. In the following the different versions and their evaluations are described and improvements are discussed.

The first version

The very first versions of the tutor used synthesized speech. However, as we decided at the very outset that the tutor would be a separate dialogue partner and there was very little dynamic content in its outputs, we started using a recorded tutor voice as soon as the prompt design was reasonably stable. A recorded voice was selected instead of speech synthesis for several reasons. It made the tutor sound different from Mailman and emphasized the design that the tutor was a separate character. Recording a human voice also gives greater control over the speech; for example, we could easily stress the important parts of the tutoring messages. In general, human voices are also easier to understand and more pleasant than most speech synthesizers. The first version of the tutor was Finnish-speaking. We recorded tutoring utterances using the voice of the developer of the tutor and it was clear that a better voice would be recorded when the tutor was further developed.

The basic structure of the tutor was such that when a user had logged in and messages had been retrieved, the tutor introduced itself to the user and explained how messages had been retrieved from the server and been automatically grouped in folders. After this the tutor was in a state where it told the user what to do. The first version required the user to give the very first command, *list folders,* exactly as requested and then verified that this had been done successfully. It also gave the user positive feedback by saying "*Good. Now Mailman will list the folders.*" After this, the tutor told the user what to say but allowed him or her to do something else at will. Then the tutor soon receded into the background and into a less directive mode, only giving hints at appropriate moments. The structure of the first version of the tutor is presented in Figure 2 in comparison to later versions of the tutor.

Evaluation

The first version was evaluated within our research group by researchers not involved in the implementation of the tutor. Volunteer researchers tried out the tutor and provided suggestions for improving it. All the testers had experience with Mailman, most had taken part in its development and their feedback was given from an expert point of view.

The most important negative feedback received was that the state of the tutoring was unclear to users. As the tutor gradually receded into the background, it was unclear how much more tutoring there would be. It was a generally agreed opinion that the tutor should provide users with information about the structure of the tutoring and the amount of tutoring content left.

There was also a request to separate the tutoring content more clearly from Mailman outputs. Just using a separate, recorded voice was not considered to be a strong enough cue. Positive comments included the fact that the positive feedback the tutor provided was a well functioning feature. The structure and style of guidance seemed to be appropriate as well. Nevertheless there were comments that the tutor messages should be carefully recorded so that they are easy to comprehend and pleasant to listen to.

The second version

In the second version we clarified the structure of the tutor. We extended the directive part of the tutoring to cover the basic functionality of the system, that is, browsing folders and messages within a folder, and reading messages. In the beginning, after the introduction, the tutor told the user that it would next cover the basic system functionality. After this it forced the user to go through the five most important functions. When this phase was over, the tutor gave the user a summary and told that it would move into the background and only give hints when considered appropriate and that the user could now use Mailman freely. Tutoring was also modified so that it started at the earliest possible moment, right after the login. We also included a brief tone at the start of each tutoring message so that users could immediately identify if an output was from Mailman or from the tutor. The second version was in English and worked with the English language version of Mailman. The voice was recorded by a native English-speaking researcher and lecturer working at the department at the time. The example in the introduction is a sample from an interaction with the second version.

Evaluation

In the second phase we had eight users new to Mailman testing the system and the tutor in a controlled experiment. The users were participants on an introductory user interface course and participating in the test was a voluntary assignment for the course. All the participants were foreign students; none was a native English (or Finnish) speaker. The native languages of the participants were Chinese, Czech, Thai, German and Greek and their proficiency in English varied significantly. This group was a very challenging one for the tutor since they had potential language comprehension problems, especially with speech synthesis. More importantly, they had deviances in their pronunciation of English so that speech recognition errors were likely.

In general, the tutor worked well with these users. All the users were able to use Mailman and carry out at least some of the tasks assigned to them. The biggest problems found were not those of the tutor but instead concerned Mailman. Novice users brought to light issues that had not been noticed by the experts and several usability problems were detected in Mailman. Most importantly, Mailman had very short timeouts when users were silent. This was due to very primitive voice activity detection and a more flexible solution was included in Mailman during the tests. We also fixed some recognition grammar inconsistencies during the tests and modified Mailman's interaction style to be slightly more communicative to smooth out the interaction. This experience proved to us that the tutor is not able to solve usability problems of the tutored system.

There were great differences between the users. Some considered the tutor quite boring while others did not report such feelings. The tutoring voice used spoke very clearly and slowly so it is understandable that fast learners at times felt somewhat bored. Some users also noted that there were rather long delays in the system and tutoring.

Several users requested some form of repeat functionality so that they could hear tutoring messages again if they had problems understanding or remembering everything. There were also requests for the option to activate the tutor whenever a user had some problems, and an option to control the verbosity and the amount of tutoring according to user needs. Positive comments commended the context sensitivity of the tutoring messages and stated that the tutor taught the basic usage of Mailman quickly. The tutor was also considered to be easy to understand and the combination of learning and doing was seen as a positive feature.

The most important single finding concerning the tutor design was, however, that a tutoring message teaching two different things not connected to each other caused problems. The tutoring message was:

> Next say read and Mailman will read you the selected message. You can stop the reading by pressing the zero key in your telephone keypad.

Even this simple message caused confusion and the zero-key was not learned or users even confused the two things and pressed the zero-key right away. Some of the confusion was probably due to language proficiency issues, but supposedly just highlighted the real problem that would also have been experienced by users with better language skills.

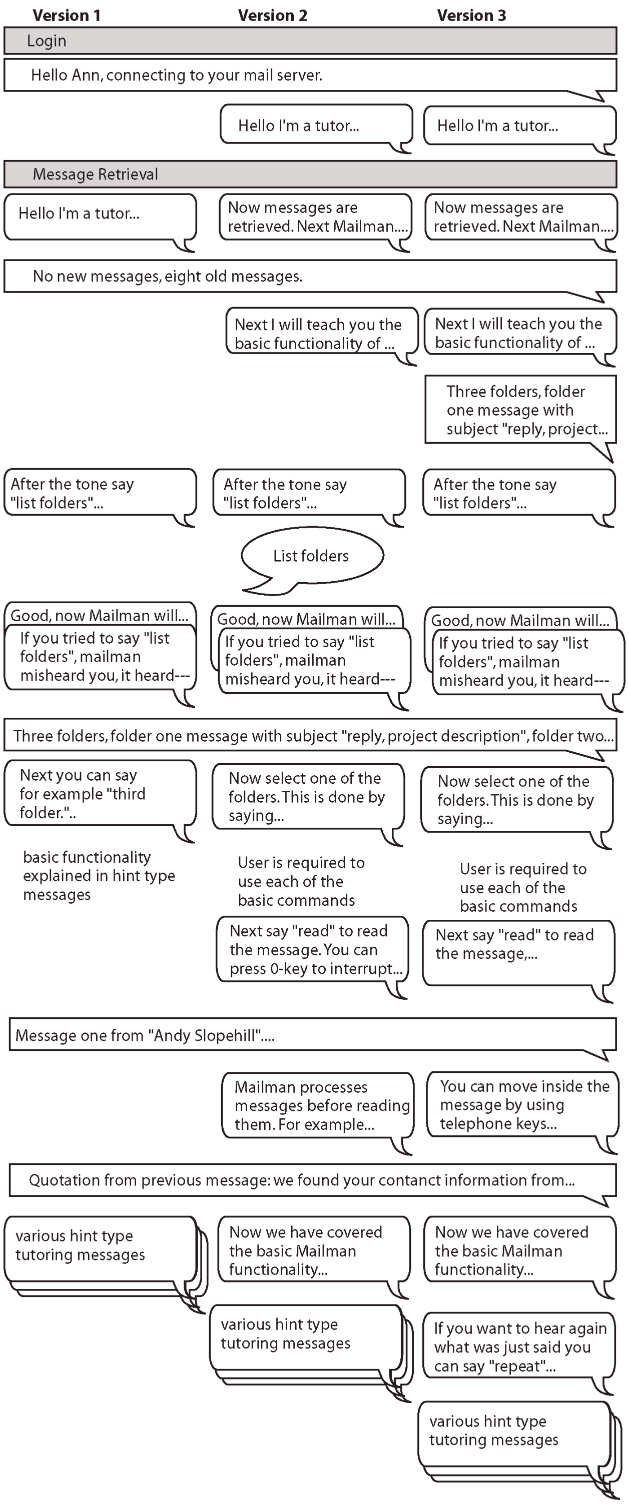

Version 1
Version 2
Version 3
Login
Hello Ann, connecting to your mail server.
Hello I'm a tutor...
Hello I'm a tutor...
Message Retrieval
Hello I'm a tutor...
Now messages are retrieved. Next Mailman....
Now messages are retrieved. Next Mailman....
No new messages, eight old messages.
Next I will teach you the basic functionality of ...
Next I will teach you the basic functionality of ...
Three folders, folder one message with subject "reply, project...
After the tone say "list folders"...
After the tone say "list folders"...
After the tone say "list folders"...
List folders
Good, now Mailman will...
If you tried to say "list folders", mailman misheard you, it heard---
Good, now Mailman will...
If you tried to say "list folders", mailman misheard you, it heard---
Good, now Mailman will...
If you tried to say "list folders", mailman misheard you, it heard---
Three folders, folder one message with subject "reply, project description", folder two...
Next you can say for example "third folder."..
Now select one of the folders. This is done by saying...
Now select one of the folders. This is done by saying...
basic functionality explained in hint type messages
User is required to use each of the basic commands
User is required to use each of the basic commands
Next say "read" to read the message. You can press 0-key to interrupt...
Next say "read" to read the message,...
Message one from "Andy Slopehill"....
Mailman processes messages before reading them. For example...
You can move inside the message by using telephone keys...
Quotation from previous message: we found your contanct information from...
various hint type tutoring messages
Now we have covered the basic Mailman functionality...
Now we have covered the basic Mailman functionality...
various hint type tutoring messages
If you want to hear again what was just said you can say "repeat"...
various hint type tutoring messages

Figure 2: Differences between the tutor versions.

The third version

In the third version some of the tutoring content was restructured so that now each tutoring message only taught one subject. This version was once again in Finnish and this time the tutor voice was recorded using a computer scientist who is also an amateur actor. The recorded voice was rather slow, like the previous one, but the style was more relaxed and a bit more easy-going.

At this point the tutor had evolved to its current phase, with two distinct parts. In the beginning of the first call, a user is required to follow instructions given by the tutor until the basic Mailman functionality is covered. This part includes a message right after login introducing the tutor and informing the user about message retrieval. Next the tutor tells the user that it will next teach the basics of the Mailman system. The concept of automatically generated folders is then introduced and the tutor asks the user to use the "list folders" command. If the user does not succeed in listing folders when requested, the tutor informs the user and asks her to try again until she successfully gives the required command. When the user gives the requested command successfully, the tutor gives positive feedback and lets Mailman follow the user's command.

After this, a similar technique is used when the user is requested to select a folder, select a message and read it. When the message is being read, the user is told how to browse inside the message. When the message has been read (or the user has cancelled the reading), the tutor gives a summary message reminding the user about the commands taught and states that this was the basic functionality and now the user can freely use Mailman. This first part of tutoring takes five to ten minutes to go through.

The tutor proceeds to state that it will give the user more info at appropriate spots. These additional tutoring messages teach the user the rest of the available speech commands and the most important key commands. The user is also informed, with an example, about the modifications made to e-mail messages to make them understandable in spoken form. Finally, the user is tutored on how to access the system's own help functionality. Using this functionality the user can learn the rest of the key commands and obtain some general information. It takes about 20 minutes of Mailman use to cover all the tutoring material.

A major modification from the second version of the tutor was the added support for the repeat command. Mailman already featured a repeat command, but in the previous tutor versions this command was implemented so that it repeated the previous system output and the tutor messages could not be repeated. The idea was that the user could not speak to the tutor and commands only controlled Mailman. This was altered and now the repeat command repeated the last output, whether it was a system or a tutor output. A tutor message informing users about this repetition option was added and placed on the tutoring agenda so that it arrived before any long tutor message, such as the description of keyboard commands, was given.

Evaluation

The third version was tested with nine native Finnish speakers. All the participants learned to use Mailman without problems and successfully worked with a set of tasks assigned to them.

One interesting problem arose with the third version. Numerous users spoke to the system so softly that the voice activity detection (VAD) in the system did not react. This is mainly because of the settings of the VAD were not optimal for the quiet office setting in which the tests were done.

However, the quiet speaking style of the tutor also seemed to make the users speak quite softly. People naturally adjust to the way other people behave and in this case the users seemed to naturally mimic the speaking style of the tutor. This problem was a serious one, since at this point the tutor did not include any timeout functionality. If the users spoke too softly and then just listened and waited quietly they did not receive any feedback. At the very beginning of interaction the users were unaware of what had happened and did not easily speak again but rather concluded that perhaps the system had crashed.

We collected feedback from the users and one major request was to increase interactivity. With the sole exception of the repeat command, the users could not in any way control what the tutor did, how much tutoring was given or when tutoring was given. In their comments, the users asked for things like “more info about this”. They also wanted instructions right at the beginning on what to do if they had problems at any point.

The most problematic tutoring content was considered to be a long message listing many keyboard commands. The message, taken from the second version, was:

> You can also control Mailman with telephone keys. The top row of keys selects folders. You get a folder listing by pressing key one and select previous and next folders with keys two and three.
> Messages are selected with keys in the second row. Messages listing is key four, previous message key five and next message six.
> When you have found the mail you want, key eight reads the message.

Although carefully written, the message was far too long and contained too many things to learn. The users were able to hear the message again but very few users did so.

The users also felt that the hints in the latter part of the tutoring session were presented in rather random situations. A negative comment about the summary repeating already familiar things was also received, but comparing this to the fact that one user had major troubles when she did not hear the summary because of disturbing noise suggests that summaries are probably rather a good than a bad feature.

There was also a technical problem with the speed of the system. System response times were awkwardly slow, usually causing a silence of 3 to 3.5 seconds in the interaction. Some users commented on this problem but it did not cause any actual problems for the users working with the tutor.

Positive feedback included the fact that one could try out the taught commands immediately. It was also reported that tutoring was well structured and split well into separate tutoring utterances. Examples were considered to be good and basic functionality was systematically covered.

LESSONS LEARNED

During the test and experiments with the various versions of the integrated tutor for the Mailman system we have learned a lot. There were many successful design decisions in our tutor but we have also come up with many ideas that could be used to improve it. Below is a summary of the most important findings to consider when integrated tutors for speech-based systems are implemented:

Teach one thing at a time. Users have to concentrate a lot when they are taught many separate things at the same time. It is noteworthy that one concept may concern several commands. For example, a tutoring message where we taught that messages and folders could be browsed with the words “next” and “previous” was one tutoring message and users efficiently learned all the resulting four commands. On the other hand, combining just two simple instructions, how to read a message and how to interrupt reading caused problems. In general, lists should be avoided when designing tutoring content.

At the start tell users explicitly what to do. This way a user can easily get something done and learn the style of interaction supported by the system. Users should always have the opportunity to try things out themselves. When a user-initiated function is taught, there should be a change to try it out without a delay. More general concepts should be tutored in a situation where a user hears an example of the phenomenon right after the tutoring.

Try to detect problems. If a user is explicitly told what to do, this should be easy, as the tutor knows what should happen when everything goes fine. This way it is possible to make sure that each user is able to successfully use the system. In the beginning we can also provide additional guidance every time there is a sign of a problem, for example we can have much shorter time-outs in the system during the first few minutes. A new user should never be left alone in trouble but rather given too many instructions. A little annoyance is better than a total failure.

Signpost the state of the tutoring. If users do not know how much more tutoring is still to come they get annoyed and possibly confused. Summaries are good ways to inform users about the status of tutoring and at the same time support learning. Just as users should know the state of an application, the tutoring status should be clear to them.

Use realistic examples in teaching. The examples should be accurate down to the level of speaking style. For example, there should not be hyperarticulation in the examples if the system cannot handle such speech. When more abstract concepts are taught, it should be done in context and by showing examples of how the idea becomes concrete in the system. Users can easily generalize from this. Conversely, teaching users generic rules first does not work very well. Users are better at induction than at deduction. Good use of context can also facilitate learning and makes things fall into place. Out of context tutoring annoys people even if it may still be effective [8].

Give users opportunities to control tutoring. This control can be explicit by the users giving commands to the tutor, or a more subtly by the tutor that monitors the users' interaction with the system. Users have different skill levels and needs so there should be a way to skip unnecessary tutoring and receive more help when in trouble. Due to the transient nature of speech, users may also miss some tutoring content. Make sure that users know what to do if they get confused. One way is to tell the users how to access the system's own help functionality. In our tests the users successfully used the tutoring and help commands together when they were first tutored about the help functionality.

DISCUSSION

The concept of integrated tutoring in a speech-only domain seems to be a working one. Our experiences with the Mailman tutor have been positive; users completely new to Mailman and speech-based systems have in general been able to learn all important Mailman functionality during 20 minutes of tutored use.

There were many potential problems with the concept of integrated tutoring in a speech-only environment. Big challenge is the fact that both tutoring and the system had to fit into the same sequential and rather slow output channel. Furthermore, audio is transient so users cannot just go back to something presented previously. However, none of these risks constituted serious problems in our tutor design.

This paper presented our experiences from just one tutor. The Mailman application is very suitable for tutoring. It is command-based and the dialogue is completely user initiative, which makes some sort of teaching absolutely indispensable. Mailman also has user profiles and, as user models are available, it is easy to present tutoring to new users and skip it for old users or let them continue from where they were. Not all applications are so favorable for a tutor. If a system is used only once per user the type of tutor design presented will probably be out of the question.

The lessons learned are likely to apply to other types of tutors as well. Most of the ideas presented here can be found in other sources and disciplines close to the subject. Presenting a little information at a time is a strengthened version of the general speech interface guideline of keeping things simple (see e.g. [10]). In tutoring things need to be even simpler than in a general interface, as it is not sufficient just to remember a set of options – they should also be learnt. In tutoring, simplifying things can sometimes override another speech interface design guideline of keeping things short, as longer explanations can give users time to understand what they are hearing.

Learning by doing is one of the fundamental reasons why tutoring works well. Another strong point is the fact that tutors can monitor user actions and take corrective actions when necessary. In integrated tutoring it is even possible to provide the user with a safe learning environment [9] where a tutor blocks potentially harmful functionality from beginners to provide an environment where they can learn without a risk of fatal mistakes.

Teaching in context and interactivity of the guidance are also issues addressed in the research of intelligent tutoring systems and could be further used in the design of integrated tutors. For example, the fact that students seem to learn well when they make errors [13] is interesting, as in traditional interface design we want to minimize the probability of errors. In the design of integrated tutors the most important thing about errors is, however, that the tutor must notice when a user makes an error or has problems tell the user that she is acting in a way the application is not capable of handling.

The benefits of tutoring can turn into disadvantages if we are not able to detect all the problems, detect them erroneously or provide guidance at inappropriate times. We were able to achieve most of the benefits to a reasonable extent. The biggest problems were in finding a meaningful context for each tutoring event. This caused some of the tutoring to appear to the users in rather random places. The greatest benefit was that we were able to monitor the users and spot most of the problems, and so ensure that all the users could learn how to use the system.

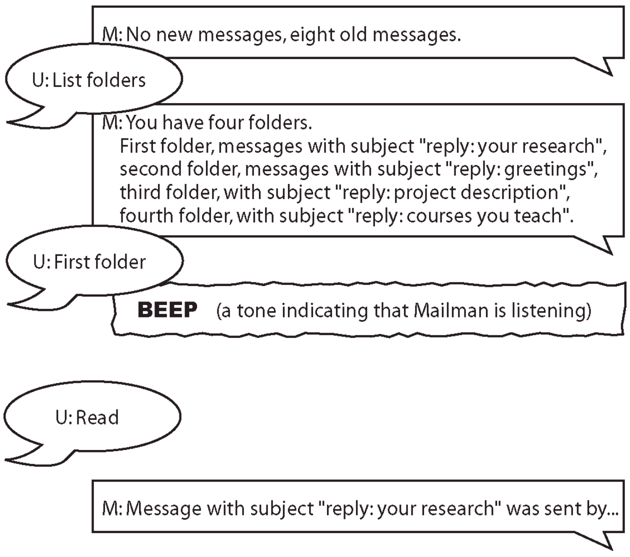

M: No new messages, eight old messages.
U: List folders
M: You have four folders.
First folder, messages with subject "reply: your research",
second folder, messages with subject "reply: greetings",
third folder, with subject "reply: project description",
fourth folder, with subject "reply: courses you teach".
U: First folder
BEEP (a tone indicating that Mailman is listening)
U: Read
M: Message with subject "reply: your research" was sent by...

Figure 3: A troublesome interaction without a tutor.

Figure 3 shows a dialogue from a test where a user was learning to use Mailman with a traditional web manual. The user knows what to say to the system, and all the commands she gives are legal inputs. However, she has not learned when to speak to the system. It is also unclear to her what the Mailman responses to the commands should be. The user gave her first command during a short pause between two Mailman utterances and as the latter utterance matched her command, she probably thought that everything went fine. However, her next command did not result in any meaningful response, as Mailman did not hear it, just as it did not hear the first one. The user learned the interaction style of Mailman eventually but it took her several minutes of usage where she did not have a clear picture of which one of her commands were successful and which failed.

The web pages mentioned several times that users should talk to Mailman only after hearing a tone, but many users had serious problems realizing this. When the tutor taught the users when to talk, every user learned it successfully. None of the tutored users suffered from the kind of problems described above. Teaching in a realistic context, during real usage and monitoring user actions ensured that all the users learned the interaction style of Mailman.

Figure 4 shows an example from a dialogue, one of the very few, where a user has problems at the start of a tutored dialogue. We see how the tutor is able to assist the user. As it has given the user an explicit instruction what to do next, it is able to spot that the user has some problems. The action taken by the tutor is successful, even though the tutor does not exactly understand what problem the user is having. Merely identifying that there is some sort of problem is enough to make the tutor to take corrective action.

However, in our tutor implementation we still missed some problems the users had, namely the situations where users spoke too softly. Developing a tutor is not as trivial as writing a manual, and the integration of a tutor may even be impossible in the case of some systems with a monolithic structure.

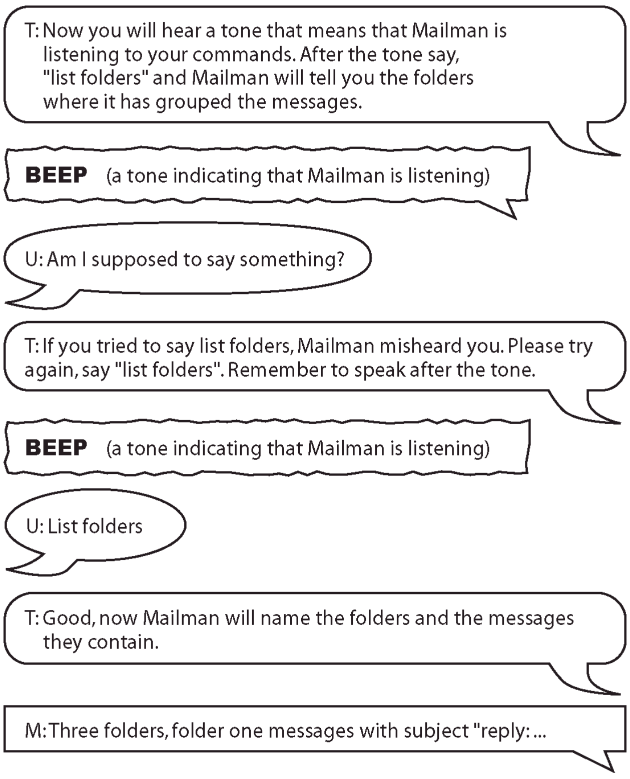

T: Now you will hear a tone that means that Mailman is listening to your commands. After the tone say, "list folders" and Mailman will tell you the folders where it has grouped the messages.
BEEP (a tone indicating that Mailman is listening)
U: Am I supposed to say something?
T: If you tried to say list folders, Mailman misheard you. Please try again, say "list folders". Remember to speak after the tone.
BEEP (a tone indicating that Mailman is listening)
U: List folders
T: Good, now Mailman will name the folders and the messages they contain.
M: Three folders, folder one messages with subject "reply: ...

Figure 4: Tutor recognizes a problem.

The viability of the whole integrated tutoring concept depends heavily on the type of application that is the target of the tutoring. We have summarized some differences between different types of guidance in speech-based systems in the table found in Figure 5.

	Manuals	Extended Prompts	Tutoring
Learning style	Studying separate material, then trying out with the application	Working with the application, possibly by trial and error	Learning by doing with real tasks and synchronized guidance
Feedback from learning	Feedback after trying things out	Feedback immediate after experiment	Feedback immediate and reinforced by the tutor
Spotting errors	User spots errors	User spots errors	Tutor and user can both spot errors
User's role	User is a passive student	User actively experimenting	User is actively working and receives guidance when necessary
Teaching higher level concepts	High level concepts can be covered with lengthy explanations	Only low level concepts are usually referred in guidance	High level concepts can be referenced and explained with real examples
Learning and application	Separate guidance material	Learning in real context	Learning in real context
Role of guidance	Passive guidance	Mostly passive guidance	Active guidance

Figure 5: Some differences between the guidance types.

The evaluations presented here were mostly usability tests detecting the problems of the tutor design and opinions on the tutoring. In total 17 users participated in the tests and learned to use Mailman with different versions of the tutor. The considerable variability among users has exposed different types of problems. The evaluations have also proved that the idea of tutoring does work in the context presented. A comparison of tutoring to a conventional manual is not presented here. We have this kind of evaluation underway and preliminary results suggest that there is no difference in the learning results. However, the users without the tutor seem to have many more problems when they first try to use Mailman. Strict guidance given by the tutor in the beginning takes a good care that each and every user learns the basics of Mailman. With a conventional manual there is more likelihood of problems and misunderstandings.

As the experiences suggest that greater interactivity could improve the tutor, in the future it would be interesting to develop a tutor enabling direct user-tutor interaction. This would cause the dialogue to become truly a multiparty one and raise totally new kinds of questions.

CONCLUSIONS

In this paper we have provided a proof of concept type presentation of integrated tutoring for speech-based applications. One successful tutor was iteratively developed and it could, and possibly will be in real use. Various lessons learned during the development of the tutor were presented and discussed.

In one sense designing a tutor for a speech-based application is even more challenging than designing the application itself. We must make sure that a user really learns what we are teaching. On the other hand, some leeway is allowed. Things do not need to be as concise as in the actual application, as most of the tutoring will be given just once and at that point the user is probably very interested in it. All in all, based on our experiences, we can recommend tutoring for teaching a speech-based application to new users.

The most positive experience during the development of the tutor is perhaps that fact that with the final version of the tutor each and every user learned to use Mailman successfully. Even users with no prior experience of speech-based systems and rather poor computing skills were using Mailman without problems after they had received tutoring.

ACKNOWLEDGMENTS

We thank Matias Hasu and Scott MacKenzie for providing us with their voices for the different tutor versions, Nuance for providing us with an academic license for the recognizer used in the English version of Mailman and the rest of the SPI group for everything. The TISE graduate school and the Academy of Finland (project 178099) funded this research. In addition some work was done in the DUMAS project funded by the EU.

REFERENCES

1. Ames, A. L. Just what they need, just when they need it: an introduction to embedded assistance. In *Proceedings of the 19th Annual International Conference on Computer Documentation.* ACM Press, New York, NY, 2001, 111-115.
2. Carroll, J. M. and Rosson, M. B. Paradox of the Active User. In Carroll John, M. (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction.* Cambridge, MIT Press, MA, 1987.
3. Contreras, J., Saiz, F. A Framework for the Automatic Generation of Software tutoring. In *Proceedings of Computer-Aided Design of User Interfaces*, 1996.
4. Esteban J. AppleGuide. *Interactions*, 3, 3 (May 1996), 36-37.
5. García, F. CACTUS: Automated Tutorial Course Generation for Software Applications. In *Intelligent User Interfaces 2000 (IUI'2000).* ACM Press, New York, 2000, 113-120.
6. Hakulinen, J., Turunen, M. and Salonen, E-P. Agents for Integrated Tutoring in Spoken Dialogue Systems. In *Proceedings of Eurospeech 2003*. 757-760.
7. Kamm, C., Litman, D. and Walker, M.A. From Novice to Expert: The Effect of Tutorials on User Expertise with Spoken Dialogue Systems. In *Proceedings of the International Conference on Spoken Language Processing, (ICSLP98).* ASSTA, 1998, 1211-1214.
8. Mackay, W. E. Does Tutoring Really Have to be Intelligent? In *CHI2001 Extended Abstracts*, 2001.
9. Nakatami, L.H., Egan, D.E., Ruedisueli, L.W., Hawley, P.M. and Lewart, D.K. TNT: A Talking Tutor 'N' Trainer for Teaching the Use of Interactive Computer Systems, In *Proceedings of CHI'86*, 1986, 29-34.
10. Suhm, B. Towards Best Practices for Speech User Interface Design. In *Proceedings of EuroSpeech 2003*. 2217-2220.
11. Turunen, M. and Hakulinen, J. Jaspis - A Framework for Multilingual Adaptive Speech Applications. In *Proceedings of the 6th International Conference of Spoken Language Processing (ICSLP'2000)*, 2000.
12. Turunen, M. and Hakulinen, J. Mailman - a Multilingual Speech-only E-mail Client based on an Adaptive Speech Application Framework. In *Proceedings of Workshop on Multi-Lingual Speech Communication (MSC 2000).* 7-12.
13. VanLehn, K., Siler, S., Murray, C. and Baggett, W. What makes a tutorial event effective? In *Proceedings of the Twenty-first Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum. 1998, 1084-1089.
14. Yankelovich, N. How Do Users Know What to Say?" *Interactions*, 3, 6 (December 1996), 32-43.

Paper IV

Jaakko Hakulinen, Markku Turunen, & Kari-Jouko Räihä. Evaluation of Software Tutoring for a Speech Interface. *International Journal of Speech Technology*, 8(3), pages 283-293.

INTERNATIONAL JOURNAL OF SPEECH TECHNOLOGY 8, 283–293, 2005

Evaluation of Software Tutoring for a Speech Interface

JAAKKO HAKULINEN, MARKKU TURUNEN AND KARI-JOUKO RÄIHÄ
Tampere Unit for Computer-Human Interaction (TAUCHI), Department of Computer Sciences, University of Tampere, FI-33014 University of Tampere, Finland
Jaakko.Hakulinen@cs.uta.fi
Markku.Turunen@cs.uta.fi
Kari-Jouko.Raiha@cs.uta.fi

Received May 5, 2004; Revised January 10, 2006; Accepted January 13, 2006

Abstract. Speech applications are typically designed to be used without any instructions or manuals. More complex applications commonly come with web-based or printed manuals. An alternative approach, software tutoring, has been studied in the context of graphical user interfaces. In software tutoring, a software component guides users while they work with an application new to them. To evaluate the viability of software tutoring in speech-based applications a two-condition between-participants experiment ($N = 18$) was conducted. Participants learned to use a speech-based e-mail reading application and performed several tasks with it. In the first condition the e-mail application included an embedded tutoring component that guided the participants on using the application. In the second condition, a web manual was used. All interactions with the systems were recorded and annotated. Participants also filled in questionnaires that reported their attitudes towards the guidance they received and towards the e-mail reading application. The speech-based tutor performed equally well with web-based manual with no significant differences between the conditions on how well the participants managed to accomplish the tasks with the e-mail application or in participants' attitudes towards the application or the guidance. In addition, during the learning period the participants in the tutored condition had significantly fewer problems with the speech interface.

Keywords: software tutoring, guidance, speech interfaces

Introduction

In speech-based interfaces interaction objects and commands are not visible to users. Since current speech applications support only limited application domains, recognition grammars and interaction styles, it is necessary that users receive some sort of guidance before they can effectively use applications. The ultimate speech interface that is as easy to use as speaking to a person is not reality now or in the near future. Therefore, to make successful speech interfaces we need to put effort on effective user guidance (Heisterkamp, 2003).

Traditionally, the necessary guidance has been embedded into the interface, usually with techniques like hints, tapering and prompts of varying explicitness (Yankelowich, 1996). These techniques can guide users through a system, but it can be hard to provide the users with a thorough picture of the system. Kamm et al. (1998) have found out that even if users are able to learn to use an application with just embedded instructions, a short tutorial can increase their perception of the quality of the system.

More complex applications commonly come with some sort of print or web manual. The manuals have a problem that was identified as the paradox of active users by Carroll and Rosson (1987). They found that users do not like to read manuals, but instead want to get their hands on action as soon as possible. This often leads to suboptimal ways of working with software applications. Conventional manuals may also have accessibility problems. In the case of speech interfaces, one important user group is visually impaired users. Conventional paper manuals may be unusable to many of them.

For visual interfaces, various forms of efficient and accessible embedded assistance have been introduced including web manuals, effective online help (Esteban, 1996), wizards, information rich interfaces (see e.g. Ames, 2001) and critiquing systems (Fischer et al., 1990). At the same time the field of intelligent tutoring systems (ITS) has been widely studied. The developing technology has made it possible to use speech in dialogue based ITS applications (e.g. Fry et al., 2001) and their possible benefits are being studied (Rosé et al., 2003).

The combination of embedded assistance and intelligent tutoring results in software tutoring systems. These are software components that teach users how to use a software application. The idea of software tutoring is that a user is using the application he/she wants to learn. A software component serving as the user's tutor is monitoring the user's actions and can provide appropriate feedback. The tutor can also give the user explicit instructions on how to proceed and what to do next. For example, García (2000) has discussed the concept and introduced a method for automatic generation of software tutoring.

Speech has been used in software tutoring of visual interfaces e.g. by Nakatami et al. (1986). Based on the discussions by García and Nakatami et al. we summarise the following advantages of software tutoring compared to conventional manuals.

– Learning happens in a meaningful context and users work on their real tasks.
– Teaching can be synchronised to application events.
– Users can try things out right away and learn by doing.
– A tutor can monitor user actions and make sure users learn successfully and do not make serious errors.
– Users can get supporting feedback on their performance.
– Teaching can be structured and presented to users in an effective order. It is possible to guide new users very carefully in the beginning.

These benefits can best be realised with tutoring. Commonly used extended prompts can only provide users with a list of possibilities or some well-selected examples. A tutor on the other hand can guide a user through a system by explicitly telling the user what to do next. This gives the tutor an opportunity to effectively monitor the user, find out if the user has problems and provide feedback and additional guidance as necessary. Since a tutor can have an active role in the interaction, it is also possible to explain complex functionality and concepts behind the application logic much better with tutoring than with extended prompts.

So far software tutoring has not been extensively used in the context of speech only applications. The nature of speech brings up issues that are very challenging when tutoring is integrated into an application. Speech is a slow output channel, and when speech interfaces are designed brevity is usually desired. Tutoring, especially when realised with speech, is inevitably something that slows down the interaction. As users should interact with the real, unmodified system while being tutored, we cannot shorten the system prompts. At the same time a lot of additional information needs to be presented in the same output channel. This can lead to slow and frustrating user experience.

Using the same output channel may also make it hard for users to separate the system and the tutoring content. If this happens, users may feel confused when tutoring is withdrawn. If the tutoring is presented as a completely separate dialogue partner to differentiate it from the application, the dialogue turns into a (non-symmetric) three party dialogue. This complicates the interaction and can cause confusion for users.

Finally, the transient nature of speech also makes tutoring content transient. Users cannot just reread something they missed or have forgotten. If not carefully designed, tutoring can become practically useless.

As discussed above, software tutoring has several potential benefits, but also many reasons to question the viability of tutoring in the context of speech applications. If we can avoid the problems and successfully introduce software tutoring in speech interfaces, it can provide an efficient and accessible guidance method that carefully introduces the tutored system to new users.

The potential benefits and problems are mostly related to how users will perceive a tutor and learn to use an application with it. Accessibility is also an important benefit, since a tutor embedded into an application is always available when the system is used, and every potential user can benefit of it. Demonstrating a tutor that is successful in its task can prove the viability of the concept of software tutoring in speech-based interfaces. The measure of success for tutoring is a complex concept. We can measure learning results by asking users to perform tasks with a tutored application. We can measure user attitudes towards the tutor and the tutored application to gain subjective views of the tutor and its effects. It is also possible to observe the interaction that users have with a tutored application and report such figures as the number of errors and the speed of the interaction. For the measure to be meaningful, we need a point of comparison. A good point of comparison is a conventional guidance method, such as a web manual, that has been successfully used with the application. If a tutor can match the currently used guidance method, and it has some benefits, such as better accessibility, the tutoring concept can be very interesting guidance method.

In this paper we describe an experiment where a software tutor was compared with a web manual. The paper continues with a description of the experiment used to compare a software tutor integrated into a speech application to a web manual. First the method is described, followed by the results. The paper continues with discussion about the results and their relevance and applicability. Finally, conclusions are drawn and some possible future directions presented.

Method

To study how software tutoring works in a speech only environment, a two-condition between-participants experiment was conducted. An existing telephone operated speech-based e-mail reading system called Mailman (Turunen and Hakulinen, 2000) was used in the experiment. Participants were given limited time to learn to use the Mailman system. In the first condition (tutor condition) a software tutor was used as training material. In the second condition (web condition) a web manual was used. In both conditions the participants had 20 min to learn to use Mailman with the help of the guidance material. Next, the guidance material was removed and the participants spent another 20 min trying to accomplish a set of given tasks with Mailman. All telephone calls made to the Mailman system were recorded for later analysis. In the last part of the experiment, participants' attitudes towards Mailman and the guidance they received were measured with two questionnaires.

Participants

Eighteen people (10 females, 8 males) participated in the tests. All participants were students, mostly undergraduates and some graduates in Finnish universities. People were asked to participate by sending e-mail to the participants on a course and giving out information in class. Some flyers were also left in the University for students to find. Members of the research group also personally informed some participants about the option to participate. In all cases it was announced that the participants would receive a movie ticket. The ticket was handed to each participant when the test was over.

The age of the participants varied between 20 and 48 years, average age being 27 years. All participants had used computers actively for several years, ranging from 4 to 16 years and 9 years in average. Participants' personal ratings of their computer skills ranged from novice to expert. None had extensive experience with speech-based systems, but some had worked a little with speech synthesis and some had used voice commands in mobile phones.

The participants were students of computer sciences, journalism and mass communication, translation studies, social work, history and archaeology. All participants were native Finnish speakers with no reported hearing problems.

Participants were randomly assigned to the conditions. There were no statistically significant differences in the age, sex, computer skills or experience with speech systems of the participants in the two conditions. All participants agreed that all the telephone calls made during the experiment could be recorded for further analysis.

Procedure

The tests were conducted in an office environment using a normal fixed line telephone and a computer to view web pages. Participants did the experiment one at a time. Each test consisted of an introduction, a 20-min learning period, a 20-min period with tasks and questionnaire filling in the end. During the experiment, an experimenter was present all the time.

In the introduction, the structure of the experiment was described to each participant. Next, a piece of paper with a short description of Mailman and the learning period was given to the participant. The information stated that Mailman is a telephone based e-mail reading application with speech input and support for telephone key input. A telephone number and necessary user codes were also included. The difference in the instructions between the two conditions was that in the tutor condition the piece of paper stated that Mailman includes the necessary help functionality and in the web condition it mentioned the Mailman web pages. The experimenter also stated all the information verbally and opened the web pages for viewing on computer screen in the web condition. In the tutor condition the computer was turned off. The participants were also informed that telephone calls would be recorded. Consents for the recordings were obtained verbally from the participants. The participants were allowed to make notes during the experiment.

Each participant had 20 min to learn to use Mailman with either integrated tutoring or a web manual. The participant was told about the time limit and was allowed to freely choose how to use the time. The experimenter stayed in the back of the room and intervened only if the participant had a serious problem and asked for help. When the 20 min was over, the experimenter asked the participant to finish his/her session with Mailman.

Next the training material was removed and the participant was told that he/she had 20 min to work on seven tasks. The tasks required searching for information from a mailbox using the Mailman system. A piece of paper with the tasks was given to the participant and the experimenter asked him/her to write down the answers that he/she could find for the task. The tasks could be performed in any order and the answers to all tasks were found from a single mailbox. The participants could, for example, skip tasks and return to them later if they wanted to. When the 20 min was over, the experimenter asked the participant to stop working with the tasks.

In the end the participants filled in two questionnaires where they answered questions about Mailman and the training material. After this the experimenter explained the nature of the experiment and there was an opportunity for the participant to provide verbal feedback.

System

The application that the participants learned to use was Mailman, a telephone-based e-mail reading application. Speech input and telephone keys can both be used to control Mailman. Mailman uses small-vocabulary (28 words) speech recognition and it does not support barge-in. However, telephone keys can be used to interrupt Mailman's utterances. When a user calls Mailman, he/she first identifies herself by entering a user code and a pass code with telephone keys. Mailman retrieves copies of the user's new e-mail messages or old messages, if there are no new messages in the user's mailbox. The mailbox in a server is left untouched. If there are more than six messages, which was always the case during the experiment, Mailman automatically groups the messages to make it easier to browse them. After the login phase the dialogue style is completely user initiated.

System functionality includes listings of message groups and messages within groups. A user can also select a group or a message by referencing them by numbers or words next and previous. Furthermore, one can request more detailed information on a particular message or read the message. When messages are being read, Mailman interprets some elements, like web addresses, and converts them into a more readable form before sending them to speech synthesis. The language used in the e-mail messages is automatically identified and a matching speech synthesiser is used to read each paragraph. Mailman uses two speech synthesisers for output, one for Finnish and another for English. The version of Mailman used in the experiment was a Finnish language version with support for English language e-mail content. A user can navigate inside mail messages using telephone keys. Other functionality includes general and context sensitive help and login and logout. There were no message composing, organizing or deletion functions in the version used in the experiment.

Two static mailboxes were used in the experiment. During the learning period a demonstration mailbox with twelve messages and bilingual content was used. For the tasks, a mailbox, which matched the tasks that the participants worked on, was used. This mailbox had eight messages and content only in Finnish.

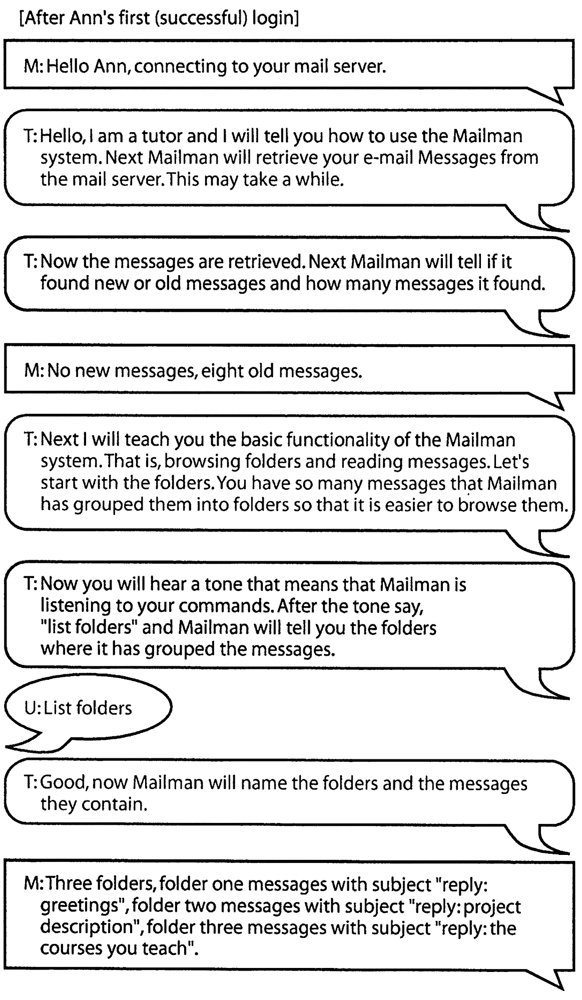
Manipulation

The difference between the conditions was the guidance used. A tutor module embedded in the Mailman system was used in the first condition. A web manual was provided to participants as guidance for the second condition. The guidance was completely in Finnish in both cases.

The tutor that served as training material in the first condition is, from the user point of view, a separate dialogue partner embedded in the Mailman system. The tutor utterances are recorded human (male) speech, while Mailman uses a speech synthesis (with male voice). Users cannot talk to the tutor, with the exception of the repeat command. However, the tutor monitors what the user is doing with the system and reacts to that. The tutor is also able to control Mailman; most importantly, it can discard user inputs so that Mailman will not process them.

The tutor contains about twenty different tutoring messages. It has two distinct parts. In the beginning of the first call, a user is required to follow instructions given by the tutor until the basic Mailman functionality has been covered. This part includes an introductory message presenting the tutor and informing the user about mail retrieval. The concept of automatically generated folders is introduced to the user and the tutor asks the user to list folders. If the user does not succeed in listing folders when asked to do so, the tutor informs the user about the problem and asks him/her to try again until the user successfully gives the requested command. The tutor then gives positive feedback to the user and lets Mailman carry out the corresponding action. This part of the interaction is depicted in Fig. 1. The monitoring of user actions is accomplished simply by comparing speech recognition results to the input that the tutor requested the user to give. Since the speech recognition grammars are small, an exact match can be required. A similar technique is used when the user is requested to select a folder, select a message and read it. When Mailman starts reading the message, the user is informed on how to browse inside the message. When the message has been read (or the user has cancelled the reading) the tutor gives a summary stating that this was the basic functionality and now the user can freely use the system. The tutor also says that it will give the user more information when appropriate.

The latter part of the tutoring consists of these additional tutoring messages. The messages teach the user the remaining available speech commands and the most important key commands. The tutor also informs user on how Mailman reads e-mail messages, e.g., how different elements like web addresses are read. Tutor also advises the user how to access the system's own help functionality. From this functionality the user can learn the rest of the key commands and obtain some general information. However, all system functionality is covered in the tutor messages. The tutoring messages in the latter part are presented in appropriate context, i.e., after specific user action and possibly after some specific other tutoring messages.


[After Ann's first (successful) login]

M: Hello Ann, connecting to your mail server.

T: Hello, I am a tutor and I will tell you how to use the Mailman system. Next Mailman will retrieve your e-mail Messages from the mail server. This may take a while.

T: Now the messages are retrieved. Next Mailman will tell if it found new or old messages and how many messages it found.

M: No new messages, eight old messages.

T: Next I will teach you the basic functionality of the Mailman system. That is, browsing folders and reading messages. Let's start with the folders. You have so many messages that Mailman has grouped them into folders so that it is easier to browse them.

T: Now you will hear a tone that means that Mailman is listening to your commands. After the tone say, "list folders" and Mailman will tell you the folders where it has grouped the messages.

U: List folders

T: Good, now Mailman will name the folders and the messages they contain.

M: Three folders, folder one messages with subject "reply: greetings", folder two messages with subject "reply: project description", folder three messages with subject "reply: the courses you teach".

Figure 1. Tutor guiding a new user and explicitely requesting the user to give a specific input.

To cover all the tutor content, a user must use the system for about 20 min. The first part, consisting of the basic functionality, takes about 5 min. The tutoring has been iteratively developed and user tested in three phases (Hakulinen et al., 2004).

As the training material for the second condition, a manual consisting of a set of web pages was used. This manual is the standard Mailman manual available to all users of Mailman. The manual pages are part of the Mailman home pages. The manual includes one page of general introduction, a page that lists and describes speech commands and another page for telephone key

(DTMF) commands. The introductory page includes notions about speech recognition and on how to speak to the system.

Compared to the tutor, the web pages include a more detailed discussion on message groupings, discuss different kinds of telephones and acoustic environments, explain briefly about the limitations of speech recognition and explain the functionality of each command in greater detail.

The web pages have been iteratively developed in three phases of evaluation with different user groups (Salonen and Helin, 2004) and have been in use for several years.

Measures

All telephone calls made to Mailman during the experiment were recorded. The experimenter transcribed these audio files afterwards. In the transcriptions all interaction that the participants had with Mailman were written down.

Speech inputs were grouped into successful and failed. The failures were further grouped into out-of-vocabulary inputs, recognition errors, barge-in errors and speaking-too-quietly errors.

Recognition errors are situations where the user used a correct command but the speech recogniser either rejected it or gave an incorrect recognition result. Out-of-vocabulary inputs are inputs given at the correct times but the given input is not included in the system vocabulary or grammar. Speaking at the wrong time is a situation where the user spoke to the system when the system was not listening. The system did not include barge-in and input could be given only at specific times identified to the user with a tone played by the system. Speaking too quietly means that the user spoke at a correct time but the voice activity detection component of the system could not detect that the user was speaking, thus the input was discarded and the system kept on listening.

Key inputs were annotated and grouped into successful key inputs, to interruptions where the user interrupted a Mailman utterance intentionally and to erroneous key inputs. A key input was considered erroneous if the user entered a key command but did not receive a response. These cases are sometimes difficult to detect conclusively, but an input was considered to be erroneous if it was followed by a long delay by both the system and the user. Cases where a user pressed multiple keys in an attempt to enter a command requiring a key sequence, but produced a command that is activated by a single key press were considered errors. All types of key related problems were caused mostly by the fact that users did not realise that the keys were used both to give inputs to the system and to interrupt the system outputs.

All the interactions and different types of errors were recorded separately for the training period and the tasks for each participant.

There were seven tasks that the participants worked on. The tasks were information retrieval from a mailbox and each task was designed to be more challenging than the previous one. The tasks were to (1) find out the number of messages in the inbox, (2) find out the number of different message senders, (3) to find out a time and place for a meeting, (4) to find out a telephone number for a specific person, (5) to write down a web address sent by a specific person, (6) to find out what administrative issue a specific person has mailed about and (7) to find out who is going on a business trip. In practice, for example the second task required a user to listen to the message group listing. The messages in the mailbox used in the tasks were grouped per sender but the last group contained messages from various senders. The user then had to select the last folder and listen to the message listing to find out the total number of senders.

Sheets of paper with the participants' answers were collected and analysed. Each answer was given a score between 0 and 1. A score of 0 meant that there was no answer or that an answer was completely wrong. A correct answer received a score of 1 and scores between 0 and 1 were given in cases where an answer, for example a telephone number, had some mistakes in it but was correct to some extent.

In the end of the experiment participants filled in two questionnaires, one concerning the Mailman system and the other on the guidance they received (the tutor or the web manual, depending on the condition). The questionnaires about Mailman and about guidance were very similar. Both included open questions asking about good and bad features of the guidance and of Mailman. In addition, there were 22 questions with five-point scales taken from Hassenzahl et al. (2000). These questions measured what Hassenzahl et al. call the "hedonic" and "ergonomic" quality. Additionally, the Mailman questionnaire included scales about efficiency, usefulness and adequacy of features of the system. The guidance questionnaire included scales asking whether enough information was provided, if guidance was coherent and if the length of the guidance was appropriate.

Results

Number of Errors and Used Commands

The telephone calls were analyzed to find out the errors the users had with the system and what commands they were using to control the system.

During the Tasks. There was a significant difference between the conditions on how often speech commands were used compared to key commands. During the tasks the tutor condition participants used significantly more ($t(16) = 2.24$, $p < 0.05$) speech commands. The key/speech command ratios remained almost the same between the training period and the tasks in the web condition while the participants in the tutor condition used on average more key commands during the tasks.

In both conditions there were participants who did not use key commands at all. In the tutor condition there were four such participants and in the web condition there were two. In the web condition there were two users who did not use speech input at all. During the tasks on average 8% of successful commands were given with keys in the tutor condition while 49% of commands were keyed in the web condition.

A special type of key command not included in the preceding comparison is navigation commands available when reading an e-mail message. This functionality is not available with speech, so key commands are the only option. Navigation inside e-mail messages was equally common between the two groups.

One more type of key usage is interruptions, i.e. users can interrupt the system prompts by pressing telephone keys. These presses are not considered commands and are not included in the numbers above.

Average numbers of inputs and errors are summarised in Table 1. There were no significant differences on the number of errors during the tasks. Figure 2 shows the average number of successful and erroneous inputs during tasks in the two conditions.

There were no significant differences between the recognition rates or out-of-vocabulary input frequencies between the conditions. The average recognition rate (excluding out-of-vocabulary entries and inputs that did not reach the speech recognition component) for the entire experiment was 94%. The rate varied between 70% and 100% between the participants. Mostly the recognition errors were rejections and caused an error-handling message from Mailman or a message from the tutor when it was active. There were also some individual false recognitions causing incorrect system response.

Table 1. Average number of commands given per user. Total number and erronous inputs reported.

	Tutor condition			Web condition		
	Training	Tasks	Total	Training	Tasks	Total
Speech commands	19.8	31.0	50.8	10.9	21.7	32.6
Speech problems	6.2	4.9	11.1	5.9	6.2	12.1
Key command	4.4	12.4	16.9	15.3	26.6	41.9
Key problems	0.7	3.2	3.9	6.6	5.7	12.2
Error (%)	28	19	22	47	25	33

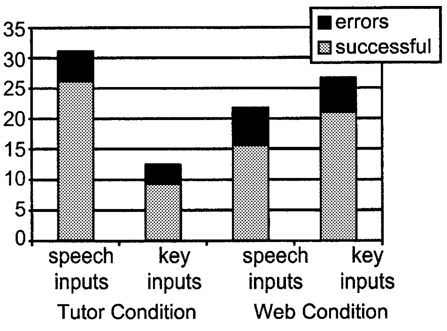


Figure 2. Average number of inputs during the tasks.

There were in total 21 out-of-vocabulary errors in the test. Mostly these were synonyms of the valid commands but not included in the recognition dictionary of the Mailman system. Two users used a phrase from the web manual that was not a command but a heading in the command list. There was no statistically significant difference in the number of out-of-vocabulary errors between the conditions.

During the Training Period. There were significant differences in how many errors occurred in the usage between the different conditions. The differences, with the exception of one case, were in favour of the tutor condition, i.e. participants in the tutor condition had fewer problems. Inputs and errors during training period are shown in Fig. 3.

During the training period more speech was used in the tutor condition than in the web condition (student's t-test $t(16) = 4.56$, $p < 0.005$).

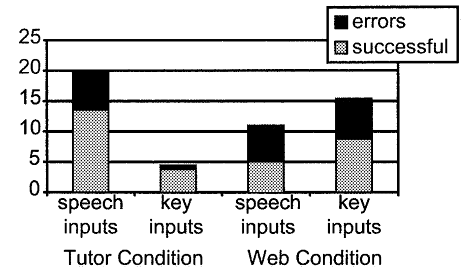


Figure 3. Average number of inputs per user during the training period.

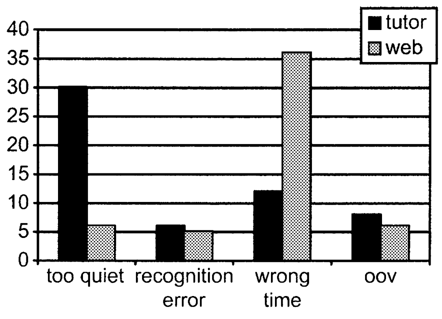


Figure 4. Total number of different types of speech input errors in the training period.

The average error percentages calculated from erroneous inputs and all inputs differed significantly during the training period. The tutor condition had fewer errors ($t(16) = -2.68$, $p < 0.05$).

Problems related to usage of keys occurred more often in the web condition than in the tutor condition during the training period ($t(16) = -5.38$, $p < 0.001$). When percentages of erroneous key inputs out of all key inputs are compared the difference is also significant ($t(16) = -7.49$, $p < 0.005$).

Problems with speech related issues are apparent if we remove the two participants from the web condition who did not even try to use speech commands. Without this data, the web condition participants had significantly higher problem percentages of speech commands during the training period ($t(14) = -3.00$, $p < 0.01$).

We have grouped the speech related problems into four types; recognition errors, out-of-vocabulary errors, talking to the system when it is not listening, and speaking so softly that no speech is detected. There were significant differences during the training period in speaking at the wrong time. Those web condition participants who used speech commands spoke more often at the wrong time ($t(14) = -2.69$, $p < 0.05$) and gave a greater percentage of their speech inputs at the wrong time ($t(14) = -5.43$, $p < 0.0001$) than tutor condition participants. The different types of errors during the training period are presented in Fig. 4.

There is a significant difference in speaking-too-softly type of errors. When the numbers of too soft speech inputs given during the training period are compared, the tutor group gave more erroneous inputs ($t(16) = 2.74$, $p < 0.05$). The differences, however, are no longer statistically significant if we look at the percentages of too quiet speaking out of all speech inputs or remove the two participants who did not try to use speech at all.

General Observations. While the participants in the tutor group usually made one 20-min long call to the tutored Mailman system during the training period, the web group had a free choice on when to call the system. On average the participants in the web condition spent 10 min reading the web pages before picking up the phone. There was a lot of variation so that the time spent reading the web pages ranged from 5 to 15 min as some read all the web pages carefully before calling the system and others studied the pages during or between phone calls.

In tutor condition, when the tutor was giving explicit instructions to users and monitoring them, 4 out of the 9 users had problems with Mailman which resulted in a corrective action by the tutor. In three cases users were speaking too softly and in one case there was a speech recognition error. The tutor was able to provide help in to all these users.

Half of the users used the opportunity to make notes during the training period. They wrote down some of the key commands and sometimes some of the speech commands. Four of these were web condition participants and six tutor condition participants. It was not observed that the users would have read their notes during the tasks.

There was one statistically significant difference between the conditions regarding what commands were used. In the web condition the "next message" command was used more often than in the tutor condition ($t(16) = -2.13$, $p < 0.05$). On average in the web condition the "next folder", "next message" and "list messages" commands were used more often. In the tutor condition commands of the type "read the first message", "read the second" were used more often on average. However, these differences were not statistically significant. The biases, if any, may be directly connected to the fact that in the web condition more key commands were used and in the tutor condition speech commands were more popular. Commands like "read the third message" are not directly available with key commands but required a combination of separate select and read commands. Furthermore, commands of the type "select second message" and "select third folder" require several key presses while the "next message" and "next folder" commands can be accessed with just one key press.

Some participants in the web condition, who had several speaking at a wrong time type of errors during the training, kept speaking at the wrong time even during the tasks. There are no statistical differences between the conditions as a whole, but two participants in the web condition had more than ten inputs spoken at the wrong time during the tasks, i.e. they did not learn the interaction style of Mailman during the training period.

Task Completion as a Measurement of Learning Results

All participants were able to accomplish the majority of the tasks. Everybody completed the first three tasks except one participant who did not answer the very first question. The fourth task, picking up a telephone number was the first task where some participants had problems. However, only two participants did not provide correct answers. The fifth task, writing down a web address, proved far more difficult, only three participants got the address exactly correct, while most wrote down the address only partially or had some mistakes in it. All but one of the participants answered the sixth question on important administrative issues. Three participants gave incomplete answers. The last task was the hardest. One participant gave correct answers and one a partially correct answer. The others either did not answer at all or wrote down incorrect answers. When answers were scored and total scores calculated for each participant. The average score for the tutor condition was 5.14 and for the web manual condition 5.38 (the maximum being 7). There were no significant differences between the scores of the two conditions.

User Attitudes

There were no statistical differences between the conditions in the participants' answers to the questionnaire items with scales. Mailman was evaluated similarly in both conditions and the evaluations of the different guidance materials were also so similar that no significant differences were found in any of the scales.

The questionnaire forms included open questions where participants could comment on good and bad features of the Mailman system and the guidance they received. Several possible improvements for the tutor were suggested. In general, participants wanted to have more control over the tutoring, for example one user requested a "more info about this" functionality, and another commented that the tutor should tell about the "what next" and/or "help" commands right in the beginning. Yet another comment was that there should be information right at the beginning on what to do in problematic situations, for example, if for some reason a user misses a tutoring utterance. Some users also requested an option to actively ask about different subjects. Some negative comments were also given because of the delays with the system and tutoring.

On the positive side, many participants appreciated the fact that the tutor gave them the opportunity to try out the commands right after they had been tutored. There were some complaints about tutoring content being unorganised and occurring at surprising points. However, positive comments were given about the consistent partitioning of the tutoring content into tutoring messages. In general, tutoring content was reported to be very easy to comprehend.

Comments about the web pages included criticism that the actual command list for the spoken commands was not easily available and that it was not clearly enough stated when a user should talk to the system.

Discussion

As the results show, the tutor was an effective guidance method. It performed as well as a carefully written web manual in most comparisons. There were no significant differences between the conditions as to how well the participants could accomplish the tasks given to them. All participants eventually learned to use the system well enough. The limited set of tasks resulted in very similar results for most participants. With a larger set of slightly easier tasks we could possibly have measured the efficiency of system usage. The set of tasks used only proved that with both guidance methods it is possible for a new user to learn to use Mailman efficiently within less than 40 min of hands-on experience.

The differences were found in the interaction during the training period. They were in the favour of the tutor and were found in the number of errors the participants made. Use of the tutor resulted in significantly fewer errors in the interaction during the first 20 min of guided usage than the use of the web manual. The tutor ensured consistent learning within the condition while in the web condition it took some participants more than half an hour to learn to use Mailman efficiently, while others learned reasonably fast. The problems that the web condition participants had were mostly speaking at the wrong time and key related problems. Both of these are related to the style and pace of interaction; the interface did not have suitable tempo (Weinschenk and Barker, 2000) for some users and this caused many of the problems. The interactive, speech-based tutor seems to be able to teach the actual interaction style to the participants better, and much faster than the conventional manuals.

Looking at the interaction that participants had with Mailman, the most important strength of the tutor seems to be that it takes care of the users. In the web condition some participants used Mailman very inefficiently for long periods, without even realising that they were working in a way that was incompatible with the Mailman interface. In the tutor condition all participants learned the correct way of interacting with Mailman in the very first minutes. The tutor asked users to give specific inputs to the system and checked that they were given successfully. This way it was possible to spot the problems that the users were having. If the tutor, or a system, was only to monitor user actions without knowing the user's intentions, problems could not be so easily spotted. Participants also commented positively on this type of interaction. They liked the fact that when something was tutored, they could try it out themselves right away. There were no comments, positive or negative, on the feedback that the tutor gave to the participants. However, during the interaction the feedback did indeed seem to work well and confirm to the participants that they were doing fine.

The tutor design seemed to work very well in the experiment. The participants had no problems in having a third participant, the tutor, participate in the dialogue they were having with the Mailman system. They could easily distinguish the tutor and the system. Somewhat surprisingly, the added length of interaction from the tutor outputs did not cause the participants any confusion or frustration. The Mailman system was somewhat slow already without the tutor, but this seemed to be a bigger problem for the web condition participants than for the tutor participants. The tutor managed to communicate the pace of interaction to the participants very well.

The only problems with the large amount of speech output the users received were when the tutor listed several commands in one tutoring utterance. This happened in one or two poorly designed utterances. These outputs conflicted with the widely accepted speech interface design rules of keeping outputs short and avoiding long lists (Suhm, 2003). In general, the tutor presented information to users in small enough chunks. Many participants gave positive comments on this. Usually one feature or command was taught to a user at a time and the user was given an opportunity to experiment with it immediately.

The design of the tutor voice worked reasonably well in the experiment. There are no easy measures to validate the design of the voice, but users commented that the voice was easy to understand and nobody complained about the voice being annoying or frustrating. The large number of users who spoke too softly to the system in the tutor condition may indicate that the tutor voice had a too soft speaking style when it gave examples. This, however, is more a problem of the voice activity detector than of the tutor design.

There are some issues to be considered when the results of this experiment are applied in a wider context. The type of application that we chose to be tutored was very amenable to the tutoring. Mailman is a user initiative system so it requires some type of guidance material for new users. Mailman has a reasonable number of features to be tutored and it also includes an individual user model for each user, so tutoring can be automatically adapted to each user. The nature of the experiment may also have removed some relevant effects. When participating in an experiment, for example the "paradox of active user" may disappear to some extent. In real life users may be less willing to listen to and especially read long instructions.

Conclusions and Future Work

The experiment has shown that tutoring does indeed work; it resulted in similar learning results as the currently used web page guidance. It is possible to teach usage of a speech-based system with an embedded software tutor. Presenting the tutor as a separate dialogue partner was a successful design decision and the participants received the mixture of guidance and actual use positively. When tutoring is applied in different types of applications new design challenges may emerge. Different types of tutoring messages and monitoring of user actions may need to be applied when the style of interaction changes.

All in all, tutoring works particularly well when new users are introduced to speech-based systems. The tutor can take care of most if not all users. In very simple applications, which are used in most cases only once by each user, tutoring is not applicable. Furthermore, a user model is something that significantly eases the design of tutoring. If an application is suitable for tutoring we can recommend that tutoring be considered as one way of guiding new users. Accessibility of embedded tutoring is a considerable advantage. While, e.g., a web manual may not be accessible when it is needed, or may be hard to use for visually impaired users, an embedded tutor is always available when the system is used.

In the future it would be interesting to experiment with tutoring in applications with more open dialogue style and domain structure. It is also an interesting idea that users could have a dialogue with a tutor. The tutor used in this study only monitored the user's interaction with the system and provided guidance. Giving users a change to talk to a tutor would raise questions on dialogue structure and bring tutoring closer to the field of dialogue based intelligent tutoring systems.

Another direction is building a multimodal tutor. The tutoring component can be placed, for example, in a web page and it could be synchronised to the application. Usage of interactive graphics and text gives the tutor wide ranging options. Finally, tutoring has theoretical advantages and each of these could be studied in more controlled experiments. Learning theories could be included in such studies and it would be interesting to see how people with different learning styles would work with tutor.

References

Ames, A.L. (2001). Just what they need, just when they need it: an introduction to embedded assistance. In *Proceedings of The 19th Annual International Conference on Computer Documentation*, New York, NY: ACM Press, pp. 111–115.

Carroll, J.M. and Rosson, M.B. (1987). Paradox of the Active User. In J.M. Carroll (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*. Cambridge, MA: MIT Press.

Esteban, J. (1996). AppleGuide. *Interactions*, *3*(3):36–37.

Fischer, G., Lemke, A.C. and Mastaglio, T. (1990). Using critics to empower users. *The SIGCHI Conference on Human Factors in Computing Systems*, *CHI90 Proceedings*. Seattle, WA: ACM Press, pp. 337–347.

Fry, J., Ginzton, M., Peters, S., Clark, B. and Pon-Barry, H. (2001). Automated tutoring dialogues for training in shipboard damage control. In *Proceedings of 2nd SigDial Workshop on Discourse and Dialogue*. Aalborg, Denmark: ACL.

García, F. (2000). CACTUS: Automated tutorial course generation for software applications. *Intelligent User Interfaces 2000. IUI' 2000 Proceedings*. New York: ACM Press, pp. 113–120.

Hakulinen, J.S., Turunen, M., Salonen, E.-P. and Räihä, K.-J. (2004). Tutor design for speech-based interfaces. *Designing Interactive Systems*. In *DIS2004 Proceedings*. Cambridge, MA: ACM Press, To Appear.

Hassenzahl, M., Platz, A., Burmester, M. and Lehner, K. (2000). Hedonic and ergonomic quality aspects determine a software's appeal. *The SIGCHI Conference on Human Factors in Computing Systems, CHI2000 Proceedings*. Hague, The Netherlands: ACM Press, pp. 201–208.

Heisterkamp, P. (2003). Do not attempt to light with match!: Some thoughts on progress and research goals in spoken dialog systems. In *Eurospeech 2003 Proceedings*. Geneva, Switzerland: ISCA, pp. 2897–2900.

Kamm, C., Litman, D. and Walker, M.A. (1998). From novice to expert: The effect of tutorials on user expertise with spoken dialogue systems. *International Conference on Spoken Language Processing, ICSLP98 Proceedings*. Sydney, Australia: ASSTA, pp. 1211–1214.

Nakatami, L.H., Egan, D.E., Ruedisueli, L.W., Hawley, P.M. and Lewart, D.K. (1986). TNT: A talking tutor '*N*' trainer for teaching the use of interactive computer systems. *SIGCHI Conference on Human Factors in Computing Systems. CHI'86 Proceedings*. Boston, MA: ACM Press, pp. 29–34.

Rosé, C.P., Litman. D., Bhembe, D., Forbes, K., Silliman, S. Srivastava, R. and VanLehn, K. (2003). A comparison of tutor and student behavior in speech versus text based tutoring. *The HLT-NAACL Workshop on Building Educational Applications Using Natural Language Processing Proceedings*. Edmonton, Canada.

Salonen and Helin (2004). Personal communication.

Suhm, B. (2003). *Towards Best Practices for Speech User Interface Design, Eurospeech 2003 Proceedings*. Geneva, Switzerland: ISCA, pp. 2217–2220.

Turunen, M. and Hakulinen, J. (2000). Mailman—a Multilingual Speech-only E-mail Client based on an Adaptive Speech Application Framework. *Workshop on Multi-Lingual Speech Communication, MSC 2000 Proceedings*. Kyoto, Japan, pp. 7–12.

Weinschenk, S. and Barker, D.T. (2000). *Designing Effective Speech Interfaces*. John Wiley and Sons, inc.

Yankelovich, N. (1996). How do users know what to say? *Interactions, 3*(6):32–43.

Paper V

Jaakko Hakulinen, Markku Turunen, & Esa-Pekka Salonen. Visualization of Spoken Dialogue Systems for Demonstration, Debugging and Tutoring. In *Proceedings of Interspeech 2005*, ISCA, pages 853-856.

Visualization of Spoken Dialogue Systems for Demonstration, Debugging and Tutoring

Jaakko Hakulinen, Markku Turunen and Esa-Pekka Salonen

Speech-based and Pervasive Interaction Group, TAUCHI, Department of Computer Sciences
University Tampere, Tampere, Finland
{Jaakko.Hakulinen, Markku.Turunen, Esa-Pekka.Salonen}@cs.uta.fi

Abstract

Graphical elements have been found very useful when spoken dialogue systems are developed and demonstrated. However, most of the spoken dialogue systems are designed for speech-only interaction and are very hard to extent to contain graphical elements. We introduce a general model to visualize speech interfaces. Based on the model we present an implemented visualization framework, and several example visualizations for demonstrations, debugging, and interactive tutoring of speech applications.

1. Introduction

Speech interfaces are efficient in many situations. They are available via telephone when other interfaces are not possible and suit well in situations where users' hands or eyes are busy. Furthermore, speech can be very efficient in certain tasks like selecting objects by multiple attributes. When graphics are available, it is best to combine benefits of speech and graphics in a multimodal interface.

There are some situations where traditional spoken dialogue system and graphics can work efficiently together. We present cases where visualizations are connected to telephone based applications. Graphics are not normally available when these applications are used. However, there are certain situations where graphics can be very useful. When users learn to use new systems, they are often presented with manuals that are graphical in nature, e.g., web pages. Visualizations can also be used when systems are demonstrated, for example, in educational purposes. Finally, developers can work more efficiently if they are provided with sufficient tools to visualize the internals of the system they are developing.

In this paper we present a general model to visualize existing speech-based applications. Based on the model, we have implemented a concrete visualization framework. Using the framework we have visualized several existing applications without any modification on their program code. The requirements for the visualization are discussed and our implementation is analyzed respectively.

We also present examples of the visualizations including a speech balloon visualization of spoken interaction for demonstration purposes and interactive software tutors that carefully guide new users when they learn to use a spoken dialogue system.

Next we study briefly the background and context of this work and consider the idea of visualizing spoken interaction. Technical details and experiences with the various uses of the visualizations follow. The paper ends with conclusions.

2. Visualization of spoken human-computer interaction

Various visualizations of speech and spoken communication have been developed. Speech signals, as well as entire dialogues [1, 2], have been visualized for research development and educational purposes. Many annotation tools also have visualizations to help analysis of dialogue annotations [3]. Finally, dialogue models have visual representations in graphical tools that are used to design and implement spoken dialogue systems, such as CSLU Toolkit [4].

The available visualizations are mostly aimed for developers and researcher. On the other hand, graphics are used in multimodal systems in many ways. The visual component can help users to better follow the interaction [5] and a mental model that the visual component can provide can transfer to speech only interaction [6].

Real time visualization of spoken dialogue systems aimed for end users have not been widely used so far. However, experiences from multimodal systems and static tutorials [7] suggest that they can be very effective.

Spoken interaction between a dialogue system and a user can be visualized in various ways using multiple dimensions. For example, the visualization can be static, dynamic or animated. Focus of visualization can be on the conveyed information, systems internal representation, turn taking, prosody etc. In this work we concentrate on dynamic visualization and discuss visualization of spoken utterances and interaction style. Speech is essentially temporal and serial in its nature and to convey the flow of the interaction to the users, the visualization must be real time. The nature of interaction easily changes if the interaction slows down significantly because of the visualization.

Target groups for visualizations include both developers and users. Developers are familiar with the nature of the interaction, and are interested in what the systems internal status is. They often need detailed technical visualizations about results of speech recognition and natural language understanding. Users, on the other hand, can benefit from the visualization when they familiarize themselves with a new system. They need to learn the interaction style, e.g., how turn-taking occurs in the dialogue. They also need to learn the system functionality and what kind of inputs the system understands. The dialogue structure can also be made more explicit when users can, e.g., see dialogue history.

As always with speech interfaces, error management is important. Users must learn to identify situations caused by speech recognition errors. When users see the results of speech recognition, they can notice the recognition errors immediately.

3. General visualization framework

A visualization framework for speech-based applications must fill certain requirements. Preferably it should be general so that there is no need to independently integrate the visualization to each new application. At the same time, its architecture should make it possible to add application specific visualizations as necessary and adapt the visualization to different needs.

The visualization and especially the connection between visualization and an application must be efficient. The visualization might become misleading, or even useless, if it slows down the interaction. We can avoid some of the efficiency bottlenecks by using an architecture that enables distribution on different computers. Distribution makes it also possible to visualize applications from different physical locations. For example, the interaction with a telephone based application could be visualized outside the premises where the application server is located. Preferably the architecture should also be robust in error situations. In particular, problems in the visualization should not have an effect on the application.

3.1. Visualization framework architecture

Our generic model for visualization framework consists of three components as seen in Figure 1. The visualized application has one or more connection components, which provide access to the internals of the system. The visualization itself is running as a standalone component. These two are connected via a "middleman" component, which is a server for both the visualization and the application acting like a proxy between them. The reason for a separate component in the middle is that this way neither of the other components needs to be a server. E.g. web applets, which are potential visualization containers, cannot be servers. The separate communication component makes it also easier to configure firewalls. Furthermore, the middleman component can increase reliability; problems with the visualization do not break down the application as middleman can tell the application to keep on running. There are also some efficiency gains, especially when the application uses a turn based interaction model. The middleman can also simplify the development of the two other components as it can take care of much of the complexity related to timing the interaction.

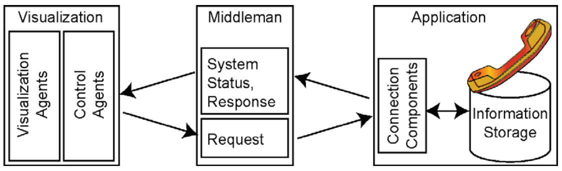


Figure 1: General model for the visualization framework.

The visualized application needs connection components that can provide information from the application. In order to control the systems behavior the connection components need also means to modify internal state of the system. This is important in debugging and some tutoring situations. For example, when tutoring new users removing incorrect inputs is an option. In general, the ability to modify the system state offers numerous possibilities to make truly interactive visualizations and add-ons to existing applications.

3.1.1. *Underlying application architecture*

The general model presented can be implemented to many systems. For example, in Galaxy II [8] the connection components could be connected to the hub just like any other component. Our applications are implemented on top of Jaspis architecture [9]. One of the core features of Jaspis is centralized Information Storage. All components have access to the storage and store all information there. This makes it possible to visualize any information wanted without modifying the application. In addition, shared information makes it possible to modify the system state, and in this way control the system behavior.

Jaspis-based systems contain managers and agents, among other components. Managers in Jaspis represent high level task distribution of spoken dialogue system. Each manager has a set of agents that are compact software components to carry out actual tasks, such as dialogue decisions.

3.1.2. *Connection components*

We have implemented generic connection components that can be included into any Jaspis-based system. These components have access to the Information Storage and all information inside the application is available for the visualization. Connection components can also modify the Information Storage and thus control the system.

The generic connection components are built as managers and they connect to the middleman component when they receive a turn. They can halt the system by not giving the turn away in timely manner. Because of this the connection to the visualization components must be fast, in particular when real time visualization is required. To optimize the speed of the connection, the connection components send the most important information to the middleman by default, such as pending outputs, input results and a short description of application status. This removes the need for separate requests for frequently needed information.

We have also a different version of the connection component that is a standard Jaspis input/output (I/O) component. This component is run concurrently with other I/O components. It also produces I/O results that are processed by Jaspis input processing components. While the component enables more concurrency it requires more complicated configuration. Both types of connection components can be used in parallel to achieve maximum control and speed.

Adding the connection components to existing applications is straightforward. The components are added to the system configuration. The default components work with all Jaspis based dialogue applications. Because the components provide access to the information storage via a standard interface, components in visualization side can access application specific information. To further optimize speed and modify the conveyed data, small application specific connection components can be implemented.

3.1.3. *Visualization components*

The visualization components we have implemented are Jaspis applications. They have two managers; one runs the actual visualization agents and the other includes control agents that decide when and what to visualize. Visualizations agents run concurrently with the rest of the system. This enables, for example, animations while the application is processing. A callback routine provides feedback from the visualizations agents to the control agents.

The control agents control the visualization agents and use the middleman component to communicate with the visualized application. In simple visualizations one control agent requests the visualized application to process normally and make the visualization agents display the information about the status of the application. In tutoring systems, control agents decide what kind of tutoring to provide to the user and control when the application is allowed to proceed. These decisions are based on the users' interaction with the tutoring and information found in the applications Information Storage.

3.1.4. *Middleman*

The middleman component is a server that serves the two other components. It stores internally the information that the application sent about its status, the requests sent by the visualization and their responses. The possible requests from visualization to the application are: *run*, *pause*, *IOResult* and *ISOperation*. The first asks the system to proceed normally. The second halts the application until new request is received. The *IOResult*-operation creates an I/O result and lets the application process it. The *ISOperation* is used to query and modify the Information Storage of the application.

Each request has also one of three possible synchrony modes. The request can be *asynchronous*, i.e., the call returns immediately and the middleman stores the request until the application connects to it. If mode is *synchronous* the call returns when the application has received the request. If mode is *waitForResponse*, the call returns when the application has replied to the request, for example, returned a result for an information storage query. Different modes can be used to optimize the speed and responsiveness.

4. Example visualizations

We have experimented with different uses of visualization framework. In demonstrations a basic visualization has been used and we have experimented with software tutors based on the visualization technique.

4.1. Visualizations for demonstrations

There are endless possibilities to visualize dialogues. We have used simple speech balloon visualizations in demonstrations. This set of components visualizes dialogues in real time with balloons as seen in Figure 2.

The visualization is used in demonstrations where prewritten slides with a sample dialogue have been used previously. The visualization is an important part of demonstrations. It provides spectators the basic view of the interaction; they can easily see what was said and how turns are taken. Spectators have something to look at and can easily follow the interaction. Visualization can also help if spectators have troubles hearing. More importantly, a visual representation of the spoken dialogue provides a persistent view. While speech disappears after it has been spoken, visualization remains on the screen. The demonstrator can refer to the dialogue to emphasize features of the system and spectators can read the dialogue back and forth.

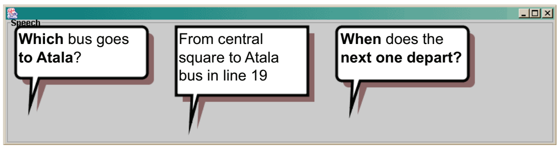

Speech
Which bus goes **to Atala**?
From central square to Atala bus in line 19
When does the **next one depart?**

Figure 2: Balloon visualization of spoken interaction.

It is possible to develop more application specific visualizations to demonstrate some particular features, for example, recognition confidences or a user model. In Figure 2 the keywords of user input are highlighted by bold facing them. The application specific visualizations do not require any modifications to middleman component or the application's connection components because these provide access to systems information storage by default. We have also implemented a component that makes it possible to stop the visualized application temporarily by clicking a pause button in the visualization. This is a handy feature in demonstration situations when you need to pause the application to explain something and is needed in particular when debugging applications.

4.2. Debugging tool

When applications are debugged, tools for monitoring the system are required. In Jaspis-based systems, where all persistent data is held in Information Storage, a view to the storage is vital. Traditionally we have used a simple passive view of the information storage. This component is included into system configuration and it opens a window in the screen when the application is run. Developer must manually update the view and can then view the information storage content as a tree. The visualization system can provide access to the same information with benefits of distribution. The visualization can now run on a separate computer when it previously had to run on the same server as the application. The visualization can also be an active one, i.e., it can automatically update when something happens in the system. When a developer learns to read such animation type visualization, it is possible to instantly see what is happening in the system. The developers of speech based systems can benefit from such tools especially since testing applications is often quite challenging because of the probabilistic nature of speech recognition. The modular structure of the visualization tool makes it easy to add application specific visualizations to different applications. These can later develop into visualizations that can be used to demonstrate the application.

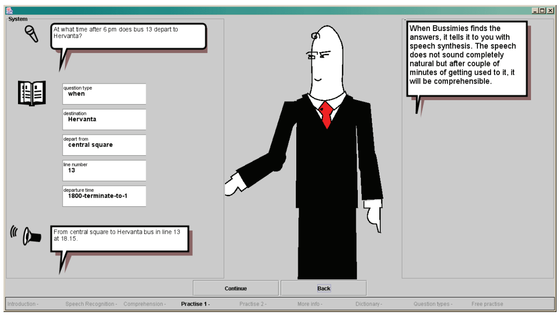
4.3. Software tutoring

The most advanced use of the visualization framework that we have experimented with is software tutoring. In these systems, a software component teaches the users how to use a spoken dialogue application. We have developed several versions of a software tutor to an e-mail system AthosMail [10], and a bus timetable system Busman. In the e-mail system a speech-only tutor was used successfully [11]. In the Busman application we used a multimodal tutoring. The tutor is implemented as a visualization component and it controls the application by stopping it while instructions are presented to the user. The tutor monitors the system status. For example, it looks at input results to see if there have been speech recognition errors and uses this information to provide appropriate guidance. The tutoring asks a user to give some specific inputs to the system so that it can unambiguously recognize error situations and make sure that the user can give inputs to the system successfully.

The tutors include real time visualizations of the spoken interaction. The visualizations enable users to see speech recognition results and see when there are errors. This helps the users to understand the system behavior. The tutors are intended to be used when a system is used for the first time. A tutoring session last 10 to 20 minutes depending on how fast a user proceeds and if there are problems in the interaction.

4.3.1. *Experiences with tutoring*

We have developed the multimodal tutoring iteratively. From the numerous possibilities we selected four different concepts to be implemented. One used basic speech balloons for visualization and text for actual tutoring. Another one added visualization of forms used by dialogue management to provide more insight into the system. Third version had a graphical user interface that had the same functionality as the Busman system. By using the GUI, users could see what kind of queries they can make to Busman with speech. The fourth version, as seen in Figure 3, had a more graphical presentation; the tutoring was presented by a cartoon character and system activity was visualized with animated icons and balloons.



At what time after 6 pm does bus 13 depart to Hervanta?

question type: when
destination: Hervanta
depart from: central square
line number: 13
departure time: 1800-terminate-to-1

From central square to Hervanta bus in line 13 at 18.15.

When Bussimies finds the answers, it tells it to you with speech synthesis. The speech does not sound completely natural but after couple of minutes of getting used to it, it will be comprehensible.

Figure 3: A Multimodal tutor for a spoken dialogue system.

Nineteen first year students without prior experiences on spoken dialogue systems tested the tutors. The balloon type visualization worked very well; users liked the fact that they saw the recognition results and also the systems outputs. When recognition errors were visible users noticed errors and often adjusted their speaking style, if necessary, to get better results. Users were also able to see what kind of errors the speech recognition makes. The other visualizations, especially the forms representing the systems internal logic, were not received so favorably and were often considered to contain unnecessary technical detail.

The technical structure of the visualization framework was very efficient for tutoring. We could monitor all the necessary information in the system. It is possible even to modify the system status, for example, discard misrecognized input. We made a design decision not to do this so that users would see how the system responds when there is a recognition error. The synchrony that the visualization framework provided was timely enough for the tutoring.

Based on the results we have further developed the basic balloon tutor and the GUI based tutor. The next step is to compare the tutors to a guidance that is not connected to the application to see, if the visualization indeed improves the learning results.

5. Conclusions

We have developed a visualization framework for spoken dialogue applications and found efficient uses for the visualizations. The framework has enabled us to use visualizations when demonstrating existing systems and it provides means to build advanced debugging tools. We have also experimented with rather complex software tutors implemented as visualization components. The architecture has efficiently provided all the functionality we have needed in these systems.

6. References

[1] Boda, P. "Visualisation of spoken dialogues", *Proceedings of ICSLP-2000*, vol.1, 146-149, 2000.

[2] Yang, L.-c. "Visualizing spoken discourse: Prosodic form and discourse functions of interruptions." *Proceedings of 2nd SIGdial Workshop on Discourse and Dialogue*. 2001

[3] Dybkjær, L. & Bernsen, N. O. "Towards General-Purpose Annotation Tools - How far are we today?" *Proceedings of the Fourth International Conference on Language Resources and Evaluation LREC'2004*, Vol. I, 197-200, 2004.

[4] Cole, R. "Tools for research and education in speech science", *Proceedings of the International Conference of Phonetic Sciences*, San Francisco, CA, 1999.

[5] Sturm, J., Bakx, I., Cranen, B., Terken, J. & Wang, F. "Usability evaluation of a Dutch multimodal system for Train Timetable Information", *Proceedings of LREC*, 2002.

[6] Terken, J. & te Riele S. "Supporting the construction of a user model in speech-only interfaces by adding multimodality", *Proceedings of Eurospeech 2001 Scandinavia,* Vol. 3, 2177-2180, 2001.

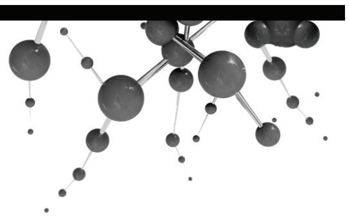
[7] Kamm, C., Litman, D. & Walker, M. "From Novice to Expert: the Effect of Tutorials on User Expertise with Spoken Dialogue Systems", *Proceedings of ICSLP-1998*, 1998.

[8] Seneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P. & Zue, V. "Galaxy-II: A Reference Architecture for Conversational System Development", *Proceedings of ICSLP 1998*, 1998.

[9] Turunen, M. & Hakulinen, J. "Jaspis - A Framework for Multilingual Adaptive Speech Applications", *Proceedings of ICSLP 2000*, 2000.

[10] Turunen, M., Salonen, E-P., Hartikainen, M., Hakulinen, J., Black, W.J., Ramsay, A., Funk, A., Conroy, A., Thompson, P., Stairmand, M., Jokinen, K., Rissanen, J., Kanto, K., Kerminen, A., Gambäck, B., Cheadle, M., Olsson, F. & Sahlgren, M. "AthosMail – a multilingual Adaptive Spoken Dialogue System for E-mail Domain" *COLING Workshop Robust and Adaptive Information Processing for Mobile Speech Interfaces*, 2004.

[11] Hakulinen, J., Turunen, M., Salonen, E.-P. & Räihä, K.-J. "Tutor Design for Speech-Based Interfaces", *Proceedings of DIS2004,* 155-164, 2004.



Paper VI

Jaakko Hakulinen, Markku Turunen, & Esa-Pekka Salonen. Software Tutors for Dialogue Systems. In *Proceedings of Text, Speech and Dialogue* (TSD 2005), LNAI 3658, Springer, pages 412-419.

Software Tutors for Dialogue Systems

Jaakko Hakulinen, Markku Turunen, and Esa-Pekka Salonen

Department of Computer Sciences
University of Tampere
Finland
{jaakko.hakulinen, markku.turunen, esa-pekka.salonen}@cs.uta.fi

Abstract. We have used text, graphics and non-speech audio to tutor new users in a spoken dialogue system. The guidance is given by a software tutor, a software component that interactively tutors the user. Four different variations of tutoring were implemented and experiences were collected from user tests in order to gain insights into these tutoring concepts. Real-time visualization of speech interaction with comic book style balloons and structured guidance were received well while various other methods received mixed acceptance.

1 Introduction

While speech is a natural way of human to human interaction, speech interfaces may become inherently easy to use only when computers understand speech like a human being. Such ultimate speech systems are not possible with today's technology and therefore guidance is needed [3]. Currently speech interfaces lack the fluency and unlimited language of human interaction. Only when users know what each system understands speech applications can be successful services.

Providing on-line help is challenging in speech interfaces. Embedded assistance on the other hand is common and is realized as hint type guidance, expanding system prompts and system initiated dialogue [9]. Many speech application use traditional manuals as well, commonly in the form of web pages.

One option for initial user guidance is a software tutor. A tutor provides guidance within the actual software application, monitors user actions and is capable of providing help in appropriate context and according to users' needs [1]. There are some studies of tutorials in the context of speech interfaces. Kamm et al. [4] studied a tutorial that was not connected to an application but showed in a web page how an example task can be carried out. Consistently higher user satisfaction ratings were found in the tutored group compared to embedded assistance only. We have found an actual software tutor in speech interfaces to be effective as well. Tutoring was delivered in between system prompts and enabled users to learn a speech interface with significantly fewer problems than with a web manual [2].

Incorporating text and graphics has potential benefits of tutoring in a speech interface. Separate tutoring modality widens communication channel and tutoring can happen simultaneously and in synchrony with the system actions [6].

Graphics can also stay on screen as long as needed. Graphical presentations can be very powerful, for example, Terken and te Riele [8] conclude that in their study the graphical part of a multimodal interface gave users a mental model of the interface. While graphics are not usually available when speech interface is used, the initial learning often happens in a situation where this is not the case, for example, web pages and Java applets can be available.

In order to effectively guide users to speech interfaces one must understand what the users need to learn. We have identified the following topics: 1) Interaction style; when and how to speak and turn taking. 2) The error prone nature of speech recognition. 3) System functionality. 4) Language models i.e. what kind of inputs the system understands.

In particular, tutoring must consider recognition errors. If a tutor can detect an error, it should provide guidance that helps users identify the error situation and correct it. One option is to visualize the recognition results and thus help the users to detect errors. A tutor can also give explicit instructions for the user to say something and monitor speech recognition results to make sure that the user can give inputs successfully.

In this paper we introduce a set of graphical software tutors that provide guidance to the users of a speech-based timetable system. The tutors use graphics and audio notifications. They are connected to the telephone-based timetable system to visualize the system and monitor users' actions. The next chapter describes the tutors and our experiences with them. The experiences and findings are discussed in the end.

2 Four multimodal tutoring implementations

The tutors teach how to use Bussimies, a telephone service for bus timetables. Bussimies has grammars of around 1500 words with word spotting. Users can express themselves reasonably freely, but they need to know what questions Bussimies can answer, and what concepts, like bus lines and destinations, it knows. Interaction style is mixed-initiative; users can ask questions freely, but when errors occur the system can take initiative.

The four tutor versions share the same four part structure: an initial instruction set, a guided hands-on exercise, a second set of instructions and a free experimentation. Initial instructions introduce Bussimies, explain speech recognition and show some example inputs. During the hands-on exercise a user gives a call to Bussimies and gives it some inputs by following tutor's instructions. The second set of instructions contains more guidance on valid inputs and a summary. Free experimentation with Bussimies is possible in the end.

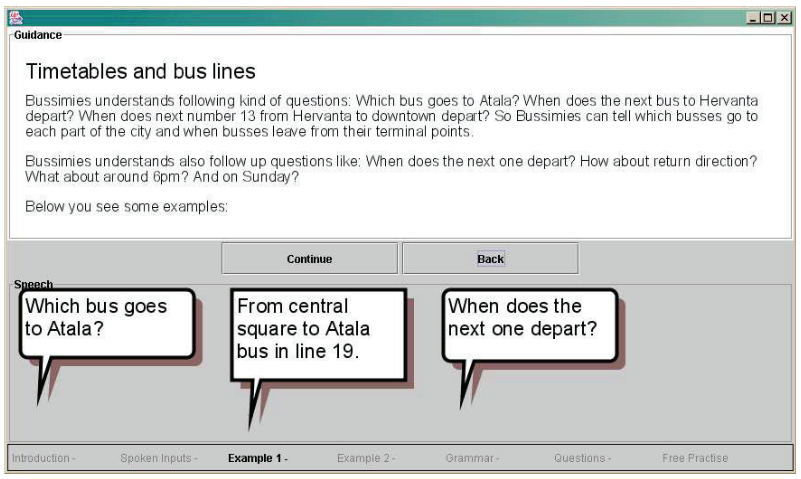
During the hands-on exercise and free experimentation the speech recognition results and systems outputs are visualized. Context sensitive help is given in error situations. Users control the tutor with *Continue* and *Back* buttons.

During the iterative development of the tutors 19 people participated in user tests. All users tested each of the four versions. Observation, questionnaires and discussions were used to collect opinions and findings to guide the development.

The differences between the four tutors are mostly in the visual representations of the Bussimies system. Next, the four variations are discussed with findings from the user tests.

2.1 Balloon tutor

The balloon tutor, as seen in Figure 1, visualizes the spoken interaction with balloons similar to those in comic books. New balloons scroll into screen from right and when more space is needed, the leftmost balloon is removed. The user and system balloons have slightly different shapes. The primary motivation for using balloons is to show the users the results of speech recognition and thus help them to notice speech recognition errors. The visualization of system output should help the users with speech synthesis. Comprehension of speech synthesis often improves after listening to it for a few minutes. During this period, seeing the same information as text can help. Balloons also leave a short term dialogue history visible on screen.



Timetables and bus lines

Bussimies understands following kind of questions: Which bus goes to Atala? When does the next bus to Hervanta depart? When does next number 13 from Hervanta to downtown depart? So Bussimies can tell which busses go to each part of the city and when busses leave from their terminal points.

Bussimies understands also follow up questions like: When does the next one depart? How about return direction? What about around 6pm? And on Sunday?

Below you see some examples:

Which bus goes to Atala?

From central square to Atala bus in line 19.

When does the next one depart?

Fig. 1. Balloon tutor

Balloons are an efficient visualization of a dialogue since people are familiar with the concept from comics and associate balloons with speech. Dialogue turns are also naturally visualized.

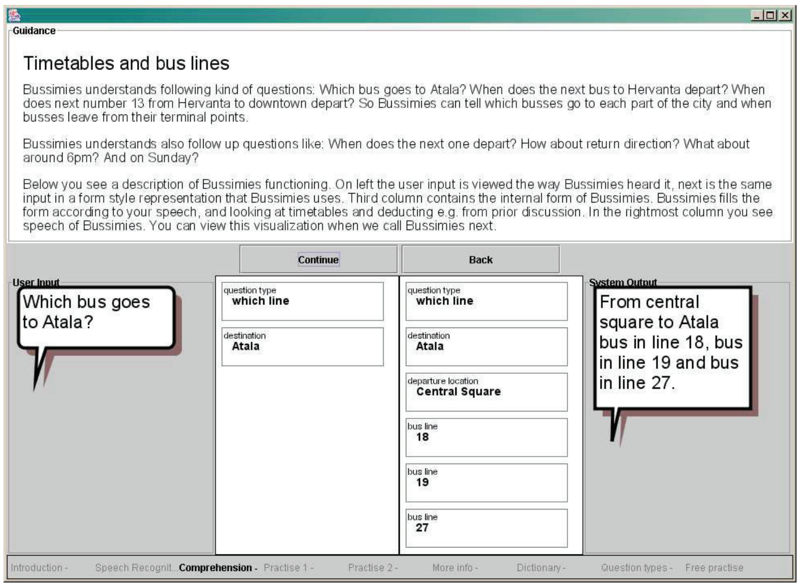
In the user tests, the balloon tutor was received very well. Mostly people liked the tutor because of its simplicity; the balloons were considered easy to understand and follow. The idea was familiar and the movement from right to left provided the feeling of an advancing dialog. The timing of the balloons that appeared slightly before Bussimies started speaking also seemed effective. The visualization matched the flow of the spoken dialogue very well.

The users were able to follow the performance of speech recognition using the balloons. In error situations they knew what had happened and some of those who had problems with recognition, reported that they used this information to adjust their way of speaking to get better results. The fact that they could see how Bussimies had recognized their speech was often reported to be the best feature of the tutoring.

The only negative comments on balloons concerned the animation; some participants found the movement a bit confusing.

2.2 Form tutor

The form tutor, as seen in Figure 2, visualizes a form that Bussimies uses in dialogue management. In addition, another form shows users' input as form items. Speech recognition results and system outputs are visualized as separate balloons. The idea is that revealing the systems internal representation of the queries provides the users with an accurate mental model of the system. This model should tell what kind of questions the system understands.



Timetables and bus lines

Bussimies understands following kind of questions: Which bus goes to Atala? When does the next bus to Hervanta depart? When does next number 13 from Hervanta to downtown depart? So Bussimies can tell which busses go to each part of the city and when busses leave from their terminal points.

Bussimies understands also follow up questions like: When does the next one depart? How about return direction? What about around 6pm? And on Sunday?

Below you see a description of Bussimies functioning. On left the user input is viewed the way Bussimies heard it, next is the same input in a form style representation that Bussimies uses. Third column contains the internal form of Bussimies. Bussimies fills the form according to your speech, and looking at timetables and deducting e.g. from prior discussion. In the rightmost column you see speech of Bussimies. You can view this visualization when we call Bussimies next.

User Input: Which bus goes to Atala?

question type: which line
destination: Atala

question type: which line
destination: Atala
departure location: Central Square
bus line: 18
bus line: 19
bus line: 27

System Output: From central square to Atala bus in line 18, bus in line 19 and bus in line 27.

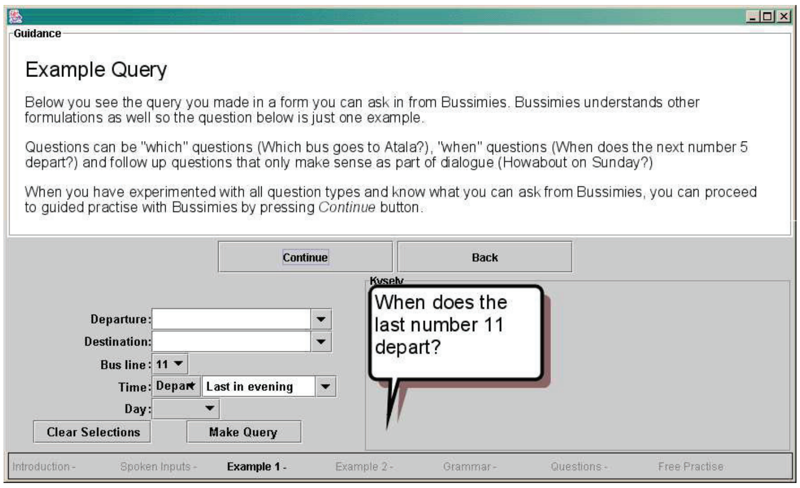
Continue
Back
Introduction - Speech Recognit... Comprehension - Practise 1 - Practise 2 - More info - Dictionary - Question types - Free practise

Fig. 2. Form tutor

The form tutor was favored by some participants because it provided a way to see how Bussimies understands the user inputs. However, many of the participants found the forms useless and irrelevant. They were also considered complicated and technical. One participant wrote: "[The form] Clarifies the understanding [...], but on the other hand requires a terrible amount of concentration."

2.3 Interactive GUI tutor

The interactive GUI tutor has a graphical user interface (GUI) that users can construct queries with. As seen in Figure 3, the resulting natural language queries are shown to the users in balloons. The tutor formulates the queries into a phrasing that Bussimies accepts. The balloons also provide the visualization of dialogue. One of the strengths of GUI is that the possibilities and limitations of the system can be seen from the interface. In the case of Bussimies, the users can see from the GUI what kind of queries can be made with speech.


Guidance

Example Query

Below you see the query you made in a form you can ask in from Bussimies. Bussimies understands other formulations as well so the question below is just one example.

Questions can be "which" questions (Which bus goes to Atala?), "when" questions (When does the next number 5 depart?) and follow up questions that only make sense as part of dialogue (Howabout on Sunday?)

When you have experimented with all question types and know what you can ask from Bussimies, you can proceed to guided practise with Bussimies by pressing *Continue* button.

Continue | Back

Kysely

When does the last number 11 depart?

Departure:	
Destination:	
Bus line:	11
Time:	Depart Last in evening
Day:	

Clear Selections | Make Query

Introduction - Spoken Inputs - **Example 1 -** Example 2 - Grammar - Questions - Free Practise

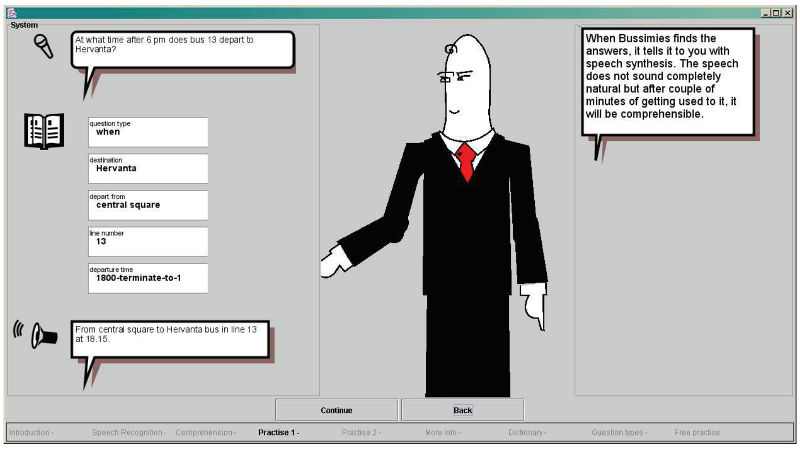
Fig. 3. GUI tutor

The participants liked the interactive GUI tutor because it allowed them to do things themselves. The interactive GUI tutor also best communicated the features of Bussimies to some participants. Many commented that after using the GUI they found Bussimies to have more features than they previously thought.

However, not all participants found the GUI useful. Partly this could be because the users tended to fill in the whole GUI form once and proceed. Therefore they did only see one type of question. Only a small proportion of the participants experimented with the GUI to find out the possibilities and features of Bussimies. The lack of interest in the GUI may be because the GUI was not available when the users freely experimented with Bussimies. The design of the GUI did not clearly suggest all the possibilities either.

2.4 Animated tutor

The animated tutor, as seen in Figure 4, has two animated features: on the left side the system components of Bussimies are visualized and on the right a human like character does the tutoring. There are visualization icons for speech input, speech output, database and a form to visualize dialogue management. The icons are animated when the corresponding system component is active. The visualization was supposed to show the users what the system is doing at any given time.



At what time after 6 pm does bus 13 depart to Hervanta?

question type
when

destination
Hervanta

depart from
central square

line number
13

departure time
1800-terminate-to-1

From central square to Hervanta bus in line 13 at 18.15.

When Bussimies finds the answers, it tells it to you with speech synthesis. The speech does not sound completely natural but after couple of minutes of getting used to it, it will be comprehensible.

Fig. 4. Animated tutor

The motivation behind using a human like character is "persona effect"; users find interface with an anthropomorphic character more helpful, credible and entertaining [5]. The character in the tutor was animated so that it pointed and looked at the icons. Additionally, the character randomly rotated its head and torso a little bit all the time. These idle-time acts [7] are supposed to tell the users that the character is "alive" and the system is responsive.

The participants who preferred the animated tutor referred to the character in a way compatible with the persona effect, e.g.: "[It] felt like there was somebody helping." However, the character received negative comments from the majority of the participants; it was considered foolish and annoying. Opinions on the visual design seemed important to the acceptance of this tutor. A feature like this should be optional and easy to get rid of if one finds it useless or annoying.

The visualizations icons were in general considered helpful. However, the initial versions that consisted of abstract visualizations of speech recognition and synthesis were considered unnecessarily complex and even scary. Simplified versions seen in figure 4 were accepted by the participants. The complex layout of the animated tutor was also found problematic as there was information on both sides of the screen. The users did not always know where to look at.

3 Discussion

The concept of graphical tutoring was received well in the user tests. Most users reported that they learned to use Bussimies with the tutors. Each participant selected a favorite tutor; balloon tutor was selected 5 times, form tutor 6, the interactive GUI tutor 6 and the animated tutor 2 times.

The interaction style tends to be the most challenging topic to teach using traditional manuals. The tutors taught this to the participants successfully. All users learnt when and how to speak. If they had problems during the tutoring, the tutors detected it and appropriate guidance was given. Additionally, the balloons enabled the users to notice problems themselves.

The real-time visualization of interaction was valued by the participants and it was often selected as the best feature of the tutoring. The participants understood well the error-prone nature of speech recognition and words like "mishear" were used to describe this. The visualization of recognition results gave the users a possibility to learn what kind of mistakes the system makes, how it behaves when errors happen, and how one should speak to avoid errors.

To some participants the guidance failed to explain the rather open nature of grammars and the word spotting. They thought that the given examples were the only valid phrasings. The form and GUI tutors that were supposed to best provide this information had the problems discussed in the previous chapters that turned many users away from them. In the discussions after the tests the term keyword was often used by the participants to refer to the parts of the input that Bussimies uses. It was suggested that this point of view could be used in the tutors. The tutoring texts have now been modified to use the keyword concept and to better convey the nature of open grammars.

After the tests the keyword concept was further developed: the keywords in the speech recognition result balloons are bolded. The enhanced visualization provides much of the same information as the form tutor but does not look technical, add the complexity or break the timely correspondence between the spoken interaction and the visualization.

One feature of the multimodal interaction with the tutor is focus shift between the speech interface and the graphical tutor. Our initial design, a notification sound from the tutor, worked well. Every time users heard the sound from computer, they knew that there was something to look at on screen.

A theory that rose from the experience is the required correspondence between the spoken interaction and the visualization in the tutor. The moving balloons as well as the animated icons successfully match the dialogue flow. The form did not have much correspondence to the dialogue flow and the GUI breaks down the speech interaction even more.

4 Conclusions

We have built four versions of graphical tutoring for a speech interface. From users tests we gained insight on this tutoring concept. In general the multimodal tutoring worked well. Several improvements for the current tutoring concepts were found but the foundation is firm.

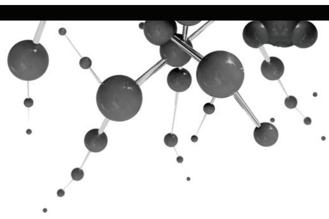
The big challenge with tutoring in speech interfaces is speech recognition errors. The users must not get stuck in the tutoring process due to errors and appropriate guidance must be given to sort out the problems. The visualization of recognition results and context sensitive guidance help users to understand what is going on and to effective adjust to the speech interface when necessary.

The best feature in the tutors was the real-time visualization of the dialogue with balloons. It helped users to notice speech recognition errors and provided an effective view on how the system sees the dialogue. The timely correspondence between the visualization and the dialogue was received favorably.

Already in the tested form, the multimodal software tutoring of speech applications was found a viable concept. It can teach novice users the interaction style and functionality of a speech interface in matter of minutes.

References

1. García, F.: CACTUS: Automated Tutorial Course Generation for Software Applications. Proceedings of Intelligent User Interfaces 2000 (IUI'2000). (2000) 113–120
2. Hakulinen, J., Turunen, M., Salonen, E-P. and Räihä, K-J.: Tutor Design for Speech-Based Interfaces. Proceedings of DIS2004. (2004) 155–164
3. Heisterkamp, P.: "Do not attempt to light with match!": Some thoughts on progress and research goals in Spoken Dialog Systems. Proceedings of Eurospeech 2003. (2003) 2897–2900
4. Kamm, C., Litman, D. and Walker, M.A.: From Novice to Expert: The Effect of Tutorials on User Expertise with Spoken Dialogue Systems. Proceedings of the International Conference on Spoken Language Processing, (ICSLP98). (1998) 1211-1214
5. Lester, J. C., Converse, S. A., Kahler, S. E., Barlow, S. T., Stone, B. A. and Bhogal, R. S.: The Persona Effect: Affective Impact of Animated Pedagogical Agents. Proceedings of CHI 97. (1997) 359–366
6. Nakatami, L.H., Egan, D.E., Ruedisueli, L.W., Hawley, P.M. and Lewart, D.K.: TNT: A Talking Tutor 'N' Trainer for Teaching the Use of Interactive Computer Systems. Proceedings of CHI'86 (1986) 29–34
7. Rist, T., André, E. and Müller, J.: Adding Animated Presentation Agents to the Interface. Proceedings of the 2nd international conference on Intelligent User Interfaces. (1997) 79–86
8. Terken, J. and te Riele, S.: Supporting the Contruction of a User Model in Speech-only Interfaces by Adding Multimodality. Proceedings Eurospeech 2001 Scandinavia (2001) 2177–2180
9. Yankelovich, N.: How Do Users Know What to Say? Interactions, 3, 6 (1996) 32–43



Paper VII

Jaakko Hakulinen, Markku Turunen, & Kari-Jouko Räihä. Tutoring in a Spoken Language Dialogue System. *Technical report A-2006-3*, Department of Computer Sciences, University of Tampere.

Jaakko Hakulinen, Markku Turunen,
and Kari-Jouko Räihä

Tutoring in a Spoken Language Dialogue System

TIETOJENKÄSITTELYTIETEIDEN LAITOS
TAMPEREEN YLIOPISTO

A-2006-3

TAMPERE 2006

Tutoring in a Spoken Language Dialogue System

JAAKKO HAKULINEN, MARKKU TURUNEN, and KARI-JOUKO RÄIHÄ
Tampere Unit for Computer-Human Interaction
Department of Computer Sciences
University of Tampere, Finland

We have developed interactive software tutors to teach users how to use a spoken dialogue timetable system. The tutors teach the functionality and interaction style of the telephone-based timetable system to new users by guiding users and monitoring their interaction. The primary modality of the tutors is graphics and they feature a visual representation of the spoken dialogue between a user and the system. Two different versions of tutoring were compared to a static web manual with the same information in a between-subjects experiment with 27 participants. Participants' evaluations of guidance materials were the most positive towards a tutor featuring a graphical interface representation of the timetable query. An otherwise similar tutor, which did not have the graphical user interface representation, received the weakest evaluations. Error rate variances suggest that tutoring is better than static guidance especially for those who most need guidance.

Categories and Subject Descriptors: *User Interfaces*; H5.2 [**Information Interfaces and Presentation**]: User Interfaces - *Voice I/O, Natural language, Training, help and documentation*;
General Terms: Human Factors, Design, Documentation, Experimentation
Additional Key Words and Phrases: Human-computer interaction, software tutoring, spoken dialogue systems, speech user interfaces, user guidance, multimedia.

1. INTRODUCTION

Advances in speech technology have made it possible to implement useful speech-based applications. Growing emphasis on timely customer services and increased safety provided by in-vehicle information systems, for instance, have radically increased the user base of speech-based systems and services. The take-up of more flexible spoken dialogue systems has been slower, though they, too, have been in public use for more than two decades [Cox et al. 2000].

The challenges of designing spoken dialogue systems are well known, as are the usual solutions. How do the users know the functionality provided by a speech-based system? How do they know what to say to the system? How do they know when to speak? A well designed speech interface supports the users' natural way of speaking. However, in practice the interface must also guide users to speak in a way that the system is able to understand. Implicit and explicit prompts, hints, and tapering are commonly used methods for this. They embed the guidance in the spoken interaction between the user and the system. [Yankelovich 1996]

Spoken dialogue systems can be used heavily by some users and user groups. For vision-impaired users a speech interface may be essential for gaining access to information services. In a mobile context, users of productivity tools, such as email, may also be dependent on their speech-enabled access to information. Characteristic of such usage is that it occurs repeatedly, over a long period of time, and may use the features provided by the system in a versatile manner. Under such conditions, it is plausible that the users are willing to invest some effort into fully learning the possibilities offered by the speech-based service.

How, then, are speech-based systems introduced to new users? When speech is an additional modality in a system that comes with a manual, as is the case with the increasingly common voice control systems in automobiles [Heisterkamp 2001], the speech-based features can be described in the owner's manual. Even when users do not bother to read the manual [Carroll and Rosson 1987], they can discover the voice control possibilities through the graphical part of the car's computer interface [Pieraccini et al. 2004]. Unimodal, telephone-based spoken dialogue systems, on the other hand, need some auxiliary material to introduce them to the users. At least the telephone number and a brief description of what the service has to offer need to be provided to potential users. This is typically done through the web.

Service providers have created materials ranging from static web pages to multimedia presentations with audio examples of the interaction. In addition to introducing the service, users are also often provided with some instructions on how to use the system. Such a web-based tutorial can improve the user experience and users' perception of the system [Kamm, Litman, and Walker 1998]. Compared to guidance embedded into an interface, a comprehensive manual accessible through the web can result in more efficient interaction, since embedded guidance can make repeated interaction tedious.

Another approach to introducing new applications to users is the use of interactive tutoring. This is popular with applications that use graphical interface, particularly in video games, but it has been almost neglected in the case of speech-based applications. However, the tutorial type guidance can also be embedded into a dialogue system, e.g., as a specific guided mode, where new users receive extensive guidance to the systems and old users can use a more effective interface. A guided mode can make the system more transparent to users and thus help them, for instance, in knowing how to correct errors [Karsenty and Botherel 2005]. This kind of guidance can be extended by implementing a software tutor, a separate dialogue partner, which not only guides users but also monitors their interaction and makes sure that the users indeed learn to use the system. We have implemented such a tutor for an email reading application and studied its effect. It was found to reduce the amount of problems users have during the learning period [Hakulinen, Turunen, and Räihä 2006].

Here we follow-up our previous work on unimodal tutoring by studying a tutor that teaches spoken interaction using web-based graphical guidance. The multimedia tutor is connected to a spoken dialogue system so that a user can try out the system under the supervision of the tutor and receive guidance. The resulting multimodal [Oviatt 1999] or multimedia [Bearne, Jones, and Sapsford-Francis 1994] guidance has potential benefits. The visual presentation and GUI-based interaction can be superior to plain speech-based guidance by overcoming the transient and linear nature of speech and its rather low output rate.

We have implemented several versions of a multimedia tutor for a spoken dialogue application: a timetable system with a telephone-based speech interface. The technical challenges in implementing such a tutor that works in synchrony with the application have been discussed by Hakulinen, Turunen, and Salonen [2005a]. The tutors can be run as separate applications or embedded into other web-based guidance and marketing materials as Java applets. Four different concepts for the multimedia representation were developed and compared in earlier experiments [Hakulinen, Turunen, and Salonen 2005b]. The two most promising alternatives were chosen and implemented fully for the controlled experiments reported here.

The research question in this study is whether we can gain some benefits from extending graphical introduction material with interactive software tutor functionality. Can the interactive multimedia software tutoring indeed benefit users and how should the guidance be designed to be beneficial? We have conducted an experiment where the two versions of the interactive graphical software tutor are compared to a non-interactive web page version of the same material. All versions introduce the spoken dialogue system to users and guide them through an elementary scenario. The graphical tutors are connected to the dialogue system, and monitor users' interaction with the system and provide guidance as necessary. For example, after speech recognition rejections, guidance on how to speak to the system is given.

We collected data on users' interaction with the tutor and the dialogue system and users' attitudes towards the guidance materials and the system. An analysis of the measured data did not show significant differences in the task completion rates, but the most troublesome interactions occurred in the web guidance condition. The software tutor with more interaction possibilities was ranked highest in the subjective evaluations, while the other tutor was ranked the worst among the three conditions. Thus, the multimedia tutor can help, but only when designed properly. The graphical form used in the most interactive guidance helps users in understanding the functionality of the spoken dialogue system. The results point out the importance of constructing the guidance material in a manner that closely corresponds to the interaction model of the system: the interface is essentially a form-filling dialogue, and the highest ranked tutor is based on a graphical version of the form.

The paper continues with a description of the two tutors and the web-based guidance material and a discussion of their design rationale. The spoken dialogue system for which the guidance materials were built for is described as well. The experiment is then reported and its results presented. We close by discussing the implications of the work.

2. TWO INTERACTIVE TUTORS AND A WEB MANUAL FOR A TIMETABLE SYSTEM

We study the effects of interactive multimedia tutoring with two different tutors and a web manual. The tutors are graphical software applications run on a personal computer and they communicate with the spoken dialogue application running on a server. For comparison, a web manual has been constructed based on the tutors by removing all interactivity and arranging the information into a static document. All guidance material is in Finnish, as is the spoken dialogue system that the users are tutored in. In this paper the figures and examples of the guidance materials have been translated into English.

2.1 Busman Timetable System

The spoken language dialogue system that the tutors guide users on is called Busman. It is a research prototype of a telephone-based service for Tampere area public transport timetables [Turunen et al. 2005a], implemented in Java on top of the Jaspis framework [Turunen et al. 2005b]. Its users can ask for information on bus lines running between two locations, including information on departure times and routes. Typical utterances understood by the system include “Which line runs from University Hospital to the city center”, “When does the next bus to Hervanta leave”, and “When after six pm does a bus depart from Hervanta to university”. The system uses form-based dialogue management providing limited reasoning based on domain knowledge and dialogue history. Implicit confirmations are used extensively and mostly the interaction is user initiative. System initiative prompts are used for obtaining missing information and after repeated error situations.

The system uses the Finnish language. The speech recognizer is a commercial, large vocabulary recognition engine, Philips SDK with unisex Finnish acoustic models. The language model consists of about 1500 words, and is based on concept spotting. A

Finnish speech synthesizer, Mikropuhe by Timehouse with default male voice, is used for voice output. The system does not support barge-in, i.e., users cannot speak at the same time as the system but only when the system signals that it is listening. However, users can interrupt the system speech by pressing any key on the telephone keypad.

In addition to system initiative prompts, error messages etc., Busman features both short and rather exhaustive spoken help messages. Users can hear these messages by giving respective commands to the system. As an example, the response to a general help request says "This is the Busman system. You can query timetables for buses in Tampere. You can, for example, say 'which bus goes to Hervanta', or 'when does the next bus go to Hallila'. For comprehensive instructions, say 'read instructions'." The help functionality was available to participants during the experiment.

2.2 Tutor Design

The goal of the tutors is to introduce the Busman system to new users. In five to ten minutes, users will learn the functionality of the system and use it by following the instructions given by the tutor. The target group for the tutors is users who are new to the Busman system and possibly to spoken dialogue systems in general.

Since we already have experiences from speech-based tutoring and the new tutors are aimed to be a part of web-based information, the starting point for the tutor design was that the visual modality, which is not used by the Busman system, should be used. The only aural component in the tutors is a notification sound that is played via computer speakers when new tutoring material appears on screen. The sound is vital as it directs users' attention from the application context to tutoring when necessary. The tutors include a visualization of the spoken interaction with comic book style speech balloons. Most importantly, this visualization shows users the speech recognition results as soon as they are available. The aim is to help the users in detecting and understanding speech recognition errors. System utterances are visualized as well, appearing just before the speech starts. This may help the users in learning to understand synthesized voice. Furthermore, the balloons displayed on the screen provide a short dialogue history for users, helping with the temporal nature of speech.

The tutors were presented to users as application windows as can be seen in Figures 1 and 2. In both tutors, the tutor window consists of a guidance area containing textual instructions, *Continue* and *Back* buttons to control the advance of tutoring, the visualization of spoken interaction with speech balloons and an outline of tutoring content in the bottom.

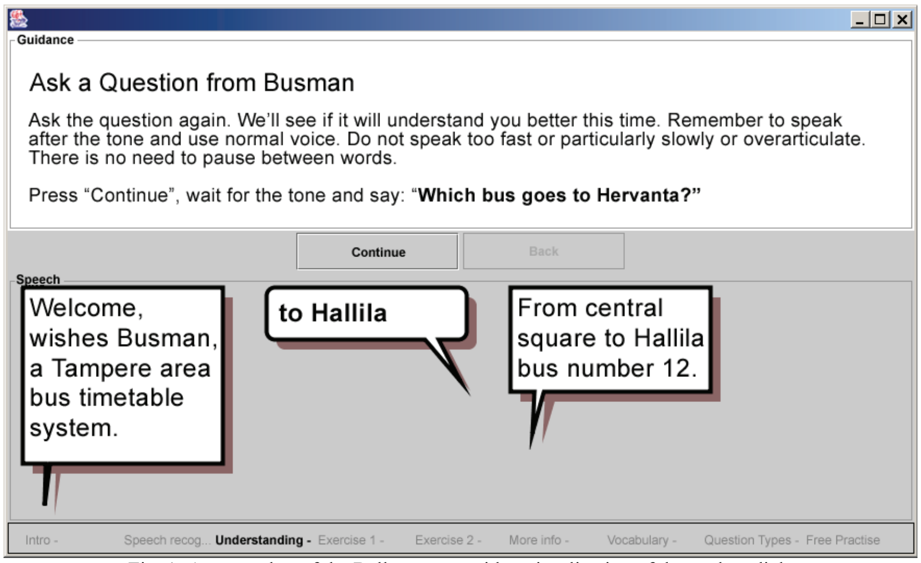
Guidance in both tutors is organized similarly into six segments, each consisting of one text screen. The users move between these by clicking the *Continue* and *Back* buttons. In addition to this text-based information, there is a hands-on exercise part in the middle of the tutoring. In the exercise, users are asked to try out Busman under the supervision of the tutor. This part consists of calling Busman and making three queries. In the end of tutoring, there is a possibility of free experimentation while the tutor is still active, i.e., visualizing the interaction and providing help in explicit error situations, but not directing the interaction. The last text segment before the free experimentation is a summary. Speech balloons are used to visualize spoken dialogue both during the hands-on exercise and free experimentation. They are also used to display an example dialogue before the exercise. The balloons use bold face font to emphasize keywords (the words and phrases the dialogue system actually use) in user utterances.

To support new users, the tutors guide the users step by step during the hands-on exercise. The users are told exactly what to say when they call the timetable system for the first time. The system usage is taught gradually with examples which the users try out themselves during the tutoring. The users control the pace of tutoring by pressing the *Continue* button. The tutors put the Busman system in a paused state as necessary so that the users can proceed at their own pace.

Since speech recognition errors are an inevitable part of speech-based interaction, error-related information is an important part of the tutoring. The tutors monitor the system, which it to display the recognition results in the balloons for users to see how the system has understood user input. During the hands-on exercise, the tutors first ask the users to give a specific input to the system and then monitor the speech recognition results for errors. The error analysis is based on the recognition result and the explicit request made to the user to give a specific input. By comparing the two on word and concept levels, the tutors can spot errors with certainty but the tutor cannot deduct their reason. The tutor does not make guesses; instead, it provides the user with guidance on how to remedy the situation. If the recognition results do not match the required input closely enough, help is given, and the user is asked to try again, simplifying the requested input if some information has already been given successfully. The help provided includes instructions on how to speak, such as to use normal voice and talk after a tone (see Figure 1 for an example). The users are asked to work on a single query for no more than three questions since we have noticed that people get frustrated by error loops at that point. By pointing out errors and providing relevant guidance, the tutors can help users in learning to detect, diagnose, and correct errors. In case of small word level errors, which still result in correct concept level outcome, the tutor points out the error and tells the user that it is not a significant one. For example, when a user has been asked to say "Which bus goes to Hervanta" and the recognition result is "Which to Hervanta", the tutor tells the user "Busman heard your input and understood it correctly. It did not hear exactly 'Which bus goes to Hervanta' but it heard the important keywords and understood that you want to know the bus line number and you are going to Hervanta. Busman does not try to understand every word you say but it tries to find the important parts of your speech."

2.3 Balloon Tutor

The first of the two tutors is the *Balloon tutor,* whose main feature is the visualization of spoken interaction between the timetable system and the user. Comic book style speech balloons are used: regular rectangles indicate utterances of Busman, rounded corners are used for utterances of the user. A snapshot of the Balloon tutor during the hands-on exercise part can be seen in Figure 1.


Guidance

Ask a Question from Busman

Ask the question again. We'll see if it will understand you better this time. Remember to speak after the tone and use normal voice. Do not speak too fast or particularly slowly or overarticulate. There is no need to pause between words.

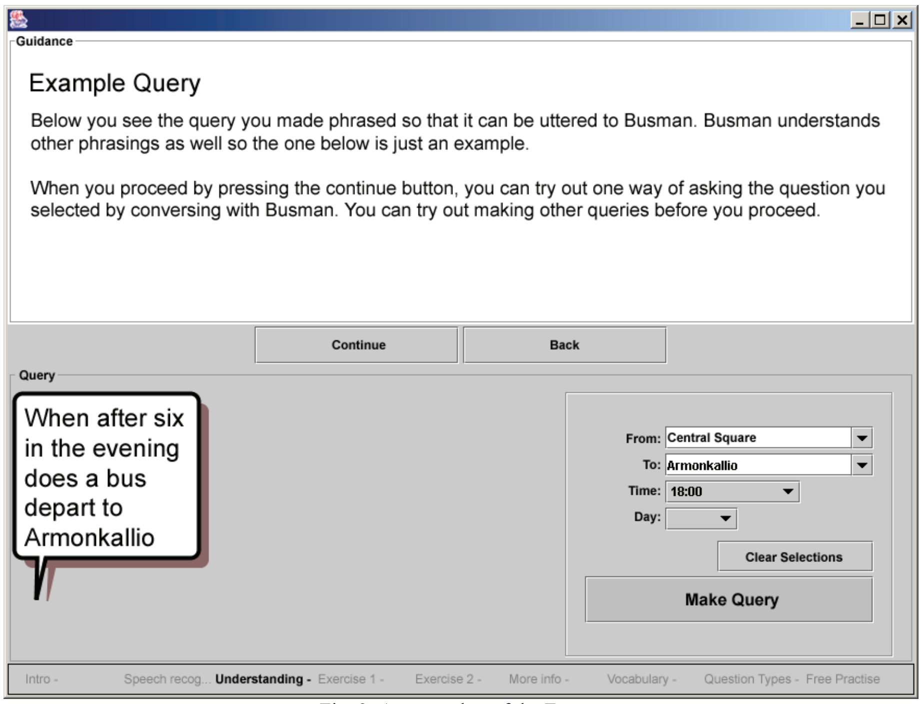
Press "Continue", wait for the tone and say: "**Which bus goes to Hervanta?**"

Continue
Back
Speech
Welcome, wishes Busman, a Tampere area bus timetable system.
to Hallila
From central square to Hallila bus number 12.
Intro - Speech recog... Understanding - Exercise 1 - Exercise 2 - More info - Vocabulary - Question Types - Free Practise

Fig. 1: A screenshot of the Balloon tutor with a visualization of the spoken dialogue.

2.4 Form Tutor

The other tutor is called the *Form tutor*. It includes all the functionality of the Balloon tutor. In addition, it features a form consisting of graphical user interface components, which users can use to create queries that can be asked from the Busman system. The functionality found in the GUI form covers most of the features of the Busman system and therefore can be seen as a visual representation of the timetable system. The Form tutor, including the graphical interface, is shown in Figure 2. The Form tutor features the speech balloons and they are used to present the form-generated queries as well as to visualize the spoken interaction like in the Balloon tutor.


Guidance

Example Query

Below you see the query you made phrased so that it can be uttered to Busman. Busman understands other phrasings as well so the one below is just an example.

When you proceed by pressing the continue button, you can try out one way of asking the question you selected by conversing with Busman. You can try out making other queries before you proceed.

Query

When after six in the evening does a bus depart to Armonkallio

From: Central Square
To: Armonkallio
Time: 18:00
Day:

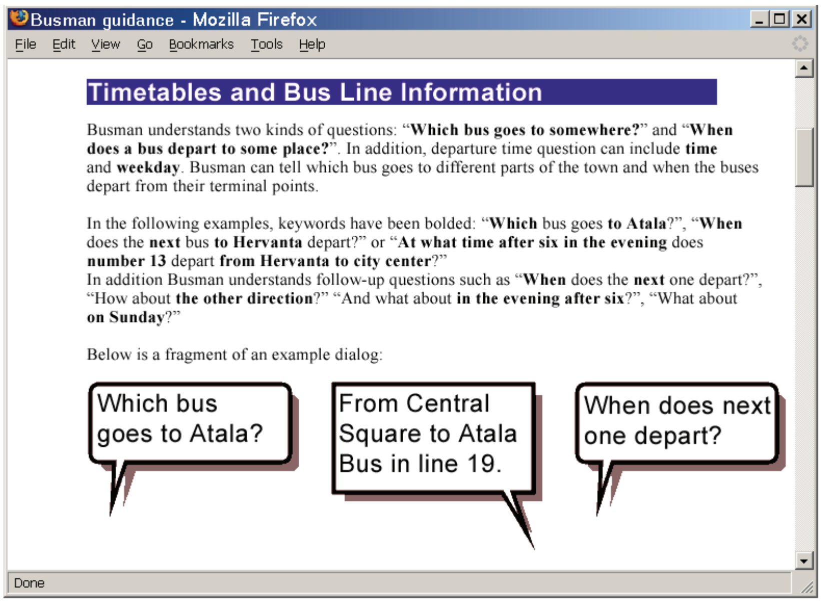
Continue
Back
Clear Selections
Make Query
Intro - Speech recog... Understanding - Exercise 1 - Exercise 2 - More info - Vocabulary - Question Types - Free Practise

Fig. 2: A screenshot of the Form tutor.

Information provided by the two tutors is the same except for the form-specific information, i.e., the examples that the user can generate with the form. The queries the tutors ask the user to make are slightly different between the tutors. In the Balloon tutor, these requests are always the same while in the Form tutor they are based on a query created by the user with the form.

2.5 Web Manual

In addition to the two tutors, a web based version of the same material was created. It contains the same texts and graphics as the tutors as far as possible. However, the material is a single web page and therefore the only interaction users can have with it is to scroll the material up and down. The error related information, such as how to speak, which is presented in error situations in the tutors, is included in the web manual. The information has been slightly edited since error detection is the users' task when the web manual is used. Part of the web manual can be seen in Figure 3.


Busman guidance - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Done

Timetables and Bus Line Information

Busman understands two kinds of questions: "**Which bus goes to somewhere?**" and "**When does a bus depart to some place?**". In addition, departure time question can include **time** and **weekday**. Busman can tell which bus goes to different parts of the town and when the buses depart from their terminal points.

In the following examples, keywords have been bolded: "**Which** bus goes **to Atala**?", "**When** does the **next** bus **to Hervanta** depart?" or "**At what time after six in the evening** does **number 13** depart **from Hervanta to city center**?"
In addition Busman understands follow-up questions such as "**When** does the **next** one depart?", "How about **the other direction**?" "And what about **in the evening after six**?", "What about **on Sunday**?"

Below is a fragment of an example dialog:

Which bus goes to Atala?

From Central Square to Atala Bus in line 19.

When does next one depart?

Fig. 3: Part of the web manual consisting of a single static HTML-page.

3. EXPERIMENT

The tutors and the web manual were used in an experiment to evaluate the effects of interactive guidance in introducing a spoken dialogue application to new users. Data on the effects of the use of the tutors was collected in the controlled experiment with 27 participants. They learned to use Busman under a controlled procedure. Interaction was recorded for later analysis and questionnaires were used to collect participants' opinions on guidance materials and the system.

There were three conditions, one for each guidance material, with 9 participants each. The conditions are called the *Web condition*, where the web manual was used as guidance, the *Balloon condition*, where the Balloon tutor was used, and the *Form condition*, where the Form tutor was used.

3.1 Procedure

The test had a between-subjects setting where each participant learned to use the timetable system in one of the guidance conditions. The participants were randomly assigned to the conditions.

Before the actual test, the participants approved that all calls to the system could be recorded and they filled in a background information questionnaire. The test consisted of a 15 minute learning period with the guidance and a 15 minute period for working with a set of 11 tasks without the guidance material. In the end of the experiment, users filled in two questionnaires where the timetable system and the guidance material used in the condition were evaluated.

The experiments took place in a regular office room where a fixed-line telephone was used to make calls to the Busman system and a desktop computer was used to access the guidance materials and fill in the questionnaires. The experiment conductor was present during the experiment but did not help users in their learning. He intervened only when it was time to move on in the experiment procedure or if technical problems arose.

3.2 Materials

A set of 34 questions known as SASSI (Subjective Assessment of Speech System Interfaces) [Hone and Graham 2000] was used to gather opinions on the Busman timetable system. A set of questions developed to evaluate usability and appeal of user interfaces by Hassenzahl et al. [2000] was used to gather opinions on the guidance. Both questionnaires used seven-item Likert-scale questions. An additional field for open comments was included at the end of both questionnaires. The questions were arranged in the questionnaires in a random order and the direction of scales in respect to their attitude towards the guidance and the timetable system varied between questions. The guidance questionnaire also included six additional Likert-scale questions on the length, amount, and consistency of guidance, resulting in a total of 28 Likert-scale questions. The questions were translated to Finnish for the study. When questions are mentioned in this paper, the original English language versions from the references are used. The questionnaires were presented to users as HTML-forms.

The tasks that the participants used Busman for were given on sheets of paper as maps with an arrow indicating the start and end locations of the trip and a clock face with am/pm next to it. One task description can be seen in Figure 4. In some tasks an abbreviation of Saturday or Sunday was also included, otherwise a weekday was to be assumed. The participants were asked to write down the bus line number for the requested route and a departure time that was near the given time. Place names were included in maps as text in the usual manner but the participants were allowed to also use their own knowledge of the town in selecting names for places. Task descriptions were given this way to minimize the amount of words given to the users. However, the first task was explained verbally to participants when the tasks were introduced to ensure that the task description was properly understood. In total there were 11 tasks starting from straightforward tasks with more complex tasks in the end. Complexities included such situations as lack of direct connections for the trip and departure times late at night when no more buses were running. All task descriptions were given to each participant at once and they were allowed to skip tasks and return to them as they pleased during the 15 minute period.

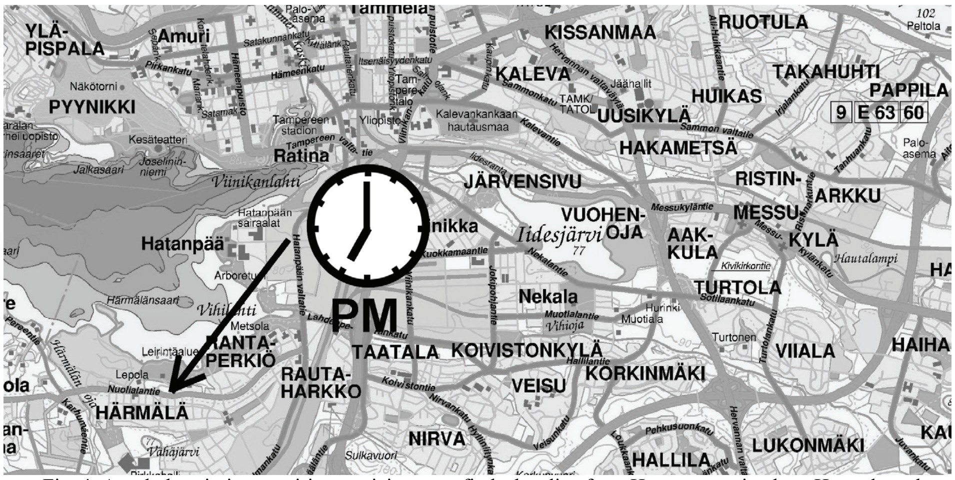


Fig. 4: A task description requiring participants to find a bus line from Hatanpään sairaala to Härmälä and a departure time for the line around 7 PM.

3.3 Participants

27 people without extensive experience in spoken dialogue systems or speech technology participated in the experiment. Their background information is summarized in Table I. The participants' familiarity with the town of Tampere and its public transport were inquired as they were relevant due to the application domain. There were no significant differences on background variables between the conditions. The participants were volunteers who replied to announcements placed on bulletin boards. They received a movie ticket for their participation.

Table I: Background variables.

Age	16-41, average 26.
Sex	10 male, 17 female.
Computer skills	Mostly common users, from inexperienced user to active hobbyist.
Years lived in Tampere	From 0 years to all their life.
Tampere area public transport use	From never used to regular users. Mostly occasional users.
Speech user interface experience	From never used to random usage.

3.4 Results

We have analyzed the task completion rates, which were similar in all conditions, the telephone calls, which reveal a wider variety of error rates in the Web condition, and questionnaires and general observations made during the experiments, which raise the Form tutor as the most highly ranked guidance type and provide some insights into differences between different kinds of users.

3.4.1 Task Completion. Participants' answers to the tasks were analyzed and scored to see if tutoring can teach the usage of the system successfully. There were no statistically significant differences on task completion between the conditions. On average, the participants were working with task number 8 when time was up and had successfully completed 5 to 6 tasks out of the 11 tasks within the time limit in every condition. Some of the earlier tasks were also left blank or given incomplete answers, as can be seen in Table II. In the table, scores given for participants' answers have been summed per condition and task. Scores were given so that completely wrong answers and missing answers received 0, half point was given in cases where only part of the answers was correct (e.g., only bus line but no time) and correct answers resulted in 1 point. None of the participants could give an answer to the final task within the time limit. Users' success with the tasks and the similar task completion rates tell that the tutors can teach the usage just as well as the web manual and the interactivity does not hinder learning. However, the benefits of interactivity cannot be seen in the task completion rates.

Table II: Sum of task scores per task and condition.

Task	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Web condition	6,5	6	6	7,5	6,5	5,5	5,5	4	2	2	0
Balloon condition	6	6	4,5	8,5	6	6	6	3	1	1,5	0
Form condition	6,5	5	6,5	8,5	8,5	6,5	3,5	3	3	2	0

3.4.2 Interaction with the System The participants' interaction with the Busman system both during the training period and during the tasks was recorded and analyzed. The metrics show what kind of problems the participants had and what the effects of tutoring were. Most importantly, users' interaction with tutors seems to be more consistent while some users of a static manual do just fine and others have serious problems.

Speech recognition rates were calculated in the form of concept recognition rates, i.e., as the percentage of the correctly recognized concepts in all utterances. An average concept recognition rate over all participants was 79%. Concepts in the Busman domain are such things as departure place, destination, departure time, help request etc. There were some interspeaker differences in the concept recognition rates, but between the conditions the recognition rates were very consistent as can be seen in Table III.

Table III: Concept recognition rates for the conditions.

	Average	Min.	Max.
Web condition	77 %	64 %	88 %
Balloon condition	79 %	70 %	86 %
Form condition	79 %	74 %	88 %

While there were no statistically significant differences in the error rates between the conditions, the variances of utterance level error rates (i.e., percentage of utterances that did not result in correct system response) between the three conditions were significantly different (Bartlett test of homogeneity of variances, df = 2, $p < 0.05$). The Web condition had the highest variance in error rates while the Balloon condition had the lowest. The difference in variances can be seen in Figure 5 where rates of total recognition error free utterances during the entire experiment have been plotted for each participant. The large variance in the Web condition can be seen especially compared to the Balloon condition.

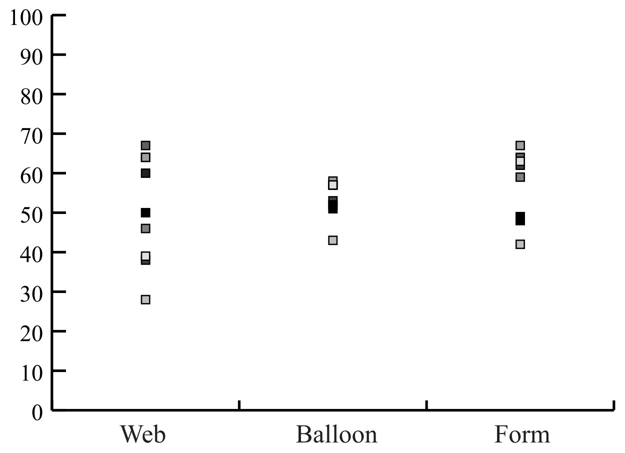


Fig. 5: Total rates of recognition error-free input per participant.

When the training part, i.e., the direct effect of the condition is removed, and only the interaction during the tasks is considered, the error rate distributions become more similar and the difference of variances is no more statistically significant.

In addition to speech recognition errors, voice activity detection (VAD) errors were analyzed. Most of these were cases where end of speech was detected, usually due to a small pause, when a user was still planning to continue the utterance. In such cases, the start of the utterance was often recognized correctly but the user had to repeat in the next utterance the concepts included in the part lost by the system. In general, users were able to detect the voice activity detector errors in all conditions since tones were used to signal both start and end of the period when the system was listening. There were also errors where VAD did not react to user's speech at all and the whole utterance had to be repeated. While the Web condition had more VAD related errors than the other conditions, as can be seen in Table IV, the difference is not statistically significant. Speech recognition errors and VAD errors are independent, i.e., any user utterance could contain no errors, only speech recognition errors, only VAD errors or both.

Table IV: Voice activity detector (VAD) errors as percentage of user utterances affected by an error.

	Average	Min.	Max.
Web condition	26 %	10 %	41 %
Balloon condition	18 %	11 %	29 %
Form condition	18 %	5 %	32 %

Percentages of other types of problems, interruptions of system prompts by the participants using telephone keys, and the total numbers of utterances for the training and task periods can be seen in Table V. The significant differences were the numbers of utterances and the numbers of interruptions during the training ($p < 0.05$ and $p < 0.0005$, respectively, on Kruskal-Wallis one way analysis of variance, df = 2). These differences are directly dependant on the controlled variable, i.e., the training material used. The difference in the amount of interruptions was also significant during the entire experiment ($p < 0.05$) but this can be attributed to the very large difference during the training period.

Table V: Average percentages of utterances spoken at a time when the system was not listening, containing Out-Of-Vocabulary concepts, and followed by an interruption of the system response, and the number of utterances per condition.

	Web		Balloon		Form	
	Training	Task	Training	Task	Training	Task
Wrong time %	2,0 %	1,0 %	4,0 %	3,2 %	6,6 %	1,8 %
OOV %	2,5 %	5,6 %	2,4 %	6,9 %	5,3 %	6,2 %
Interruptions	0,0 %	2,5 %	6,1 %	2,2 %	1,7 %	6,0 %
# of utterance	11	41	23	44	14	41

3.4.3 Questionnaires Participants' subjective evaluations on the guidance materials and the spoken dialogue system were collected using questionnaires. These evaluations provide information on which guidance material the participants liked most as well as some insights into the effect of computer skills and some other background variables on the evaluations.

The guidance evaluation questionnaire resulted in different overall evaluations for the guidance materials. The differences are highly significant (Friedman rank sum test (of evaluation medians), df = 2, $p < 0.001$). Rank sums (higher value – better evaluation) were 55.5 for the Web condition, 39.5 for the Balloon condition, and 73.0 for the Form condition. There were no statistically significant differences between the conditions within single guidance evaluation questions.

There was no significant difference between the conditions on the SASSI evaluation of the Busman system. However, participants' backgrounds correlate with some evaluations, raising the question of differences between different groups of users. Computer skills is a variable that highly significantly correlated (Pearson's product-moment correlation df = 25, p < 0.01) with answers to several questions: "The system is pleasant", "The system is friendly", "The interaction with the system is irritating", "The interaction with the system is frustrating" and "The system responds too slowly". In all cases more experienced computer users considered the timetable system worse, i.e., less pleasant and more irritating. This is understandable as people who are more knowledgeable of computers may have higher expectations of such systems. Their expectations may also be based more on how desktop computers behave. Speech user interface experience correlates also with computer skills (Pearson's product-moment correlation, df = 25, p < 0.05). However, computer skills did not correlate with error levels or task completion rates. Furthermore, the correlations of computer skills were only with system evaluations. There was no significant correlation with the guidance evaluations, which suggests that the tutors, while not equally necessary to, were equally accepted by the different types of users. There was also a difference between male and female participants' evaluations on the question "The system is easy to use". Women agreed more that the system is easy to use. This might be because female users reported lower computer skills than male users. Women had also a slightly lower total error rate in their interaction with Busman, but the difference is not significant.

In guidance questions age correlated (Pearson's product-moment correlation, df = 25, $p < 0.001$) negatively with answers to the question "Guidance was too long", i.e., younger participants considered the guidance too long more often than older ones, suggesting that older users are willing to study more exhaustive guidance material than young users.

4. DISCUSSION

Spoken dialogue systems need to be introduced to new users somehow. Rarely does a potential user receive only a phone number to call to. They usually read some marketing material and other introductory information before they decide to call a system. This material is commonly available in the web, and in addition to marketing material, or as a part of it, the information commonly contains some form of guidance on how to use the system.

In this study, we compared different guidance materials that can be embedded into other web-based material to teach to use of a spoken dialogue system. The materials are aimed to users new to the dialogue system and possibly to spoken dialogue systems in general. Two interactive graphical software tutors and a static web manual were used to teach how to use the spoken dialogue system. The aim was to see if we can gain some benefits by using interactive multimedia software tutoring, and how should such tutoring be designed.

The results indicate that interactive tutoring helps especially those people, who would have most problems learning the use with static guidance materials. While some users can learn to use a system just fine with just a static manual or even without any guidance, others have many problems in learning the style of interaction required in human-computer spoken dialogue. Unlike static guidance, tutors were able to take care of all users. This can be seen in the variations of error rates between the different guidance conditions, i.e., some users in the static guidance condition had much more problems than others while tutors provided learning results that are more consistent. This supports our experiences with speech-only tutoring [Hakulinen, Turunen and Räihä 2006], where the interactive tutor could reduce the number of problems users had during the learning period, making sure that everybody could learn the system smoothly.

We carried out a small follow-up experiment with the multimedia tutors with an additional condition where no guidance material was available. The results suggest that variance in error rates will be even higher when no guidance is available. One user in that condition received the best task completion speed of all participants by learning with just trial and error. Another user in the same group gave up during the learning period. She was given an opportunity to try the system with the Balloon tutor and could proceed successfully. It is also worth mentioning, that especially those, who felt more insecure on using the system, reported that they felt comfortable when they received support from the tutor in the beginning. Tutoring can support users who could not learn the system otherwise, but not all users should be forced to use one.

Since speech is considered a natural interaction method, the success of speech-based systems depends heavily on matching users' expectations. If users assume that a system can understand spontaneous speech like a human, they are very likely to be disappointed. On the other hand, too low expectations can make users ignore the system, or use only a part of its functionality. Tutors support learning by making sure that they use the system correctly. Overall, in speech-based human-computer interaction, error recognition and correction are very important due to the probabilistic nature of speech recognition. Error correction methods and capabilities of spoken dialogue systems are much more limited than those of human-human interaction. Therefore, errors are important part of guidance material as well. By giving explicit instructions to users on what to say, the tested tutors could detect speech recognition problems, which the spoken dialogue system itself is not capable of spotting. In addition, the tutors enabled users to spot recognition errors from the visualization of speech recognition results. Other problems, like speaking so softly that the voice activity detector did not detect any speech, or speaking when the system was not listening, were corrected by tutors very efficiently since the error condition could be analyzed well by the tutor. When interactive guidance is not used, great care must be taken to produce supporting static guidance to helps users to learn error detection and recovery. Naturally, the spoken dialogue system must have sufficient error handling functionality to be taught to users.

Concrete visual example dialogues were a successful design decision, both in the static web page and as visualizations of the spoken dialogue in the tutors. The participants liked the example dialogues presented as speech balloons. They worked well also in the static web manual where participants commonly scrolled the page so that an example dialogue was visible on screen when they called the system for the first time. The solution of presenting the keywords understood by system with boldface in the balloons, while being designed for the real-time visualization, received a positive comment also in the Web condition, where words in static text were bolded. This kind of examples of basic interaction with a system are very easy to produce and can support insecure users when they try to use a system for the first time.

Participants' evaluations of the guidance materials raise the most interactive guidance, Form tutor, significantly above the Balloon tutor and above the static guidance as well. A form with a graphical user interface in the Form tutor, presenting the same functionality as the spoken dialogue system, was the only difference between the two interactive tutors. The benefit of a graphical form is in line with a finding by Terken and teRiele [2001] that a multimodal interface with a graphical query interface provided a mental model that was useful with a speech only interface. The results of our study show that a similar approach featured in the Form tutor was successful and well received by users. The fact that users can transfer skills from a graphical interface to spoken interaction has implications not only in designing introductory material for speech interfaces, but also to developers of multimodal systems. The findings also show that the design and features of interactive tutoring are important; the guidance material should closely correspond to the interaction model of the system.

REFERENCES

BEARNE, M., JONES, S., AND SAPSFORD-FRANCIS, J. 1994. Towards usability guidelines for multimedia systems. In *Proceedings of the second ACM international conference on Multimedia*, San Francisco, CA, USA, October 1994, ACM Press, 105-110.

CARROLL, J. M. AND ROSSON, M. B. Paradox of the Active User. In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*. Cambridge, J. M. CARROLL, Ed., MIT Press, MA, 1987.

COX, R. V., KAMM, C. A., RABINER, L. R., SCHROETER, J., AND WILPON, J. G. 2000. Speech and Language Processing for Next-Millenium Communications Services. *Proceedings of the IEEE*, 88, 8, 1314-1337.

HAKULINEN, J. TURUNEN, M., AND RÄIHÄ K.-J. 2006. Evaluation of Software Tutoring for a Speech Interface. *International Journal of Speech Technology*, 8, 3, 283-293.

HAKULINEN, J., TURUNEN, M., AND SALONEN, E.-P. 2005a. Visualization of Spoken Dialogue Systems for Demonstration, Debugging and Tutoring. In *Proceedings of Interspeech'2005 - Eurospeech — 9th European Conference on Speech Communication and Technology,* Lisbon, Portugal, September *2005*, ISCA, 853-856.

HAKULINEN, J., TURUNEN, M., AND SALONEN, E.-P. 2005b. Software Tutors for Dialogue Systems. In *Proceedings of Text, Speech and Dialogue*, LNAI 3658, MATOUSEK, V., MAUTNER, P., AND PAVELKA, T. Eds. Springer, 412-419.

HASSENZAHL, M., PLATZ, A., BURMESTER, M., AND LEHNER, K. 2000. Hedonic and ergonomic quality aspects determine a software's appeal. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, The Hague, The Netherlands, April 2000, ACM Press, 201-208.

HEISTERKAMP, P. 2001. Linguatronic product-level speech system for Mercedes Benz cars. In *Proceedings of the 1st International Conference on Human Language Technology Research, HLT '01,* San Diego, CA, USA, March 2001. ACL, 1–2.

HONE, K., AND GRAHAM, R. 2000. Towards a Tool for the Subjective Assessment of Speech System Interfaces (SASSI), *Natural Language Engineering, Best Practice in Spoken Language Dialogue System Engineering,* Special Issue, 6, 3 & 4, September 2000.

KAMM, C., LITMAN, D., AND WALKER, M. A. 1998. From Novice to Expert: The Effect of Tutorials on User Expertise with Spoken Dialogue Systems. In *Proceedings of the 5th International Conference on Spoken Language Processing,* Sydney, Australia, November-December 1998. MANNELL, R.H., AND ROBERT-RIBES, J. Eds. ASSTA, 1211-1214.

KARSENTY, L., AND BOTHEREL, V., 2005 Transparency strategies to help users handle system errors. *Speech Communication*, 45, pp. 305–324.

OVIATT, S. L. 1999. Ten myths of multimodal interaction. *Communications of the ACM*, 42, 11, 74–81.

PIERACCINI, R., DAYANIDHI, K., BLOOM, J., DAHAN, J.-G., PHILLIPS, M., GOODMAN, B. R., AND PRASAD, K. V. 2004. Multimodal conversational systems for automobiles, *Communications of the ACM*, 47, 1 Multimodal interfaces that flex, adapt, and persist.

TERKEN, J., AND TE RIELE, S. 2001. Supporting the Construction of a User Model in Speech-only Interfaces by Adding Multi-modality. In *EUROSPEECH 2001 Scandinavia,, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event,* Aalborg, Denmark, September 2001, DALSGAARD, P., LINDBERG, B., BENNER, H., AND TAN, Z-H. Eds. ISCA, 2177-2180.

TURUNEN, M., HAKULINEN, J., SALONEN, E-P., KAINULAINEN, A., AND HELIN, L. 2005a. Spoken and Multimodal Bus Timetable Systems: Design, Development and Evaluation. In *Proceedings of 10th International Conference on Speech and Computer*, Patras, Greece, October 2005, 389-392.

TURUNEN, M., HAKULINEN, J., RÄIHÄ, K-J., SALONEN, E-P., KAINULAINEN, A , AND PRUSI, P. 2005b. An architecture and applications for speech-based accessibility systems. *IBM Systems Journal*, 44, 3, 485-504.

YANKELOVICH, N. 1998. How Do Users Know What to Say? *Interactions*, 3, 6, 32-43.