

Tutor Design for Speech-Based Interfaces

Jaakko Hakulinen, Markku Turunen, Esa-Pekka Salonen, Kari-Jouko Riih 

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].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DIS2004, August 1–4, 2004, Cambridge, Massachusetts, USA.

Copyright 2004 ACM 1-58113-787-7/04/0008...\$5.00.

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]

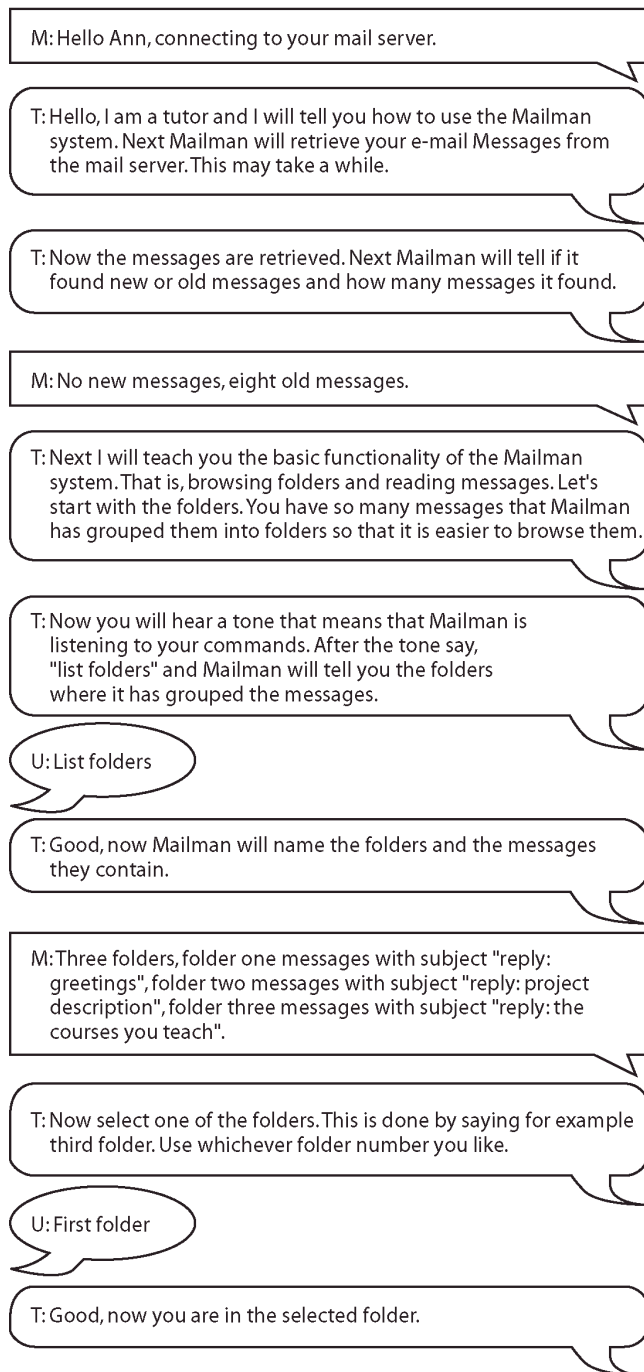


Figure 1: Sample dialogue between the tutor, a user and Mailman.

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 phases 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 tutor-

ing 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

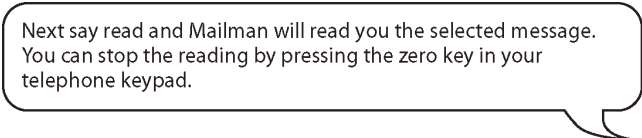
In the second phase we had eight users new to Mailman testing the system and the tutor in a controlled experiment. The users were participants on an introductory user interface course and participating in the test was a voluntary assignment for the course. All the participants were foreign students; 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.

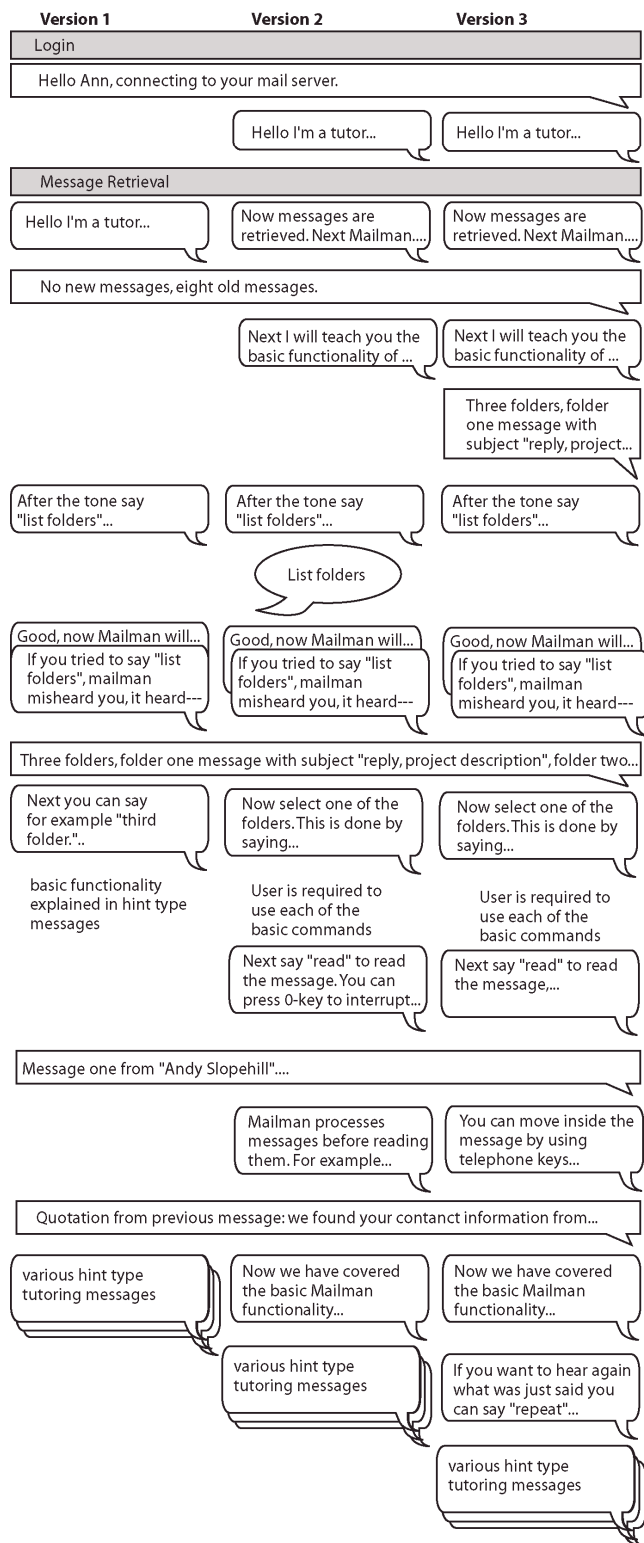


Figure 2: Differences between the tutor versions.

The third version

In the third version some of the tutoring content was re-structured 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 ar-

rived before any long tutor message, such as the description of keyboard commands, was given.

Evaluation

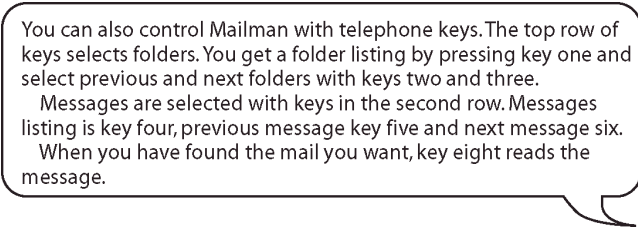
The third version was tested with nine native Finnish speakers. All the participants learned to use Mailman without problems and successfully worked with a set of tasks assigned to them.

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, usu-

ally 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 exam-

ple, 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.

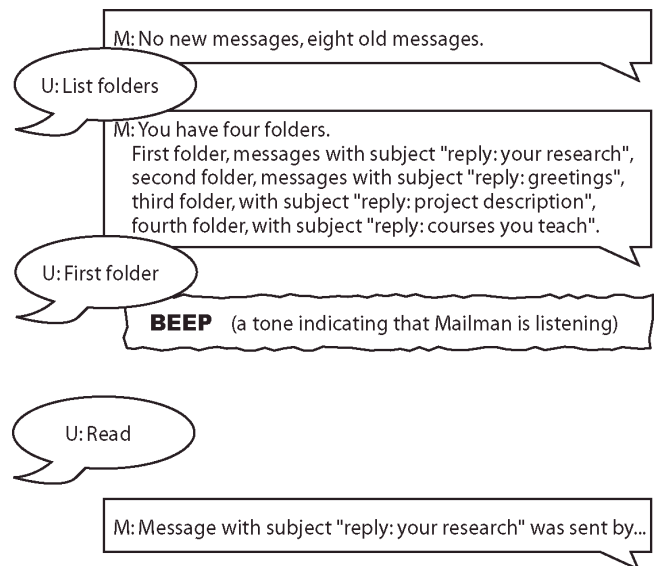


Figure 3: A troublesome interaction without a tutor.

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.

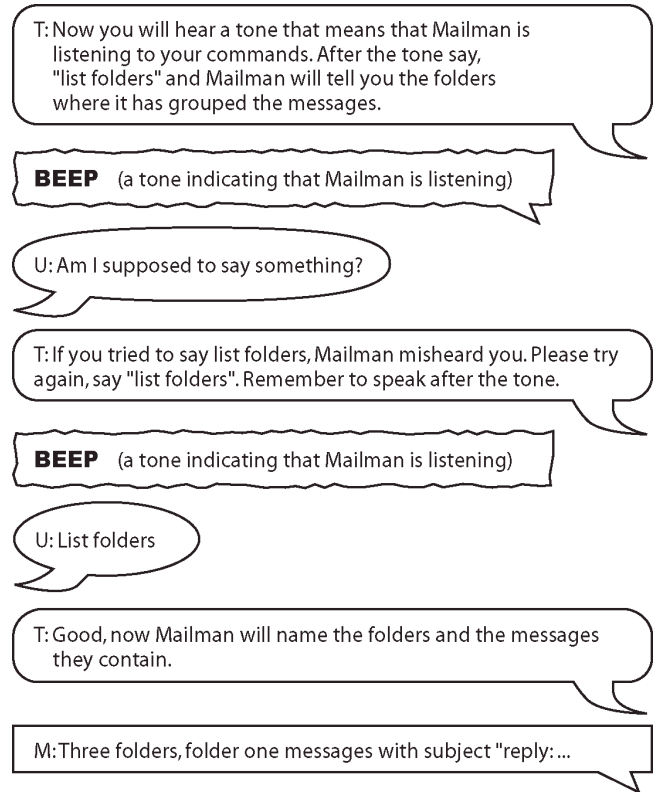


Figure 4: Tutor recognizes a problem.

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.