

# MAILMAN - A MULTILINGUAL SPEECH-ONLY E-MAIL CLIENT BASED ON AN ADAPTIVE SPEECH APPLICATION FRAMEWORK

*Markku Turunen and Jaakko Hakulinen*

Human-Computer Interaction Group  
Department of Computer and Information Sciences  
FIN-33014 University of Tampere, Finland  
{mturunen, jh}@cs.uta.fi

## ABSTRACT

In this paper we introduce Mailman (Postimies), a multilingual speech-based e-mail client. We discuss the most critical and interesting problems found in the domain of multilingual e-mail, such as the problematic presentation of structured elements in a multilingual context, and introduce solutions to them. We also present how our speech application development framework Jaspis, on which Mailman is heavily based, can provide support for multilingual applications. We also give examples of what we have learned from the development of Mailman and its experimental usage. Our current work deals mainly with multilingual speech outputs, but we also introduce how multilingual speech inputs and dialog management are supported in Jaspis and how they are going to be utilized in Mailman.

## 1. INTRODUCTION

The development of a multilingual speech application with current tools can lead to the development of several monolingual applications. Support for actual multilingual systems where, for example, a single output message can contain several languages, is usually missing in the contemporary development systems. The use of multiple languages in the same application at the same time creates a need to support multiple or multilingual speech recognizers, synthesizers and natural language processing engines simultaneously. We also need guidelines on how to handle multilingual problems in speech applications.

Mailman, our telephone based e-mail reader brings to light a lot of problematic multilingual questions. These include deciding when to use what language in different situations, how to handle prosody in different languages, how to adapt dialogue handling to totally different languages etc. Consequently, we had to build a proper basis for multilingual speech applications. This led to the construction of *Jaspis*, an adaptive speech application development framework [6]. Most of the problems that we have addressed during the development and user tests of Mailman, such as the non-trivial reading rules of multilingual e-mail messages, can be solved using Jaspis.

So far our work with Mailman has focused on speech outputs. In this area we have made use of *presentation agents*, which is one of Jaspis's core modules. Presentation agents form an output generation framework which enables a system to have dynamic, adaptive, multilingual and prosody rich speech outputs. To make this possible, speech outputs are presented on a conceptual level inside the system and the natural language form is generated when the message is about to be presented to the user.

erated when the message is about to be presented to the user. This way, we have maximum control over how messages will be spoken. Using conceptual presentation, it is possible to support multilingual speech outputs and have dynamically constructed prosody, which can make speech outputs more intelligible and pleasant for users [1]. The results are much better than just relying on the capabilities of different speech synthesis engines. In this paper we discuss mostly the work done with multilingual speech outputs in Mailman.

Beside speech outputs, we have also addressed the problems of multilingual speech inputs and dialog management. The ability to support multiple dialogue handling strategies cooperatively in a single application makes it possible to construct applications that adapt to the communication customs of different languages and cultures. Multiple concurrent recognizers and input handling strategies provide a way to handle multilingual inputs. Both of these are implemented in Jaspis and the work to include them in Mailman has begun. Here we describe how they are going to be used in Mailman.

This paper is organized as follows. First, we provide a discussion of e-mail as a multilingual domain. Then we briefly describe Mailman and the most important features of the Jaspis framework. Next we present how Mailman utilizes Jaspis to handle speech outputs and interaction. Then we provide examples of how multilingual issues are handled in Mailman. Finally, conclusions are drawn and future work is presented.

## 2. E-MAIL AS A MULTILINGUAL DOMAIN

E-mail provides a challenge for speech application designers with its rich multilingual features. E-mail messages can contain any mixture of different languages in unpredictable combinations. Especially in multilingual countries, e-mail messages often contain several languages. In the following we talk mostly about the situation in Finland which is officially a bilingual country. In Finland many people speak both Finnish and Swedish and in addition English is used very often.

E-mail messages are multilingual in its richest form. Mixtures of different languages can be found in many different combinations. First of all, there are separate e-mail messages in different languages. When people communicate with each other, they use their mother tongue. If two people speak different languages, they usually choose the one that is easiest for both. People also receive a lot of messages from foreign countries. In Finland these messages are usually written in English.

Often, a single message can contain several different languages. It is very common that somebody forwards an interesting e-mail message written in English to Finnish colleagues and writes a little introductory part into the beginning of the message in Finnish. People also sometimes comment on texts written in a different language than their native language. In both of these cases there are different languages in different quotation levels of a single message. Another very common source of multilingual content are e-mail signatures. Many people have signatures in English so that the same signature is appropriate for all e-mails. Therefore, an e-mail that is written completely in Finnish often has an English signature at the bottom.

Sometimes language can change even from sentence to sentence. This however is not so common and usually only happens when people are discussing, for example, some documents that are written in a foreign language. The actual discussion is usually in the users' native language but it may contain, for example, suggestions on how to write something in the language of the document.

In the lowest level, language can change back and forth within consecutive words. This is actually the most common form of multilingual e-mail content. Usually e-mail is written in one language, but there are single words that are in other language(s). These words can, for example, be English terms, technology jargon, names of different applications and terms used in the applications.

One more form of multilinguality can be found in structural elements. With structural elements we mean entities that have some specific, non-linguistic structure. Common structural elements in e-mails are WWW and e-mail addresses, different lists and tables. All these components have some plain text content and some elements that do not have determined lingual form. For example, HTTP addresses often consist of English language words, but sometimes the computer and folder names are not words in any language and they do not have any 'correct' pronunciation. Then there are special characters between these textual components. These characters can be read in any language, so their final lingual form is not initially determined. Similarly, tables usually have some textual components in their cells, but the structure of the table does not have any initially determined verbal presentation.

E-mail is also a challenging domain for speech input handling, since the interaction with multilingual messages often raises needs to handle multilingual inputs even within a single user, for example, a need to handle inputs which may contain names and addresses in different languages. This also affects dialog management.

## HOW TO HANDLE MULTILINGUAL ISSUES

When e-mail messages are spoken to a user using speech synthesis, multilingual content needs to be handled somehow. Feeding English text into a Finnish speech synthesizer does not sound like English at all. Occasionally the results are quite understandable for a Finnish listener, but for English speaking people the resulting mumble would probably be something completely unrecognizable. The results are not any better when an English synthesizer receives Finnish text. Therefore we need to use appropriate synthesizers for each language. The optimal situation would be to have a truly multilingual speech synthesizer, such as

the quad-lingual synthesis by Traber *et al.* [5], so that every different language could be spoken correctly. Even this kind of synthesizer needs to be instructed when to use what language. Until such synthesizers are widely available, we need to use several monolingual synthesizers. In simple cases one solution could be to try to detect words in a language unknown to the synthesizer used and alter their written form so that the synthesizer would read them relatively correctly.

Multiple synthesizers also bring in the problem of multiple different voices. When a voice changes considerably, users may need time to adjust to the new voice. This often annoys users even if it does not affect the comprehension. This is especially true of speech synthesizers, as they have not yet reached the quality of natural speech. Therefore, frequent switches from one synthesis to another can destroy the usability of a system. However, different voices could be also utilized, for example by using opposite sexes in synthesizers with different languages. We did this with Mailman and it seems to help users.

To successfully read multilingual e-mail messages, there are two things we need to do. First, we should recognize the language from the text. This can be done rather easily and reliably on paragraph level. Even something as simple as basic logical rules based on letter frequencies of the text have given us acceptable results. If we want to recognize language on the level of much smaller units like sentences or even single words, we need to use more advanced methods – still, the confidence level decreases.

More interesting, from the viewpoint of usability studies is to decide when to change from one language to another. A simple solution is to always use the right synthesizer to read each part of the text. This is the natural and correct solution when the text units are something as large as paragraphs. But as we reach smaller units, the situation turns into a more ambiguous one. We basically cannot change synthesis between every single word.

There are some situations where changing the language may not be a good solution even if we can handle the changes of synthesizers etc. Structural elements are one case of such situations. For example, web addresses are a very interesting case. These are very often combinations of English words with special characters in between. The special characters are usually spoken and this can be done in any language. The question is, should we use the language of the web address, the language of the text surrounding the web address or the listener's native language. There are also some standard abbreviations, such as "www", and "com" in the web addresses which can be read in any language. Therefore we have lot of material that can be read in any language and some material the language of which we can have a guess at certain confidence.

We give an example of this: how should the web address "java.sun.com" be read to a Finn when it is in a Finnish text? All the words in the address are in English so it could be natural to switch to English for the whole address and also read the dots in English. Or should the dots be read in Finnish, but the words in English if we had a multilingual synthesis. In the case of the address in question it would be perfectly understandable if the address was read entirely in Finnish. All the words are commonly known with people that are likely to receive a mail with such an address and the words are often used in their Finnish forms in daily speech. Also, when the words are read in Finnish they are close enough to the English forms to be understood.

Therefore, the correct solution for this case would be not to change the language from Finnish. However, this cannot be deduced without a significant amount of world knowledge. As the example shows, the problems of multilingual applications are far from trivial.

### 3. MAILMAN

Mailman (Postimies in Finnish) is a multilingual e-mail client designed to provide basic e-mail reading capabilities for telephone users (for both desktop and mobile phone users). E-mail has been a very popular application for speech user interface research since the 1980's (see for example [4]) and it has many properties which make it both suitable and challenging for research purposes. Telephone based interaction is very natural for most of us and the richness and broad scale of e-mail messages provides a great challenge not only for efficient spoken outputs but also challenging opportunities to research different interaction and dialogue strategies.

When we begun the design of Mailman it seemed that the e-mail messages which we were facing contained many problematic issues. We arranged an informal experiment to find out what the main problems were. The most problematic were parts written in different languages and different kinds of structured elements like tables and addresses. In our case the great challenge was to construct a truly multilingual e-mail application which can deal with real life e-mail messages.

Mailman is bilingual, using both Finnish and English. Although Swedish is the second official language of Finland, English is more important for most users. However, we have planned Mailman to support Swedish and possibly other languages. This should not be a major problem since our architecture supports multilinguality.

The basic functionality of Mailman is very straightforward, following other speech-based e-mail clients such as [3]. When a call is received the system asks the user to log in. This input is implemented as DTMF sequences, mainly for security reasons. The most annoying feature from a multilingual point of view is that the first identification prompt must be either in one common language or then all the necessary languages must be spoken. The latter case is not very efficient if there are a lot of users with different languages. We have also planned to add more identification methods, such as speaker recognition and calling-number identification, but DTMF-sequences would always be there at least as a backup-method. One feature to experiment with would be the use of multiple recognizers to detect the language of the user at login situation.

When the user is identified, the system connects to the user's mailbox and retrieves the e-mail headers. After that, the e-mail messages are grouped to folders of at most six e-mails per group. This way the navigation between the messages does not exceed the limits of cognitive capabilities of the users and only a small number of words are needed for interaction. The latter case is important when multiple languages with different technologies are used: the basic interaction methods for all languages can be shared and more advanced interaction methods could be built on top of that if the technologies, most importantly speech recognition, allow that.

After the grouping procedure (which is non-trivial and not optimal in all situations) has finished the classification of the messages, the groups are presented to the user. Only the main

features of the groups are presented at this point. The user is able to select a group and when he/she makes a selection, the information about the messages belonging to the group is presented to the user. When the user selects an individual message, the message is spoken to the user in small segments. The user is able to navigate between the segments. All these features are very common in many previously constructed speech user interfaces for e-mail and in this area Mailman does not provide any new features. Instead of providing a hard-tuned interface for one language and interaction strategy, it is designed to be extensible for multiple languages and interaction strategies.

The first version of Mailman has been in a daily use for over a year now and it has a small voluntary user group consisting of users both in our laboratory and outside it. They have been using Mailman to read their own e-mails with both desktop and mobile phones. We have planned a user study with more users to start before the end of this year.

Most of the work done with Mailman concerns speech outputs. Other work, such as support for multilingual speech inputs and multilingual dialogue management has just begun. We have now implemented the basic architecture in the form of Jaspis, a framework for adaptive speech applications. Next, we describe the fundamental features of Jaspis. After that, we discuss how Mailman uses Jaspis to produce multilingual speech outputs and handles speech inputs and manages dialogs.

### 4. MULTILINGUAL SUPPORT IN JASPIS

One of the main factors behind the difficulty of constructing multilingual speech applications has been the lack of proper tools. In particular, most existing approaches are very naïve when handling multilingual speech outputs and dialogues. It is not an exaggeration to say that building multilingual applications with current tools is a painful process.

First, a framework must support multiple components for different languages. Second, a framework must support components which can be shared between languages. Both of these are needed. When we began the work with Mailman, there existed no proper tools to construct multilingual applications. That was one of the main reasons why we decided to build Jaspis (Java-based adaptive speech user interface development system) to assist in the construction of adaptive, multilingual spoken dialogue applications. In Jaspis, adaptive communication methods are the focus – the whole framework has been designed to enable the adaptation of system components, including output generation, input handling and dialogue management to different users, situations and languages.

Jaspis is based on Java and XML, standard platform independent solutions. We will provide Jaspis as a freely available framework to support spoken language application research and development. Jaspis is described in detail in [6]. Here we are interested in the features that allow the multilingual communication between the user and the system. In order to understand how the multilingual features of Mailman, such as reading e-mails are implemented, we include a brief view to the key components of Jaspis.

In Figure 1, the general system architecture of Jaspis is presented. Jaspis contains two fundamental features that are needed in multilingual applications. First, all information is stored in a single place called *information storage*. A conceptual, language independent form to present information is used where

possible. This is a key feature to adapt applications to different languages and it is crucial for multilingual speech outputs which are generated from a single conceptual form. All parts of the system store all their information in the information storage and therefore each component can utilize all the information that the system contains in their operations. This makes it possible to construct components that utilize information such as dialog history and user profiles in the interaction between the system and the user.

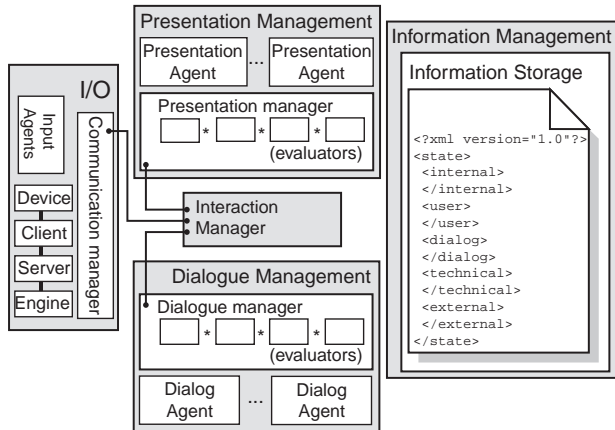


Figure 1: Jaspis framework

The second important feature is the concept of *evaluators and agents*. These components together provide flexibility to adapt to different situations. The agents are system components that handle various interaction situations, such as speech outputs and dialogue acts. The evaluators try to choose the best possible agents to handle each situation. For multilingual speech applications this means that the system can choose the agents that produce speech outputs in a language that the user understands and prefers. This is the duty of a presentation framework which uses *presentation agents* and *presentation evaluators*. Furthermore, it means that dialogue flow, which is in some cases language and culture dependent, can be adapted to the users. This is done by the dialogue management subsystem which is similar to the presentation subsystem and utilizes *dialogue agents* and *dialogue evaluators*. Agents are also used in input handling in the form of *input agents*.

## 5. PRESENTATION OF MULTILINGUAL SPEECH OUTPUTS

We have identified three kinds of speech outputs in Mailman: system utterances, views of e-mails and the e-mails themselves. To efficiently synthesize outputs, we need to know as much about their content as we can. The first type of outputs forms the easiest case because we know in advance what these utterances are. In the second case things get more complicated since we deal with information that is not known in advance. However, the structure of information is still fixed. The third case is the most challenging one since information is totally unconstrained. All of these cases introduce many challenges for multilingual communication in spoken outputs.

Mailman uses presentation agents to convert conceptual messages into speech outputs. The agents do natural lan-

guage generation and add all the necessary speech synthesis control, including the selection between different synthesizers for the generated output messages. In addition to multilingual issues, presentation agents can use prosodic information to make speech outputs more intelligible and pleasant for the user [2]. From a multilingual point of view the use of prosody is more problematic – simple rules do not work, since prosodic elements differ greatly from one language and culture to another. Successful prosody for multilingual applications is a challenge for future work.

There are several presentation agents in each implementation of a Jaspis based system. There can be different output agents for different types of output messages. For example, in Mailman, e-mail messages are read by one agent while other agents take care of messages that inform the user about the system state or give instruction to the user. In the extreme case this approach can lead to a situation where we have a different agent for each type of output message.

In addition to agents that handle different tasks, we can have several agents for a single task. We can write several different types of agents that deliver that same message to the user, but in different ways. For example, some agents can be very brief and others more verbose. In multilingual applications we can have different agents for different languages. In this way we can build multilingual applications in a modular way. Nothing prevents a single agent from being multilingual – this is not however an optimal way to build applications and we prefer to keep agents small and specialized. This supports the reusability and maintenance of complex applications, among other things.

We use evaluators to select the most suitable agent for each situation. When a conceptual system message needs to be presented to the user, an agent needs to be selected for the task. The selection procedure is divided into small units, each implemented as an evaluator. A single evaluation may be, for example, one that checks if the agent under evaluation is at all capable of handling the current message or if the details of an agent's profile match user preferences. Each agent has a profile that tells something about its features. This profile can be compared to the current situation found in the information storage. One of the features of an agent is the language(s) it can speak so that we know if an agent is able to speak the message in a language that the user prefers.

In the case of Mailman, e-mail messages are stored in as conceptual a form as possible. When the system retrieves e-mails from a server, the e-mails are analyzed and conceptualized on a level as high as possible. The languages of the different parts of the message are recognized and the different parts of the message are separated. For example, quotations are recognized. Also, problematic elements, such as web addresses are recognized and conceptually marked. All the recognized information is marked into the message with our own XML mark-up which is language independent and doesn't consider how the recognized components should be presented. We have three reasons why we designed our own mark-up: first, we do not want to tie our system to any particular synthesizers or markup-languages like Sable or JSML. Second, we have found all the current mark-up languages to have insufficient sets of features. Third, we want to use conceptual mark-up in some parts of the messages.

When an e-mail message is going to be read, a presentation agent receives a segment of e-mail that should be read to

the user. As the agent receives the conceptually marked-up version and it has access to all the system information such as the user model and the current situation of dialog, it can now, at the last possible moment, decide how to read each element in the e-mail.

## 6. MULTILINGUAL DIALOGUE MANAGEMENT AND INPUT HANDLING

Different languages and cultures may have very different communication strategies. This shows very clearly in spoken communication. The adaptive dialogue model provided by Jaspis is motivated by this fact. In order to support multiple languages we need to implement a dialogue-handling model which is able to support different communication strategies even if the problems of language interpretation and generation are solved by other parts of the system.

The second motivation for this model comes from the findings that single communication strategies are not suitable for all users. For example, mixed-initiative and system-initiative dialog handling strategies are found to have different kinds of benefits and drawbacks [7]. Thus, a framework for adaptive applications should support multiple dialog handling strategies even if a monolingual communication is used.

Our dialogue management solution tries to provide support for both of the problems mentioned. It begins by abandoning the static dialogue structure and providing instead a dynamic and context-sensitive alternative in the form of dialogue agents. In this model, a single dialogue flow is distributed to multiple dialogue agents which are specialized for certain kind of situations and languages. In every dialogue turn the most suitable agent is selected to handle it. In this way it is possible to support different dialogue management strategies. This is the case, for example, when different languages need to be handled differently on the dialogue level. It should be noticed that since all messages to/from the user are handled on a conceptual level, most dialogue situations can be shared with multilingual applications and only those that need to be handled differently need language dependent dialogue agents. This also allows the construction of reusable dialogue components for common situations such as error correction.

The selection of the most suitable dialogue agents is similar to the selection of a presentation agent. Dialogue evaluators are used to choose the best possible agent based on the overall situation which contains current inputs, dialogue context and user preferences. Often there is only one agent suitable for a particular situation independent on a language – in this way we can build multilingual applications with only a minimal amount of work needed to add a new language to the system.

Dialogue management is aware of the style of the communication. As we stated before, the style of the communication can vary between cultures, languages, individual users and situations. In order to support different communication strategies, such as mixed-initiative and system-initiative strategies, the dialogue manager keeps explicit track of communication style changes. We believe that this could be a very powerful feature in situations where the user population is extremely heterogeneous. One possible scenario would be a public information kiosk serving users with a wide choice of languages and backgrounds.

In addition to speech output presentation and dialogue handling modules, an agent-based approach is also used in the input handling module. Speech input handling uses a model in which multiple recognizers, grammars and error correction strategies can be used simultaneously to adapt to context, user properties and situation. This model is implemented in the form of input agents. For multilingual applications the main feature of input agents is the possibility to construct strategies to handle speech inputs given in multiple languages. Also, it is possible to add to interaction models language dependent features such as error-correction procedures. We are currently working with the input agent concept and building new, more flexible and adaptive input handling mechanisms to Mailman. We believe that it could give reasonably good results in situations where a wide range of users are present. Examples on this are public transportation timetable systems and information kiosks in big cities.

Currently Mailman uses only a small amount of the dialogue management and input handling features that Jaspis has to offer. Since we wanted to investigate what kind of challenges a relatively straightforward application, such as Mailman, can offer to multilingual dialogue management and input handling, we decided to build two different versions, one for Finnish and another for English. First we built the Finnish version with English output capabilities (this feature is too crucial to be left out) and then we designed the English version from scratch. Currently we are investigating what the differences between the two versions are and how they could be implemented using all capabilities of the Jaspis architecture. Then we should arrange user studies to examine how well our adaptive dialogue management and enhanced input handling work in real situations.

## 7. EXAMPLE SITUATIONS

To understand how Mailman (and Jaspis) handles multilingual communication in practice, we give a simplified example on the use of Mailman.

In our example the user has said “*read the third message*” (or “*lue kolmas posti*” in Finnish). This is conceptualized using input agents to the form of “READ MESSAGE [3]” and stored in the information storage. Next dialogue evaluators select a dialogue agent to handle the situation. In our example a dialogue agent that can handle the reading of an e-mail message is selected. This agent can be a compact dialogue fragment that handles only the reading of the messages or it can be a part of a bigger dialogue component.

The selected dialogue agent takes control of the application and decides what to do next. If there are at least three messages in the current folder then the dialogue agent decides to read the message, otherwise an error message will be presented to the user. In the first case the dialogue agent creates an output message in the information storage. This message, like others, is presented in conceptual form which is quite similar in this case as the conceptualized input. In another case, when there is no third message in the current folder, the conceptualized message would be “NO MESSAGE [3]”.

At this point the control shifts to the presentation module. Presentation evaluators choose a presentation agent to speak this conceptual message to the user. If the user prefers to hear the message in English, an English-speaking presentation agent is selected, otherwise a Finnish-speaking agent is selected.

The presentation agent can use all the information found in the information storage to speak the message to the user. For example, if the dialogue history states that the message has been read before to the user, we can safely skip some parts of the header information, which we would read otherwise.

If there is no third message in the current folder the agent can produce something like “*There is no <SLOW>third</SLOW> message at the <EMP>first</EMP> folder*” (or in Finnish “*<EMP>Ensimmäisessä</EMP> kansiossa ei ole <SLOW>kolmatta</SLOW> viestiä*). Another example of the same conceptual message in a situation where there is only one folder could be “*Message number <EMP>three</EMP> does not exist*” (“*<EMP>kolmatta</EMP> viestiä ei ole olemassa*” in Finnish).

When a presentation agent has finished, the formed message is stored in information storage and the control shifts to the communication manager. The communication manager now selects the output devices needed to present this message to the user. In the case of Mailman, a telephone output device is selected. This device uses other devices, such as a synthesizer device, to present the message to the user. The message is converted to the form required by a synthesizer and presented to the user.

The example given is simplified in many respects. First, all information is presented as XML-documents. Here we omit this presentation for the sake of conserving space. Secondly, in order to be efficient, the reading of e-mail messages needs to be more atomic than in our example. When a message is presented to the user it is divided into small segments. Especially different languages and complex structural elements such as tables should not be read like normal text. Also, when an e-mail message is presented to the user there is usually some sort of additional information about the sender etc. For these reasons, reading a single e-mail message is in fact not a single action and means that it should involve multiple dialogue, presentation and input agents. This supports also multilingual issues since specialized agents could be utilized better, for example, when tables and addresses are read to the user.

## 8. CONCLUSIONS AND FUTURE WORK

Successful presentation of multilingual information in speech systems is possible with the combination of conceptual, language independent information storage and output construction that takes advantage of various sources of information in the system. The Jaspis framework supports these with a centralized information storage and presentation agent framework.

As the system provides access to all the information it has to offer, we have several ways to yet improve the handling of multilingual issues. One great source of information that could be used is the user model. If we can form a model of the user’s linguistic skills, for example how well he/she speaks and understands different languages and which of them he/she prefers, speech outputs can be adapted for each user. Material in language unfamiliar to the user could be bypassed with just a notification and ambiguous situations with structural elements could be handled on a reasonable basis.

We should also build a model of the system for itself. Of course the system should know which are the languages it supports. Furthermore, we could give evaluations about the quality and characteristics of the voices of each of the speech synthe-

sizers. It would be helpful to know, for example, how similar two synthetic voices are. As the quality of most speech synthesizers is still not excellent, especially foreign text can be hard to understand. The system could therefore pick the best possible synthesis and way to present the required information in the ambiguous situations so that it would match both the user preferences and the capabilities of the system.

Our future work will include the proper handling of multilingual inputs and dialogue management. In this paper we have discussed how this support can be built. Next we are going to implement these features in Mailman and arrange user tests to see how they work with real users. It is extremely important to study how users really behave when multilingual speech inputs are available. Not only is the proper handling of speech inputs in different languages needed, but also the dialogue management should be adapted to the user and the language. Interesting questions include, for example, how multilingual error-handling works and how language-independent these components can be. We like also to use recognizers with different languages to make the application not only multilingual but also language-independent.

## 9. ACKNOWLEDGMENTS

This work was supported by the Academy of Finland (project 163356) and by the National Technology Agency (Tekes).

## 10. REFERENCES

- [1] Hakulinen, J. and Turunen, M. “Presentation Agents for Speech User Interfaces”. In *CHI 2000 Extended Abstracts*, ACM Press, 2000, 115-116.
- [2] Hakulinen, J., Turunen, M. and Räihä, K.-J. “The Use of Prosodic Features to Help Users Extract Information from Structured Elements in Spoken Dialogue Systems”. In *Proceedings of ESCA Tutorial and Research Workshop on Dialogue and Prosody*, 1999, 65-70.
- [3] Marx, M. and Schmandt, C. “MailCall: Message Presentation and Navigation in a Nonvisual Environment”. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, 1996, 165-172.
- [4] Schmandt, C. “Speech Synthesis Gives Voiced Access to an Electronic Mail System”. *Speech Technology*, August/September 1982, 66-68.
- [5] Traber, C., Huber, K., Nedir, K., Pfister, B., Keller, E., Zellner, B. “From Multilingual to Polyglot Speech Synthesis”. In *Proceedings of Eurospeech 99, Volume 2*, 1999, 835-838.
- [6] Turunen, M. and Hakulinen, J. “Jaspis - A Framework for Multilingual Adaptive Speech Applications”. In *Proceedings of 6th International Conference of Spoken Language Processing*, 2000 (to appear).
- [7] Walker, M., Fromer, J., Fabbrizio, G., Mestel, C., and Hindle, D. “What can I say?: Evaluating a spoken language interface to Email”. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, 1998, 582-589.