Generating speech user interfaces from interaction acts

Stina Nylander, Thomas Nyström
Swedish Institute of Computer Science
Box 1263,
16429 Kista,
Sweden.
Tel: +46-8-633-1500
E-mail: {stny,sthomasn}@sics.se

Botond Pakucs
Centre for Speech Technology (CTT)
Royal Institute of Technology (KTH)
Lindstedtsvägen 24
100 44, Stockholm, Sweden.
Tel: +46-8-790-9269
E-mail: botte@speech.kth.se

SICS Technical Report T2005:13, ISSN 1100-3154, ISRN:SICS-T--2005/13-SE

ABSTRACT
We have applied interaction acts, an abstract user-service interaction specification, to speech user interfaces to investigate how well it lends itself to a new type of user interface. We used interaction acts to generate a VoiceXML-based speech user interface, and identified two main issues connected to the differences between graphical user interfaces and speech user interfaces. The first issue concerns the structure of the user interface. Generating speech user interfaces and GUIs from the same underlying structure easily results in a too hierarchical and difficult to use speech user interface. The second issue is user input. Interpreting spoken user input is fundamentally different from user input in GUIs. We have shown that it is possible to generate speech user interfaces based on. A small user study supports the results.

We discuss these issues and some possible solutions, and some results from preliminary user studies.

KEYWORDS: device independence, multiple user interfaces, speech interaction, interaction acts

INTRODUCTION
We have applied *interaction acts* [13, 14], an abstract specification of user-service interaction, to speech based user interfaces to explore their strengths and limitations in the light of a different type of user interface. Interaction acts have previously proved to be adequate for generating graphical user interfaces and Web user interfaces, but have not been used for speech user interfaces.

Users have a wide range of computing devices and electronic services to choose from today, but it is not at all obvious that the preferred device can be used to access the preferred service. It is not unusual that people use several services from different providers for the same purpose, for example managing calendar information. This seems unreasonable [18]. To facilitate the match between services and devices, service providers need to create services that can be manifested on many different devices [1]. There are several approaches to achieving that: creating several versions of services, each tailored to a specific device; letting the thinnest device set the limitations and use the same presentation on all devices; using an abstract specification of the service or the user interface to generate different user interfaces. Regardless of which solution is chosen, the resulting user interfaces must not be restricted to a single type of user interface or a single modality. Users must have the possibility to choose for example a GUI in some situations and a speech user interface in other situations to access the same service.

We believe that an abstract specification of the user-service interaction to create multiple user interfaces for services. is the most promising solution. We have used interaction acts, the abstract specification used in *the Ubiquitous Interactor* system (UBI) [13]. UBI provides generation of multiple user interfaces to services from an abstract specification, combined with possibilities to feed presentation information for given devices to the generation process. Previously, interaction acts have been used to generate graphical user interfaces (Java Swing, Java Awt, Tcl/Tk) and Web user interfaces [13].

The most interesting issues that were raised in the application of interaction acts to VoiceXML user interfaces was the structural differences between graphical user interfaces, and the handling of user input.

Next we describe UBI in more detail, followed by a discussion on speech user interfaces in general and for UBI in particular. The issues raised by speech interaction are described followed by some preliminary results from user testing. The paper is concluded by a review of related work, and a discussion of future work.

THE UBIQUITOUS INTERACTOR
The Ubiquitous Interactor (UBI) is a system that provides support for device independent development and use of services. In UBI user-service interaction is described in a general way, without any presentation information. Based on the general description, different user interfaces can be generated for different devices. When needed, device and service specific presentation can be fed into the process to tailor the user interface. Since the general description of the

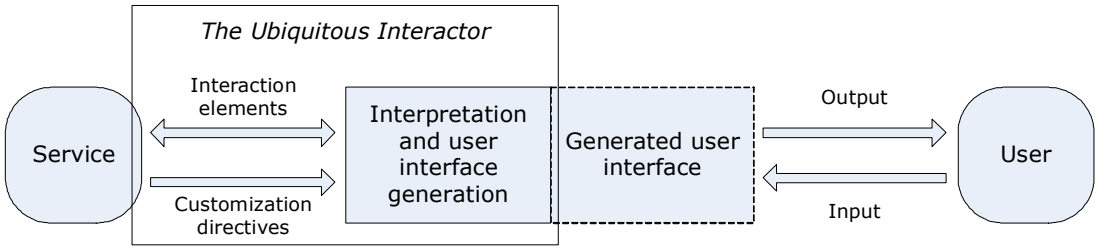

Figure 1: An architectural overview of the Ubiquitous Interactor.

service stays the same for all user interfaces, service providers can develop services for an open set of devices. New devices or user interfaces can be added without changes in the service logic. It also makes it possible for service providers to control how their services are manifested for the end-users, which is very important [5, 10].

UBI consists of three major parts: interaction acts that describe the user-service interaction, customization forms that contain presentation information, and interaction engines that generate user interfaces based on interaction acts and customization forms, see figure 1. The Ubiquitous Interactor was developed as an extension of the sView system [2, 3], and interaction engines as well as sample services for UBI are implemented as services in sView.

Interaction acts

Interaction acts [13] are units for describing user-service interaction without any information about presentation. At present, UBI handles a set of eight different interaction acts: Start and stop handles the beginning and ending of the interaction session, while create, destroy, and modify handles service specific objects (for example meetings in a calendar or avatars in a game). Input handles data input from the user to the system, output handles presentation of data from the system to the user, and select handles choice between a finite number of alternatives. Input and select are mainly used for data that is not stored in the service, for example navigation data. Interaction acts are encoded in the Interaction Specification Language (ISL), which is XML compliant.

Interaction acts are linked to each other in a tree structure. They can be grouped to form units of interaction acts that are related in some way, for example by offering related operations like the rewind, play, and forward operations on a CD player. Groups can be nested. Groups are manifested as nodes in the tree, and interaction acts are manifested as leaves.

The idea of describing the user-service interaction with a small set of interaction acts is based on the assumption that a wide range of interaction can be described with a fairly small set of units. Keeping the set fixed makes it possible to provide default presentations for different user interfaces. However, we do not believe that the current set of interaction acts is the final one. So far we have mostly worked with information services, and new types of services might raise the need for new interaction acts.

Customization Forms

Customization forms [13] contain information about presentation for a given service and a given user interface. They can contain two types of information, directives and resources. Directives are mappings to user interface components as widgets or templates, while resources are links to text, sound, images or other resources that should be used to generate the user interface.

Customization forms are optional. If no customization form is provided, a user interface is generated using default components. A customization form does not need to contain directives or resources for all interaction acts for a service. Those that are not specified are presented with default components. By providing detailed customization forms, service providers can control exactly how their services will be presented to end users.

Interaction Engines

Interaction engines [13] are responsible for interpreting interaction acts and customization forms, and generate user interfaces based on their content. An interaction engine is specific for a type of user interface and a device, i.e. they can handle all services that express themselves with interaction acts, but they can only generate one type of user interface. For example one engine could generate graphic user interfaces for desktop computers, another one graphic user interfaces for handheld devices, and yet another one could generate speech user interfaces. The Ubiquitous Interactor has interaction engines for Java Swing, web user interfaces Tcl/Tk and Java Awt. The Swing and web engines primarily generates user interfaces for desktop screens, while the Tcl/Tk engine generates user interfaces for handheld computers and the Java Awt engine generates user interfaces for cell phones of the Sony Ericsson P800/900 type.

Sample services

A calendar service and a stockbroker notification service [12] have been developed as sample services. The calendar service has customization forms for web user interfaces, Tcl/Tk, and several customization forms for Java Swing. The stockbroker notification service has customization forms for web user interfaces, Java Awt, and several customization forms for Java Swing.

```
<output>
<name>logo</name>
<id>342564</id>
<life>persistent</life>
<modal>false</modal>
<string>SICS AB
</string>
</output>
```

```
<select>
<id>235690</id>
<life>persistent</life>
<modal>false</modal>
<response-number>1</response-
number>
<string>Navigation</string>
<alternative>
<id>98770</id>
<string>Previous</string>
<return-
value>previous</return-value>
</alternative>
<alternative>
<id>66432</id>
<string>Next</string>
<return-value>next</return-
value>
</alternative>
</select>
```

```
<modify>
<id>342564</id>
<life>persistent</life>
<modal>false</modal>
<string>
<m>
<id>1234_567</id>
<s>10.00-11.00 Staff
   meeting</s>
</m>
</string>
</modify>
```

Figure 2: ISL code for an output, a select, and a modify interaction act.

RELATED WORK

The Ubiquitous Interactor (UBI) has its origin in the problem of matching devices and services that has emerged with mobile and ubiquitous computing. Users can choose from a wider range of devices and services than ever before, but it is less than evident that the preferred device can be used to access the preferred service. Much inspiration in solving this problem has been drawn from early attempts to achieve device independence by working on a higher level than device details.

Mike [15] and IST [21] were among the first systems that allowed developers to specify presentation information separately from the application, and thus make changes to the presentation without affecting the application logic. However, Mike and ITS were restricted to graphical user interfaces, and had other important limitations. Mike could not handle application specific data. In ITS, presentation information was considered as application independent and stored in style files that could be moved between applications, something that was not very useful [21]. In UBI, we instead consider presentation information as application specific and tailor it to different devices.

More recent systems like XWeb [16], PUC [11], and SUPPLE [7] respond to the same proliferation of devices and services that UBI addresses. They are similar to UBI in that they all use an abstract specification of the service to generate both graphical and speech-based user interfaces automatically. However, XWeb and SUPPLE do not provide any mechanisms for controlling the presentation of the generated user interfaces, and PUC only provides simple templates to give a certain look and feel to the user interface. There is no possibility to control the presentation of individual units of the abstract description, as the customization forms of UBI allow.

All of the systems described above have focused on user interfaces for control purposes, PUC [11] is mainly used to generate user interfaces to home appliances like VCRs, stereos, and TV sets, while XWeb [16] and SUPPLE [7] mainly generates user interfaces to facilities in buildings like ventilation, heating, and light. These application areas only allow the user to perform pre-defined actions (program the VCR, playing music on the stereo, turn up the heat). This means that no free form user input need to be handled, which is one of the new issues that UBI needed to address when providing user interfaces.

In the speech community, there are different standards for describing the speech-based interaction and enabling the development of speech user interfaces. For instance, VoiceXML (www.voicexml.org) is a standardized markup language used in a large number of telephony-based commercial speech services, while the SALT platform (www.saltforum.org) focuses on web-based and multi-modal applications.

SPEECH USER INTERFACE VS. GUI

Due to the advances in speech technology during the last decade, there is an increased interest for providing speech – based user interfaces. Recently, numerous new commercial services has been made available employing speech and language technology. Speech based interaction is desired and believed to be advantageous in mobile situations and where user's eyes or hands are occupied. Furthermore, speech is very flexible and powerful. In a single utterance a lot of information can be conveyed. For instance an utterance like "I would like to have the cheapest flight to Boston from LA for the next weekend but no early mornings please" conveys information which has to be collected in several tedious steps while using traditional GUI-based solutions.

However, speech user interfaces differ from traditional graphical direct-manipulation user interfaces in a number of ways, which highly affects the designers work in creating usable interfaces. These differences become even clearer in the context of the Ubiquitous Interactor, where services are used from many different user interfaces.

When designing speech user interfaces, it must be taken into account that speech is transient. Once the words have been pronounced, there is no trace of what has been spoken except for maybe in the users' memories. Accordingly, designers must be careful not to overload users' short term memory by avoiding long sequences of menus and commands. It might also be necessary to repeat anything that the user did not understand. Here, a good speech user interface design is crucial for achieving trade off between avoiding the user interface to be slow and maybe tedious to use, and not overloading the memory of the user.

Another major difference between speech user interfaces and graphical user interfaces is that user input is much more difficult to interpret in speech user interfaces. The audio input must be recognized with the help of automatic speech recognition (ASR) which is still computationally expensive. ASR results may be negatively affected by individual speaker characteristics such as age, sex, dialects, speaking style, mood etc. Noise from the environment and bad channel conditions (i.e. microphone characteristics, telephony bandwidth etc.) causes additional problems for the speech recognition. In applications with multiple user interfaces, the interpretation is extra crucial since data need to be transferred between different media.

Another problem in design of speech user interfaces is that speech interaction is still new to most users, and there are no standard interaction elements for speech interaction like the buttons and menus for direct-manipulation user interfaces [17]. This means that most users of a speech user interface are novices, and even if they are expert users of some services there is little certainty that their experience will be applicable on the new service.

SPEECH USER INTERFACES IN UBI

To provide a new type of user interfaces to services in the Ubiquitous Interactor (UBI), default mappings between interaction acts and user interface components of the new user interface needed to be created, along with an interaction engine that could generate them. During this work, it was important to consider best practice from the speech community [20, 22].

In this initial phase, we have chosen to work with a VoiceXML speech only user interface, i.e. speech input and speech output. In future scenarios it would be interesting to investigate multimodal interaction. The existing calendar service was used as sample service.

The speech interaction engine

To incorporate speech user interfaces in the Ubiquitous Interactor we needed to develop an interaction engine that generated speech user interfaces, in our case VoiceXML speech-based user interfaces [6]. VoiceXML is a standardized markup language which shields the developers from low-level implementation details, facilitating rapid application development. VoiceXML also provides support for ASR grammars and language understanding. Furthermore, VoiceXML is widely used for several commercial speech based services.

Since interaction acts are encoded in ISL which is XML compliant, a standard XML parser can be used to obtain a parse tree. In the tree, interaction acts are manifested as leaves and groups of interaction acts are realized as a set of leaves with a parent group node. The speech interaction engine traverses the tree and constructs a corresponding VoiceXML tree with a chunk of VoiceXML for each interaction act. If a customization form is provided the VoiceXML template specified in the customization form is used, otherwise a default template is used.

Technology used

For handling the generated VoiceXML documents, off-the-shelf solutions has been used. We employed a commercial voice portal, SpeechWeb [8] platform from PipeBeach [4] which employs the Nuance automatic speech recognizer (www.nuance.com). For accessing the speech server, OpenPhone, a voice ower IP (VoIP) based solution has been used (www.open323.org).

We have used Apache ECS and a set of dedicated Java classes to generate the VoiceXML code. Since both ISL and VoiceXML is XML compliant, it would have been possible to use XSLT [9] to generate the VoiceXML. However, in this first attempt in applying interaction acts and ISL to speech user interfaces we found it easier to use our existing tools for generating the user interface. In future projects, the user interface generation might benefit from using XSLT.

Default VoiceXML mappings for interaction acts

The initial work on mapping interaction acts to templates for VoiceXML concentrated on creating general templates that could be used in many types of services. However, the nature of speech recognition makes it difficult to create default templates for example for the input interaction act, since unrestricted speech cannot be allowed due to ASR limitations, see below for a more detailed discussion on this issue. Therefore, some templates cannot function without a grammar specified in a customization form.

Below, an example of a default template for an output interaction act is shown:

```
<vxml version="1.0">
 <form id="id1">
 <block name="id3">SICS AB</block>
</form>
</vxml>
```

Listing 1: An output interaction act realized in VoiceXML. See figure 2 for the corresponding ISL code.

This would cause the system to read the value of the `block` element to the user, in this case the line "SICS AB". The value of the `block` element is the value of the `string` element of the original ISL coding of the interaction act. Output interaction acts do not accept any user input. The output default for VoiceXML does not differ much from the defaults for Java Swing and HTML. They all use the simplest way to present information to the user, Java Swing using a JLabel and HTML using plain text.

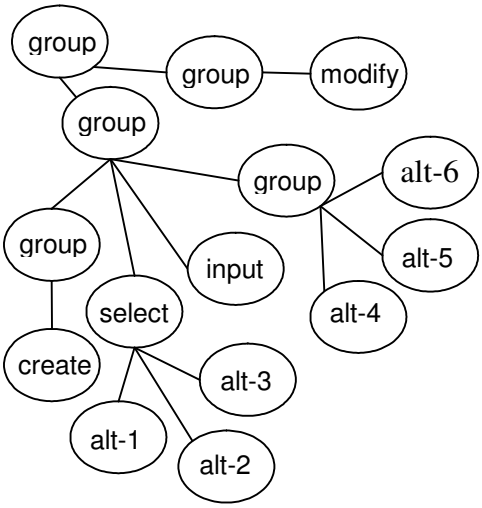

Figure 3: The interaction acts tree for the calendar service.

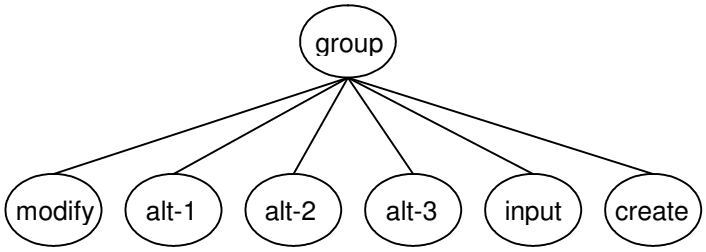

Figure 4: The structure of the implemented speech user interface.

Below, an example of the default template for a select interaction act is shown:

```
<vxml version="1.0">
 <form id="id1">
  <field name="id5">
   <prompt>You may choose previous, or
    next </prompt>
   <grammar root="id5SelectGram">
    <rule id="id5SelectGram" scope="public">
     <one-of>
      <item>Previous</item>
      <item>Next</item>
      <tag>this.id5=this.\$value</tag>
     </one-of>
    </rule>
   </grammar>
   <filled>
    You chose:
    <value expr="id5" />
    .
    <if cond="this.id5=='previous'">
    <var name="Previous" expr="'Previous'" />
    Sending value
    <value expr="id5" />
    <submit next="0" namelist="Previous" />
    <elseif cond="this.id5=='next'" />
    <var name="Next" expr="'Next'" />
    Sending value
    <value expr="id5" />
    <submit next="0" namelist="Next" />
   </filled>
   <noinput>
    Sorry, I couldn't hear you.
    <goto nextitem="id5" />
   </noinput>
   <nomatch>
    <goto nextitem="id5" />
   </nomatch>
  </field>
 </form>
</vxml>
```

Listing 2: A select interaction act realized in VoiceXML. See figure 2 for corresponding ISL code.

This would cause the system to speak the text in the `prompt` element "*You may choose previous or next*". This line is not generated based on the `string` element of the select interaction act, since that in general is kept very short. Instead, the VoiceXML template has its own default prompt. The alternatives and the values to be returned for a chosen alternative are taken directly from the `alternative` elements of the `select` interaction act.

In a speech user interface, all interaction acts that accept user input must be able to handle the case of no input. If the system does not capture any user input for 10 seconds it will go to the `noinput` element and say "*Sorry, I couldn't hear you*" to inform the user that no input was detected. When the system has interpreted the user input as a valid alternative for the selection, it will go to the filled element, confirm the choice for the user, and go to the `if` element where the corresponding response is returned to the interaction engine. The engine interprets the response and forwards it to the service.

The default mapping for the input interaction act has been implemented as a VoiceXML template that has to be complemented with a grammar for interpreting the user input. The grammar is specified as a resource in a customization form. This is a solution that works well for well defined user input, as dates when navigating in the calendar, but is not sufficient for free form input such as notes associated to a meeting in the calendar since grammars for free input becomes very large, and make the recognition slow and not very accurate.

Below, an example of the default template of an input interaction act is shown. Since the grammar for input of dates is too large to show here, a simple illustrating grammar is used.

```
<?xml version="1.0" standalone="yes"?>
```

```
<vxml version="1.0">
 <form id="id1">
  <field name="id2">
   <prompt>Select a week day: </prompt>
   <grammar root="id7SelectDay">
    <rule id="test" scope="public">
     <one-of>
      <item>Monday</item>
      <item>Tuesday</item>
      <item>Wednesday</item>
      <item>Thursday</item>
      <item>Friday</item>
      <item>Saturday</item>
      <item>Sunday</item>
      <tag>this.\$value</tag>
     </one-of>
    </rule>
   </grammar>
   <filled>OK,  <value expr="id2"/>.
    <submit next="0"/>
   </filled>
   <noinput>Sorry, I couldn't hear you.
    <goto nextitem="id2"/>
   </noinput>
   <nomatch>
    <goto nextitem="id2"/>
   </nomatch>
  </field>
 </form>
</vxml>
```

Listing 3: An input interaction act realized in VoiceXML.

The input template has a prompt element to guide the user, and the same handling of no input as the select template. The example grammar above contains a finite set of accepted alternatives, but grammars can also contain for example regular expressions to create an infinite set of accepted input alternatives.

In the current implementation of the VoiceXML interaction engine we have worked both with automatically generated grammars, and hand written grammars for small, restricted vocabularies. For a select interaction act it is very easy to automatically generate a grammar that interprets the input of the specified alternatives. For an input interaction act, there is no obvious way to automatically predict user input, so we allow developers to write grammars in customization forms. This way, a restricted vocabulary for a certain operation, for example input of a date to the calendar service, can easily be created.

Calendar specific mappings

Since the calendar service handles a very specific kind of information, the default mappings were not enough to implement the whole VoiceXML user interface. Calendar specific mappings were needed. For example, we implemented a modify template that handles calendar meeting information. It reads meeting information for a specific date, and offers users the possibility to add, edit, and delete meetings. Due to the lack of space, this template cannot be shown here.

ISSUES RAISED BY SPEECH INTERACTION

As stated above, there are some fundamental differences between graphical user interfaces and speech user interfaces. When applying interaction acts to a new modality by generating speech user interfaces, two aspects turned out to be more central that others. The first is how the structure of the interaction acts tree is translated into the user interface, and the second is how to handle free user input.

The structure of the user interface

The tree of interaction acts that a service exports is a hierarchical structure. To generate a user interface, an interaction engine traverses the tree and builds a corresponding tree of user interface components. This means that the structure of the interaction acts tree is reflected in the user interface, see figure 3 for the interaction acts tree of the calendar service.

In general, an interaction acts tree for a service has more than two levels, a top level, a second level with some interaction acts and some grouping nodes, and a third level with more interaction acts (see figure 3). There is nothing in the Ubiquitous Interactor that prohibits trees with two levels, or trees with more than three levels, but three has been the most common number of levels in our sample services so far.

In graphical user interfaces generated from interaction acts, the structure of the interaction acts tree mostly is reflected in the layout of the user interface, not in the navigation. This means that an interaction act that has a leaf position in the tree, i.e. three or even more steps from the top of the tree, still can be directly accessible to the user. Groups of interaction acts are normally rendered within the same area of the user interface, for example in the same panel, and when needed the group nodes are rendered as panels (in Java Swing) or tables (in HTML) to help control the layout of the user interface. Smaller structural changes in the user interface can be achieved simply by traversing the interaction acts tree in a different order that the default top-down order. This way some interaction acts can be rendered in more salient positions in the user interface to avoid for example excessive scrolling [12].

In speech user interfaces there are no graphical layout, so the structure of the interaction acts tree is mainly reflected in the navigation of the user interface. In the default case, users need to perform a navigation operation for each level that is passed in the tree. To access an interaction act that is a leaf in a three level tree, two navigation operations would need to be performed just to get to the intended interaction act. When we generated a speech user interface for our test service without manipulating the structure, the user interface turned out to be very tedious to interact with. Three, or sometimes even four, menus needed to be traversed to carry out core calendar functions like accessing or adding a meeting.

This is far from the desired natural speech-based interaction, and we looked for a way to generate a user interface with a flatter and more accessible structure from the original interaction acts tree. This is something that often is done in graphical user interfaces too, for example by providing short cuts for menu choices. That way, users

can perform tasks with a single command instead of a series of operations. The immediate solution for the calendar service was to use the <goto> tag in VoiceXML, to allow the user to skip intermediate levels in the tree, see figure 4 for the final structure of the speech user interface. This way, users do not need to perform "unnecessary" operations, and spend a lot of time getting to the functions that they really want to use.

To handle this problem in a more efficient and robust manner, the VoiceXML interaction engine would need a function that flattens the interaction acts tree by default. A starting point for services with a small number of levels in the interaction act tree, could be to let the VoiceXML interaction engine create a totally flat structure containing only the leaves of the interaction acts tree. However, this might not be a good default solution for services with a very large interaction acts tree where the flat structure could contain too many interaction acts. The flattening function could be controlled much in the same way that the presentation of interaction acts can be controlled by customization forms.

Another way to create more natural speech user interfaces from interaction act trees is to use mixed initiative dialogue strategies that allow experienced users to take initiative and input data without the restrictions of detailed menus, while novices get the step by step guidance that menu-based systems offer. If the system detects acceptable user input and manage to interpret it, no detailed menus are presented. If no user input is detected, or the system has problems interpreting it, the system takes control and detailed menus are presented. This way, two structures of the same user interface can be presented. The groups of interaction acts might be natural units for this. However, strategies demand more advances grammars than we had possibilities to develop in this project.

Handling the speech input

The user input in speech user interfaces, and how the input has to be handled by the system, is fundamentally different from handling the user input in graphical user interfaces. Speech user interfaces do have a number of limitations [19] due to the nature of spoken language and due to the limitations of (current) speech technology. Spontaneous speech is ambiguous and exhibits patterns and behaviors learned in human-human communication such as turn-taking, clarification, back channel communication, anthropomorphism. The transient nature of speech in combination with the limitations of human cognition restricts the amount of information which can be presented to the users. Furthermore, in the users' environment there are multiple factors which may negatively influence the speech interaction such as noise, multiple voices, interruptions etc.

The above limitations are well-known by the speech technology community and addressing the challenge of handling spontaneous speech is one of the major challenges for the speech community. On the other hand, even if the employed speech technology would be "perfect", the user input still has to be interpreted by using natural language understanding.

Albeit it is evident that designing speech user interfaces is much more than speech recognition, limitations of current speech recognizers remains still the major concern while designing speech user interfaces. In spite of the advances in large vocabulary automatic speech recognition, the most of the commercially available speech services rely on grammar based speech recognition. The size, quality, and coverage of the grammars highly determine the results of the recognition.

Furthermore, the handling of the free user input still remains a major limitation for the speech based services. Services that has to cope with free input, for example note taking services, can sometimes resort to the audio recording of the user input. The recorded audio can be replayed later. However, in the context of services with multiple user interfaces, free spoken user input poses the additional challenge of making the spoken information available from other types of user interfaces. The speech data has to be recognized and interpreted or if user input was recorded then it must be presented as audio in other types of user interfaces too.

In the calendar service we have not allowed free input so far, even though several functions would benefit from it. A graphical calendar allows users to freely compose headings of meetings (for example "lunch with Anna", "annual board meeting ", or "grandma's birthday party"). For test purposes, we have chosen to provide classes of meetings in the speech user interface such as: *lunch*, *dinner*, and *meeting*. The same solution was applied for the location field, users could choose between a number of pre-determined places. In both these cases, hand written grammars were placed in the customization form. The note taking functionality that is available in the Java Swing user interface and the web user interface of the calendar was not implemented in the VoiceXML user interface due to time constraints.

PRELIMINARY USER TESTING

We have performed a small and informal user study to find out both to find out how well the user interfaces generated by UBI work, and what users think about the concept of having multiple user interfaces to services and thus be able to use them from many different devices.

Eight subjects, five male and three female, all between the age of 20 and 30, participated in the study. They worked in pairs and used both a Java Swing user interface and a speech user interface. The participants all used computers daily, and five of them had previously used a service with some kind of speech user interface. Six of them used some kind of tool to plan their time, for example a calendar (in the computer, cell phone, or a traditional paper format) or a paper notebook. Two groups started with the speech user interface, and continued with the graphical user interface and two groups used the user interfaces in the opposite

order. All groups got a one-page instruction with the available commands for the speech user interface to the calendar, before starting to use it.

Each group performed two sets of tasks. The first set consisted in adding a meeting on the current date, navigate forward until they found a meeting in the calendar, change the location of that meeting, go to a given date about a month ahead in time and delete the meeting on that day. The second set consisted in changing the location of the meeting the participants added in the first set of tasks, navigate forward in the calendar until they found a meeting, delete that meeting, go to a given date a few weeks ahead in time and add a meeting at that date. The first set was always performed first, regardless of which user interface the participants started with. The task sets were constructed so that the participants when performing the tasks of the second set would come back to meetings that they had seen and manipulated in the first set. This way it would be clear to them that they were using the same calendar service from different user interfaces, not one service with a speech user interface and another one with a graphical user interface.

The participants had no problem completing the two sets of tasks in either of the user interfaces. The instructions to the speech user interface seemed to be sufficient even for the participants that had never used a speech user interface before. One group said that the speech user interface had worked better than they expected. Not all of the participants realized that they accessed the same meetings from different user interfaces. However, when asked they all stated that they were sure that all information entered from one user interface could be accessed from another. Several of the responded that "otherwise something is wrong with the implementation because that is the whole point of this system".

The concept of multiple user interfaces to services was well received by the participants. They liked the possibility to choose different user interfaces in various situations and suggested that user groups as doctors, taxi drivers, people with RSI or other disabilities, and travelers could benefit from this. However, in the case of the calendar most of them reported that their schedule was not very busy so they did not feel that they needed multiple user interfaces to access their calendar.

FUTURE WORK

We will implement support in the speech interaction engine for manipulating the structure of the interaction act tree. A simple default mechanism could be to extract the leaves from the tree and thus create a flat structure. Obviously, this is not suitable for all kinds of services so it must be possible to block the flattening function as well as tailor it.

The issue of user input needs to be treated from two different angles. First, we need to investigate how to incorporate known solutions from the speech interaction area, such as strategies, into UBI. This way we can find general solutions that benefit speech user interfaces in UBI. Secondly, we need to look at service specific solutions, as we used classes of meetings in the calendar service. This way we can solve service specific problems.

Further user studies are planned, to evaluate the quality and usability of the user interfaces generated with the Ubiquitous Interactor. We will also compare them to user interfaces created with other systems for services with multiple user interfaces, for example web user interfaces that in general look the same for all devices and user interfaces that are created from scratch for each device.

CONCLUSIONS

We have shown that it is possible to generate speech user interfaces based on descriptions of user-service interaction created with interaction acts. Default VoiceXML templates for all types of interaction acts have been created, as well as a working user interface to a sample calendar service. Preliminary user studies conducted with a Java Swing user interface and a VoiceXML user interface to the calendar service show that users have no problem working with user interfaces generated from interaction acts.

The application of interaction acts to VoiceXML user interfaces has identified two important issues that need extra attention. The first is the structure of the user interface. The structure of the interaction acts tree in graphical user interfaces is manifested mainly in layout while in speech user interfaces it is manifested in navigation. This means that measures need to be taken for speech user interfaces to avoid unnecessary navigation operations. The second is the handling of user input. It is difficult to allow free user input for speech user interfaces, and we have tried several alternative solutions.

We did not see any need for modification of the interaction acts to accommodate to the new type of user interface. However, we identified the need of an automatic flattening function in the VoiceXML interaction engine to avoid unnecessary hierarchical user interfaces. The flattening function need to be tailorable, much in the same way as the customization forms tailor the generation of user interfaces.

ACKNOWLEDGMENTS

The authors would like to thank members of the Userware lab and the Interaction lab at SICS for valuable help and advice during this work. We would also like to thank the participants in the user study for giving up some of their time to help us. Part of this research was carried out at the CTT, Centre for Speech Technology, a competence center at KTH.

REFERENCES

1. Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J. and Zukowski, D., Challenges: An Application Model for Pervasive Computing. In Proceedings of 7th International Conference on Mobile Computing and Networking, (2000).

2. Bylund, M. Personal Service Environments - Openness and User Control in User-Service Interaction. Licentiate thesis, Department of Information

Technology, Uppsala University, 2001.

3. Bylund, M. and Espinoza, F., sView - Personal Service Interaction. In Proceedings of 5th International Conference on The Practical Applications of Intelligent Agents and Multi-Agent Technology, (2000).

4. Currently part of the HP OpenCall suite http://h20208.www2.hp.com/opencall/index.jsp.

5. Esler, M., Hightower, J., Anderson, T. and Borriello, G., Next Century Challenges: Data-Centric Networking for Invisible Computing. The Portolano Project at the University of Washington. In Proceedings of The Fifth ACM International Conference on Mobile Computing and Networking, MobiCom 1999, (1999).

6. FORUM, V. Voice eXtensible Markup Language version 1.0, 2000.

7. Gajos, K. and Weld, D.S., SUPPLE: Automatically Generating User Interfaces. In Proceedings of 2004 International Conference on Intelligent User Interfaces, (2004), 93-100.

8. http://h20208.www2.hp.com/opencall/products/media/speechweb/index.jsp.

9. Kay, M. XSL Transformations (XSLT) Version 2.0, W3C Working Draft 11 February 2005, World Wide Web Consortium, 2005.

10. Myers, B.A., Hudson, S.E. and Pausch, R. Past, Present and Future of User Interface Software Tools. ACM Transactions on Computer-Human Interaction, 7 (1). 3-28.

11. Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R. and Pignol, M., Generating Remote Control Interfaces for Complex Appliances. In Proceedings of 15th Annual ACM Symposium on User Interface Software and Technology, (Paris, France, 2002), 161-170.

12. Nylander, S., Bylund, M. and Boman, M. Mobile Access to Real-Time Information - The case of Autonomous Stock Brokering. Personal and Ubiquitous Computing, 8 (1). 42-46.

13. Nylander, S., Bylund, M. and Waern, A., The Ubiquitous Interactor - Device Independent Access to Mobile Services. In Proceedings of Computer Aided Design of User Interfaces, (Funchal, Portugal, 2004), 274-287.

14. Nylander, S., Bylund, M. and Waern, A. Ubiquitous Service Access through Adapted User Interfaces on Multiple Devices. Personal and Ubiquitous Computing, 9.

15. Olsen, D.J. MIKE: The Menu Interaction Kontrol Environment. ACM Transactions on Graphics, 5 (4). 318-344.

16. Olsen, D.J., Jefferies, S., Nielsen, T., Moyes, W. and Fredrickson, P., Cross-modal Interaction using XWeb. In Proceedings of Symposium on User Interface Software and Technology, UIST 2000, (2000), 191-200.

17. Rosenfeld, R., Olsen, D.J. and Rudnicky, A. Universal Speech Interfaces. Interactions, 8 (6). 34-44.

18. Shneiderman, B. Leonardo's Laptop. MIT Press, 2002.

19. Suhm, B., Improving Speech Interface Design by Understanding Limitations of Speech Interfaces. In Proceedings of AVOIS 2003, (San Jose, CA, 2003).

20. Suhm, B., Towards Best Practices for Speech User Interface Design. In Proceedings of EUROSPEECH 2003, (Geneva, Switzerland, 2003), 2217-2220.

21. Wiecha, C., Bennett, W., Boies, S., Gould, J. and Greene, S. ITS: a Tool for Rapidly Developing Interactive Applications. ACM Transactions on Information Systems, 8 (3). 204-236.

22. Yankelovich, N. and Lai, J., Designing Speech User Interfaces. In Proceedings of Human Factors in Computing Systems, (1999), 124-125.