Audio Navigation Patterns

Dirk Schnelle, Fernando Lyardet and Tao Wei
Telecooperation Group
Darmstadt University of Technology
Hochschulstrasse 10
D-64283 Darmstadt, Germany
{dirk,fernando}@tk.informatik.tu-darmstadt.de
taowei@gmx.net
phone: +49 (6151) 16-5411
fax: +49 (6151) 16-3052

Abstract

Nowadays, the design of efficient speech interfaces is considered to be more an art than an engineering science. Several guidelines exist, but are more or less hints about what should be avoided. We introduce a first set of patterns integrating many guidelines and solution approaches developed during the past 15 years. Our goal is to document and share this knowledge with new designers as well as providing a language to talk about *Voice User Interface* (VUI) designs

1 Introduction

Speech was promised to be a revolutionary means of interacting with a computer for years. Voice enabled services are offered via the telephone, enhancements to GUI applications to make them multimodal or standalone to aid visually impaired people. However, at the heart of making these concepts a reality is an efficient VUI.

The design of an efficient speech interface is still a challenging task. One reason for this is that language is deeply ingrained in human behavior. As a consequence, the expectations regarding such an interface became very high. They lead to the vision of a patient listening, competent and informed homunculus, with which users could talk like they talk to a human. These

expectations have not been reached, yet. Speech technology has improved a lot but the issue of a speech application that meets the user's needs and is easy to use is still unsolved.

The central goal of VUI design is supporting the user in navigating through the options, commands, and information available in order to accomplish a task. We assume that the information is stored in a tree-like structure. The audio navigation document structures borrow from hypertext the notion of enabling users to access information in an intuitive and associative way. However, hypertext systems rely on graphical user interfaces to navigate through a two-dimensional space.

Unfortunately, accessing information through navigation is more complex in the audio domain. In aural interfaces, concepts such as menu selection, control widgets, and link anchors must be performed and revealed differently than in visual interfaces. While speech is a powerful means of communication, its transient nature severely limits the amount of information that can be conveyed to the user. The eye is active whereas the ear is passive, i.e. the ear cannot browse a set of recordings the way the eye can scan a screen of text and figures. It has to wait until the information is available, and once received, it is not there anymore. Furthermore, the voice as a means for selection brings forth the set of ambiguities inherent to the natural language. Finally, other human factors such as user's short term memory (STM) capabilities and listening skills strongly influence the efficiency of a speech interface.

Researchers began to investigate how to face these problems and started to write guidelines about VUI design. A centralized, clear and structured way to provide this knowledge is still missing in the audio only world.

In the context of an on-going audio navigation project, we have been investigating a set of patterns that repeatedly appear in real world applications as well as in our prototypes. These patterns also comprise several known style guides [3] and design guidelines such as those described in Cohen et. al. [2] on VUI.

2 Audio Navigation Patterns

In this section we present some patterns that are helpful in the domain of audio navigation. The patterns can be categorized according their main purpose. An overview is given in figure 1, and connections between them are shown in figure 2.

Our focus is on ubiquitous computing environments, where users can walk around and retrieve information about items of the real world. Artifacts of the real world are starting points for navigating to the desired information.

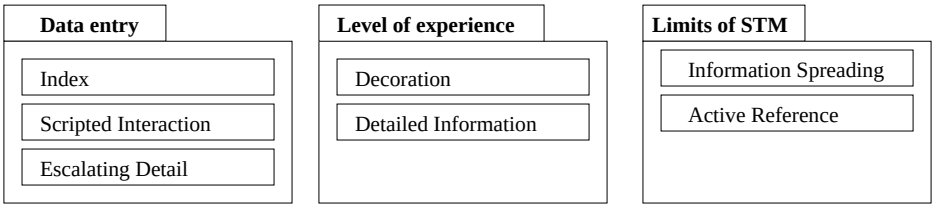


Figure 1: Categorization of Audio Navigation Patterns

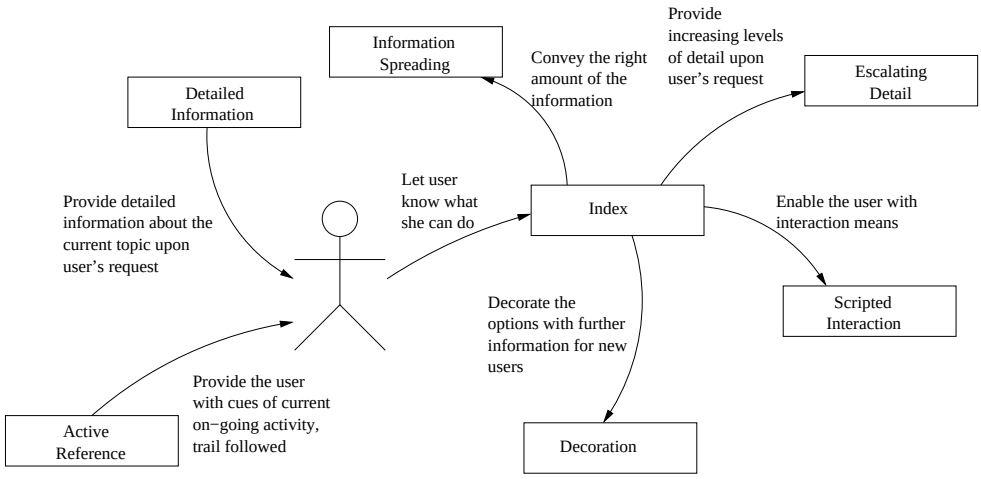


Figure 2: Relation of Audio Navigation Patterns

The user makes an *utterance*, a special command or a complete sentence in natural language, to start browsing. This is evaluated by the *recognition engine* and handed over to the *audio browser* to generate a response in audio. The response consists of natural language, generated by a *text-to-speech engine* (TTS engine) and some *auditory icons*, prerecorded sounds used as a means for structuring.

Although this scenario is different from the scenarios where we typically find audio based user interfaces, such as the telephone, we think that many of the problems we face are common to all kinds of VUIs. Hence, the task of navigating to a desired service in a telephony application, has a similar structure as browsing for a certain *audio document*. Most of the patterns are useful in a audio-only user interface, although some of them, for example the patterns for data entry, can also be used in multimodal applications.

An application designed for audio only input and output faces several challenges before it can be named easy to use. Some of them are named

in [5]. As pointed out in [7] these can be classified into challenges inherent to audio and technical problems.

- Problems inherent to audio
  1. Speech is transient. Once you hear it or say it, it is gone.
  2. Speech is invisible. The user does not see the interface.
  3. Speech is asymmetric. Speaking is faster than typing, but listening is slower than reading.
- Technical problems
  1. Speech synthesis quality.
  2. Speech recognition performance. Speech recognition is still error prone, increased by out of vocabulary utterances and background noise.
  3. Recognition: Flexibility vs. Accuracy.

These are forces which are met in all our patterns. An example implementation of each pattern is given. The scenario of our examples is at a car inspection, where a worker processes a list of work items. We use VoiceXML for the example code and a specification of the tags can be found in [8].

Some of the pattern descriptions feature a section about known uses. In general, this is a real telephony application which can be called Some of the phone numbers may incur calling charges. It is also possible that they are not accessible from outside of Germany.

2.1 Data entry

Since speech is invisible, users may not know what to answer or do not know which words they can say. Even if explained by a prompt, they may have forgotten about it, due to to the transient nature of speech, and may utter a command which is not in the grammar. This in combination with recognition errors for not successfully recognized valid commands can make the application unusable.

2.1.1 Index

Intent Present a menu of options to the user.

Context In order to use an application, users must be able to recognize the available functionality in order to operate the software. In VUI applications, it is particularly difficult to let users become aware of what the available options are and how to perform a selection. User selections can be performed in different modalities (voice or keystroke), and given a chosen modality, how the selection is actually performed (voice commands or keystroke sequences) must also be defined. Finally, the imperfect performance of speech recognizers forces the introduction of further complexity in the application in order to recover from errors while sustaining user's confidence and interest.

Problem How should a VUI reveal the available functionality to the user?

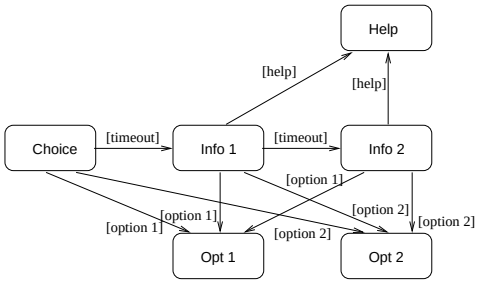
Forces

- Novices require more detailed information about each option in order to decide the most convenient.
- Experts do not want to be bothered with the same long explanation they already know by heart, nor do they want to hear all the options every time they access the system.
- Voice activation is simpler
- Button selectable options are faster

Solution There are two aspects involved in the solution that need to be addressed: letting the user learn the system's vocabulary, and providing different means to perform the selection of an option. To introduce the system's vocabulary there are three common techniques that can be combined:

1. Explaining the options at every point in the discourse (*a directive prompt*), guiding the user towards issuing a valid command.
2. Give directive prompts only if the user fails to issue a command within a few seconds (*a timed prompt*).
3. Provide context sensitive *help on demand*. Users may become confused as to what state the conversation is and what the system would accept as a valid command. In this way, users are given a chance to re-establish the context, recall the available options, and ultimately find a way out.

Structure This is illustrated in the following figure. The user is requested for an utterance in node *Choice*. If she does not make an utterance within a certain time, more information is given in *Info 1*. If she again does not know what to say, she gets more info in *Info 2*. At any time, she can get more help in *Help* by the corresponding command or select the desired option to continue with *Option 1* or *Option 2*



On the other hand, there are a number of approaches to enable more opportunities for the user to perform the selection of an option using voice and buttons. Again, we can divide the approaches in two:

- *Transient*: Options can be unambiguously selected while they are being conveyed to the user, until the next option is named. Usually activation is enabled by stating *Yes* or pressing the equivalent button on the phone.

- *Persistent*: Options are indexed and assigned an ID that is always the same for the same list of options and the current task. Options become active immediately after the user access the current list, thus enabling experienced users to quickly navigate to a known option. Furthermore, keeping the coding of options persistent enables users to memorize sequences for common, repetitive tasks.

Finally, since audio interfaces are mostly linear, it it important to provide users with the possibility of skipping (*barge-in*) options. Such skipping could be again triggered by voice (*next* or *previous* voice commands) or by pressing a number for other options.

Consequences

+ Users are provided with a wealth of choices to make a selection, making it easier to use a new systems using conventions and assistance to effectively completing a required task.

- Enabling the selection of options using buttons typically limits the number of choices to about 12, where 10 (0..9) can be used for item selection and the remaining 2 (typically "#" and "*") for control purposes.

Sample code At our car inspection, the worker can get more help on how to behave after a timeout. This is an extension to the example given at SCRIPTED INTERACTION to provide more guidance.

```
<field name="detailChoice">
    ...
    <prompt>
        If you are not familiar with the procedures of this
        station, you can get more help on:
        <mark name="oilChange" />
        <audio src="item.raw"/> Oil change <break />
        <mark name="changeTyres" />
        <audio src="item.raw"/> Change tyres
        <mark name="checkNoise" />
        <audio src="item.raw"/> Check noise from exhaust
    </prompt>

    <timeout count="1">
        Say next, when you want to proceed.
    </timeout>

    <timeout count="2">
        You can also say repeat, to listen again.
    </timeout>
    ...
    <filled>
        ...
        <goto expr="'#detail' + detailChoice$.markname" />
        ...
    </filled>
</field>
```

2.1.2 Scripted Interaction

Intent Guide the selection of an option with a limited vocabulary.

Context It is common use to map each list item to a certain keyword. Dynamically generated lists thus enforce a change in the active grammar.

Problem Selection from a list is a problem for users of speech based applications, especially novices. Menu driven applications, which which are used in most telephony applications, have the same problem, since menu selection can be reduced to list selection. Novices need more guidance to choose the desired option or service in menu driven applications. They do not know the content of the list, which is identical to the keyword. If the list content varies, this is also true for all users, so the problem turns to handling dynamic lists with a limited vocabulary.

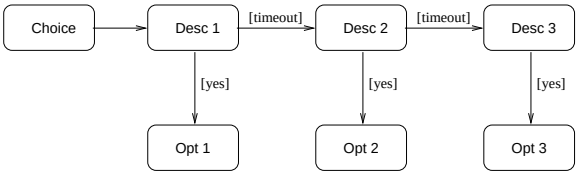
Forces

- Selection of items by name enables fast selection.
- List items may be not acoustically different.
- Items of the list are unknown to the user.
- Utterance of the user must match the grammar.
- Not all speech recognizers allow a dynamic change of the grammar.

Solution Write the items as a script which leads the user through the selection process, as proposed in [4]. Read this script to to the user. Expect a single command to activate the current alternative.

Add the possibility to navigate forwards and backwards within the list, to the first or last section or repeat the current section by corresponding voice commands to allow for a faster access to the desired list item.

Structure This following figure illustrates the structure of this list. After the choice introduction in node *Choice* has been read, the system continues with naming the first option in node *Desc 1*. The user can select this option *Opt 1* by a confirmation. If she says nothing, the next option in node *Desc 2* is read after a short timeout.



Note that *Choice*, *Desc 1*, and *Desc 2* will appear as a single node in the graph. They form their own navigation space.

Consequence

+ Guidance of the user through the selection process.

+ Processing of any list with a fixed and small vocabulary.

- Navigation to an item may require many interactions from the user, especially for larger lists.

Known Uses The Deutsche Bahn AG offer in their travelling service the possibility to select an option, such as an earlier or a later train, after the schedule has been read. The user can select the command by *yes* after the option has been named. The application can be called at +49 (800) 150 70 90.

Borussia Dortmund offers a ticket reservation system via telephone. While the matches are read to the user, she can select the named match by the voice command *stop*. The application can be called at +49 (1805) 30 9000.

Sample code At our car inspection, the worker can get more information about the items of the inspection process. The items are read out and introduced by an auditory icon *item.raw*. During the break before the next item starts, the worker can select the item by the utterance *Yes*.

```
<field name="detailChoice">
    ...
    <prompt>
        If you are not familiar with the procedures of this
        station, you can get more help on:
        <mark name="oilChange" />
        <audio src="item.raw"/> Oil change <break />
        <mark name="changeTyres" />
        <audio src="item.raw"/> Change tyres
        <mark name="checkNoise" />
        <audio src="item.raw"/> Check noise from exhaust
    </prompt>
    ...
    <filled>
        <goto expr="'#detail' + detailChoice$.markname" />
    </filled>
</field>
```

2.1.3 Escalating Detail

Intent Provide an error recovery strategy when the recognizer does not return a recognition hypothesis.

Context The performance of speech recognition is still a problem. The most common results in case of errors are *reject*, which means that the utterance could not be mapped successfully to the grammar, and *no-speech timeout*, which means that no speech was detected at all.

Problem How to aid the user to recover from an error?

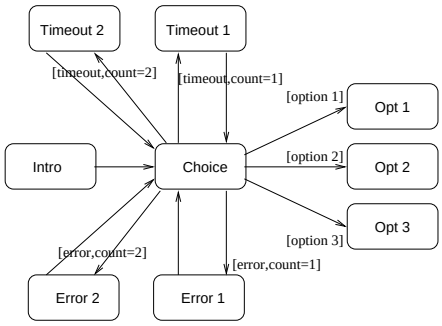
Forces

- In case of a timeout, it is likely that users did not listen carefully and do not know what utterances they may perform. They need more guidance in entering the required data.
- In case of an error, experienced users are forced to listen to long explanations, as a consequence of the one-dimensional medium, even if they just need another try. They are interested in a faster system response.
- Users do not known what to say and need more detailed explanations. This hinders them from using the application..
- Different types of errors, *reject* and *no-speech timeout*, need different strategies to solve them.
- Low recognition performance and background noise result in a misinterpreted type of error.

Solution Write handlers for different types of errors. Use a counter for the occurrences, levels, of the affected type in a row. Write prompts which increase the amount of help provided at each level.

The higher levels may also offer alternative approaches, e.g. a form-based data entry for a date versus free-form input, to enter the required data. Within these repetitions the initial prompt can be reused to provide information about the input possibilities.

Structure The structure is shown in the following figure. The choice introduction is in node *Intro* and the actual presentation of the options in *Choice*. The error document is chosen depending on the error count and the type of the error.



Consequences

+ The error recovery strategy matches the type of the error and adapts to the dialog state. The application gives only so much help as it is needed for the current dialog state.

+ The user is guided and does not need to listen to lengthy and general explanations.

- Difficult to find the right amount of help at each level.

- Not suitable for open-ended prompts, i.e., *How may I help you?*.

Related patterns DECORATION can be used to reuse the initial prompt.

Sample code At our car inspection, the worker can get more information about the items of the inspection process while the items, introduced by the auditory icon *item.raw* are read and during the break before the next item starts, by the simple utterance *Yes*.

```
<field name="answer">
    <nomatch count="1">
        Please say next to proceed.
    </nomatch>

    <nomatch count="2">
        I could not understand you.
        Please say next to proceed, or help to get a list
        of valid commands.
```

```
    </nomatch>

    <noinput count="1">
        Please say something.
    </noinput>

    <noinput count="2">
        <reprompt/>
    </noinput>

    <prompt>
        When you are ready, please enter the car
        and sit in the drivers seat.
        Say next when you are there.
    </prompt>
    ...
    <filled>
        <if cond="answer=='next'">
            <goto next="#nextDialog" />
        </if>
    </filled>
</field>
```

2.2 Different level of experience

An application that has to deliver information with different levels of detail has to make a trade-off between a more experienced user (*expert*) and a novice user (*novice*) with only little background information.

Since speech is only one-dimensional, the expert would have to listen to all the information, which she already knows, whereas novices may not get enough information to use the application.

2.2.1 Decoration

Intent Present the same choices to the user with different background information.

Context Selection of an option is a common task in menu based telephony applications. While the commands are unknown to novice users, the expert knows what to say and does not need any explanation.

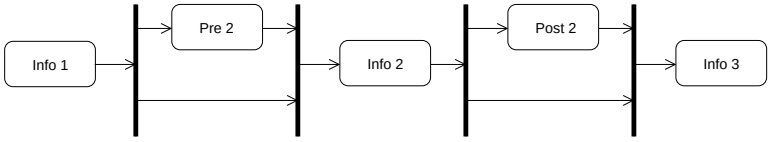
Problem How to present the same choices to users with different background?

Forces

- The same options must be presented differently to users with different background knowledge or experience.
- The users to be supported and their domain knowledge may be unknown and can change over time.
- Not every prompt is suitable as a candidate to be decorated.

Solution Use a profile to classify the expertise of the user with the system. This can be achieved by an identification mechanism as it is used i.e. in a banking application or by observing the speed of the user's interaction with the system and making guesses about the user's experience. The solution is built reusing the design concept of the GOF's Decorator Pattern [1]. The intent of a decorator is to provide a flexible alternative to subclassing for extending functionality. In this way, VUI can provide a simple and extensible adaptation mechanism by transparently adding or *decorating* existing options with further information. Transparency refers to the actual definition of the option remaining unchanged. The decoration procedure is very simple and can only happen either before or after the given option is conveyed to the user, by inserting or removing explanatory information according to the user's profile.

Structure The following figure illustrates the structure. *Info 2* is the node to be decorated. Decoration can be done by either *Pre 2* before or after this node by *Post 2*. using both, pre- and post-decoration for one node is also possible.



Consequences

+ Users are provided with different description of the options available.
- It might be difficult or impossible to obtain the user's profile.

- Decoration cannot be used until the user's profile is obtained.
- Guessing maybe wrong, if the user knows only parts of the application very well.

Related patterns The GOF DECORATOR pattern. ESCALATING DETAIL can be used to recover from wrong guesses. DETAILED INFORMATION can be used if the profile cannot be determined.

Sample code The following example is again taken from a car inspection. The list of items is dynamically generated from a database.

This list is decorated when it is read to the worker with a short introduction to the list:

```
<var name="decoration"
     value="This car needs the following repairs"/>
...
<prompt>
    <value expr="decoration" />
    <audio src="item.raw"/>Oil change
    <audio src="item.raw"/>Change tyres
    <audio src="item.raw"/>Check noise from exhaust
</prompt>
```

2.2.2 Detailed Information

Intent Deliver information with a different level of detail.

Context An application that has to deliver information with different levels of detail and that has to address novices and experts has to deliver all the needed information with respect to the user's background information. Solutions for this problem are solved in written documents by a footnote, or in the hypermedia world by a link. They enable the user to retrieve more detailed information about a certain passage on demand. Thus, users are able to control the amount of information being delivered on their own. In addition, this does not hinder the normal reading process.

Problem How to control the amount of information users get?

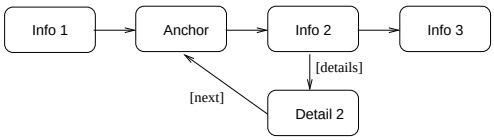
Forces

- Users with different background knowledge require different levels of details.
- Expert users do not want to be bothered with the same long explanation they already know by heart, nor they want to.
- The users to be supported and their domain knowledge may be unknown and can change over time.

Solution The mechanism of a footnote can be copied to audio information delivery. Use an auditory icon or a speaker change to mark the passage where users can follow a link to more detailed information. Expect a single command to activate the link and define an anchor point where the user can continue after listening to the detailed information, e.g., the start of the passage or an earlier point to get back into context. Expect a command to continue at this anchor point.

The change of context can be introduced by another auditory icon. A repetition of the whole document is not needed.

Structure This is illustrated in the following figure. After *Info 1* has been delivered to the user, the system passes the anchor point *Anchor* and continue with *Info 2*. The user may utter a command to get more detailed information in *Detail 2* and continue with another command at the anchor point. If she decides not to request more details, she continues with *Info 3*.



Consequences

+ Users get enough information to work with an application, independent of their background knowledge.
+ No profile needed.
+ Novice users can follow a link to get more information to enhance their background information and can easily get back into the main context.
+ The added auditory icon has less impact on the amount of information to be delivered to the user than any explanation given at this point.

- Too many occurrences in a row for only short paragraphs will overload the memory capacities of novice users.
- This pattern cannot be applied if the information cannot be grouped or sorted in a way that is not obvious to the user.
- An additional, reserved, word is needed.

Related patterns If the user's profile is known, DECORATION can be used to provide additional information.

Sample code The following example extends the list of items of the car inspection to get a more detailed description of the item.

```
<field name="detailChoice">
    ...
    <prompt>
        This car needs the following repairs:
        <mark name="oilChange" />
        <audio src="item.raw"/> oil change <break />
        <mark name="changeTyres" />
        <audio src="item.raw"/> change tyres <break />
        <mark name="checkNoise" />
        <audio src="item.raw"/> check noise from exhaust
           <break />
    </prompt>
    ...

    <filled>
        <goto expr="'#detail' + detailChoice$.markname" />
    </filled>
</field>
```

The worker can ask for a more detailed information by uttering *details* from the start of the marker, while the item is being delivered by the Text-to-speech (TTS) engine and throughout the pause after the item. The detailed information for the the item *change tyres* may look like this:

```
<prompt>

<audio src="section.raw"/> Change tyres

<audio src="paragraph.raw"/>
This car needs tyres of size 185/55 R15 S82.
```

```
</prompt>
```

The worker can resume with the command *next* after the detailed information has been read to get back into context.

2.3 Limits of the short term memory

An application that has to deliver a large amount of information faces the problem that users are not able to keep all the information, or extract the information in which they are actually interested. The main reasons for this is the limitation of the short term memory (STM).

2.3.1 Information Spreading

Intent Divide the amount of information being delivered to the user at once into manageable pieces.

Context Treating all the information to be delivered as a single logical entity conflicts with the limits of STM and has the drawback that users cannot navigate to the information in which they are actually interested, and have to listen to all the information that is given.

Problem How to handle large amounts of information?

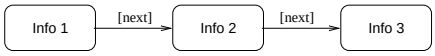
Forces

- A large amount of information has to be delivered.
- The short term memory capabilities of users are strongly limited.
- It might be impossible to find logical entities within the piece of information to spread.

Solution Group the information to be delivered into several sub nodes. Groups can consist of information items that meet certain criteria or are limited by their number. In the latter case they have to be sorted in an order that is obvious to the user.

A command or a timeout after each information subnode can be used to proceed with the next subnode. We add commands to allow for back and forth navigation, and to handle the case that users realize that the information of interest cannot be in the current subnode, but may appear in a previous subnode or a subnode to come next.

Structure This is illustrated in the following figure. The information is spread over the node *Info 1*, *Info 2*, and *Info 3*. The user can proceed by a utterance.



Note that this structure forms it's own navigation space.

Consequences

+ The information that is delivered to the user is split up into pieces that the STM can handle.

+ Users gain control about the information being delivered at once. They may also navigate within this information. This allows for easier concentration on information of interest.

+ From application side, the spread information can still be treated as a single logical entity.

- Each information node must form a logical entity and has to be set up manually.

Known Uses The FAZ Electronic Media GmbH reads columns, e.g. sports, to the user. The headlines and the corresponding contents are read one by one. The user has the possibility to navigate in these headlines using voice commands. The application can be called at +49 (1805) 32 91 000.

Sample code If the worker at our car inspection has to check the radio, the information about the use of the radio is spread over several documents. She can navigate through them by simple browsing commands *next* and *go back*. Note that only the first document contains the title.

```
<form name="start">
    <field name="start">
        ...
        <prompt>
            <audio src="section.raw"/> Radio

            <audio src="paragraph.raw" />
            The radio is located under the clock and the
            volume knob is green. To turn on the radio, turn
            the volume knob clockwise.
```

```
        </prompt>
        ...
        <filled>
            <if cond="start == 'next'">
                <goto next="#tuning" />
            <else />
                <reprompt />
            </if>
        </filled>
    </field>
</form>

<form name="tuning">
    <field name="tune">
        ...
        <prompt>
            <audio src="paragraph.raw"/>
            Turning the volume knob further clockwise
            increases the volume. To tune the radio, use
            the two buttons marked with arrows which are
            located on top of the volume button.
        </prompt>
        ...
        <filled>
            <if cond="start == 'next'">
                <goto next="#frequency" />
            <else />
                <reprompt />
            </if>
        </filled>
    </field>
</form>
```

2.3.2 Active Reference

Intent Help the user stay oriented.

Context Designing a voice interface requires careful planning of what functionality will be exposed to the user. The design of the navigational space is particularly complex, as it tends to be modeled spatially, which makes voice navigation difficult to map [6]. As the number of options and amount of information available increases, it relies more and more on the user's at-

tention and memory. It becomes complex to identify successive actions as a part of the same task, remembering the options available, and deciding the path to follow. Furthermore, limitations of speech recognition performance introduce an indefinite number of further steps for error recovery and action confirmation. Since speech is invisible, after confirming, repeating options, selecting, listening instructions, correcting selections, the user may face the *lost in space* problem.

Problem How to help the user not to become lost in the audio space?

Forces

- Lack of permanent marks or cues as in visual interfaces, due to the transient nature of audio.
- STM is limited and cannot be constantly exercised to operate the application.

Solution The problem of orientation is similar to the hypertext domain, and it is centered within 4 cardinal questions: *Where am I?*, *How did I get here?*, *Where can I go from here?* and *Why am I here?* (*What am I doing?*). This means the solution must address all these aspects. However, the linearity of the aural environment forces us to address each aspect separately. Furthermore, some of the solution proposals are split into transient and permanent aspects.

- **Where am I?**: Name the place at the moment of entry and allow the user being reminded under request.
- **How did I get here?**: Name a path of arrival upon request of the user.
- **Where can I go from Here?**: Usually implemented as an INDEX (see Section 2.1.1), also should be available on demand.
- **Why am I here?** (What am I doing?): Give a sense of unity to the steps a user is performing. Create auditory indicators serving as a binding for related steps to the current task. In that way, users can distinguish whether current place belongs to the ongoing activity of if the latest action implied a change of activity.

Consequences

+ Users are provided with different cues that help them improve their awareness of their current status

- Excessive cues pushed to the user increases overhead, and user resistance

Related patterns Escalating Detail can provide a means to provide increasing location help to the user. Scripted Interaction is used to navigate to the current path item.

Sample code As an examle we modify the example given for Scripted Interaction to play back a prerecorded *radio* sound, before or while the information is being delivered.

```
<form name="start">
    <field name="start">
        ...
        <prompt>
            <audio src="radio.raw" />
            <audio src="section.raw"/> Radio

            <audio src="paragraph.raw" />
            The radio is located under the clock and the
            volume knob is green. To turn on the radio, turn
            the volume knob clockwise.
        </prompt>
        ...
        <filled>
            ...
        </filled>
    </field>
</form>
```

Note Unfortunately VoiceXML is not capable to play audio in background. As a workaround, the whole audio can be prerecorded and mixed manually. Since we wanted to show the mixing, we decided to add the text as a prompt.

3 Summary

In this paper we have introduced a set of patterns that repeatedly appear in voice based applications. These patterns comprise several known design guidelines for VUI design. The complexity for designing VUI applications has its roots in the inherent transience and invisibility of the medium, the user's memory capabilities, and the varying performance of currently available speech recognition technology. These three factors introduce a particular set of problems and requirements to the application that must be addressed, and that we have documented in this paper.

The design patterns we presented in this work aim to help the designer of a VUI understand the nature of the problems, and successfully analyse and solve these issues to provide a successful voice interface.

Scripted Interaction, Escalating Detail, and Index offer guidance for data entry, while Escalating Detail focuses on recovering from a recognition error.

Decoration and Detailed Information address the problem of supporting users with different background information and experience.

Information Spreading and Active reference tackle the problem of the Short Term Memory limitations. Active Reference helps the user stay oriented by providing different cues that help improve the awareness of the current status, and Information spreading provides the means to avoid overloading the cognitive capabilities of the user.

An overview, how these patterns can be combined and how they behave in action is illustrated in figure 3. This is a very dynamic process, which we show in four usage scenarios (a to d). Each scenario is shown with the time line in the middle. Above the time line the active grammars are shown. For scenario a, this is the permanent grammar *1—2—3—4* and the transient grammars *yes—no* which is only active while an option is delivered to the user. The system's output and the user's answers, in grey, are shown below the time line.

The menu with four options are presented to the user using the Index pattern. Scripted Interaction defines, how the user can perform a selection. Some commands are active all the time, while others are time constrained.

Usage scenario b shows, how Escalating Detail can be used, if the option was not recognized because the user may not have understood the option. A detailed description of the item is presented using Decoration. The *explanation* decorates the description with further information.

The next usage scenario c shows, how Detailed Information delivers the information with another granularity according to the user's need if she

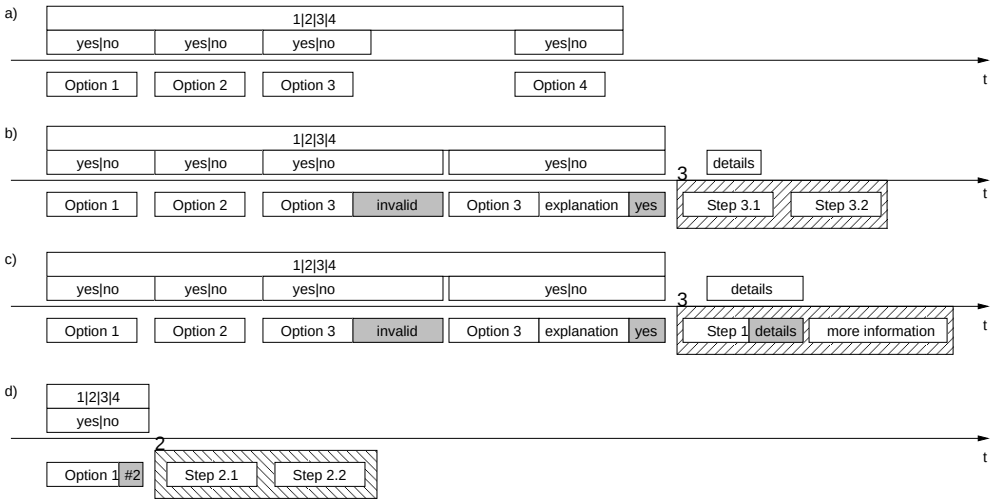


Figure 3: Audio Navigation Patterns in action

asks for *details*.

The last usage scenario d shows, how a user can activate an option efficiently, i.e. by pressing the *#2* button without waiting to hear the index. Active Reference is used in the shape of a background sound to bring unity to a set of actions that occur over time. This is indicated by the hatched background. The large amount of information information is being delivered using Information Spreading.

These patterns demonstrate that it is possible to have a language to talk about VUI designs. It enables designers to get a quick overview about the benefits and drawbacks of each design decision they make. We have started building a new pattern language that we hope will continue growing with the help of the VUI community.

4 Acknowledgments

Thanks a lot to Dietmar Schütz who shepherded this paper for EuroPLoP 2005. His comments were very useful and helped to develop this paper.We would also thank Jussi Kangasharju for reviewing the paper. Thanks also to the members of the writer's workshop, namely Lubor Sesera, Hans Wegener, Halina Kaminski, Tim Wellhausen and Andy Kelly, for their helpful discussion and comments about this paper.

References

[1] *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[2] Michael H. Cohen, James P. Giangola, and Jennifer Balogh. *Voice User Interface Design*. Addison-Wesley, 2004.

[3] James A. Larson et. al. Ten Guidelines for Designing a Successful Voice User Interface. `http://www.speechtechmag.com/issues/9_7/cover/11311-1.html`.

[4] Frankie James. *Representing Structured Information In Audio Interfaces: A Framework For Selecting Audio Marking Techniques To Present Document Structures*, page 23. 1998.

[5] SUN Microsystems. *Java Speech API Programmer's Guide*. SUN Microsystems, October 1998.

[6] M. J. Muller and J. E. Daniel. Toward a definition of voice documents. COIS, 1990.

[7] Dirk Schnelle, Jussi Kangasharju, Andreas Hartl, and Max Mühlhäuser. Stairs - navigation strategies. Technical report, Telecooperation Group, Department of Computer Science, Darmstadt University of Technology, 2004.

[8] VoiceXML. `http://www.w3.org/TR/voicexml20/`.