# Voice User Interface Design Patterns

Dirk Schnelle and Fernando Lyardet
Telecooperation Group
Darmstadt University of Technology
Hochschulstraße 10
D-64283 Darmstadt, Germany
`{dirk|fernando}@tk.informatik.tu-darmstadt.de`
phone: +49 (6151) 16-4267
fax: +49 (6151) 16-3052

**Abstract**

We present in this paper a set of design patterns we have mined in the area of Voice User Interfaces (VUI). In a previous paper [14], we introduced several patterns regarding fundamental issues of developing a voice application. In this paper we explore further aspects concerning the internal structure of an audio interface, the construction of the interaction style, the system response architecture, and implementation strategies to meet the demands of real world scenarios.

## 1    Introduction

Voice User Interfaces are particularly difficult to build due to their transient and invisible nature. Unlike visual interfaces, once the commands and actions have been communicated to the user, they "are not there anymore". Another particular aspect of VUI is that the interaction with the user interface is not only affected by the objective limitations of a voice channel, but human factors play a decisive role: auditive and orientation capabilities, attention, clarity, diction, speed, and ambient noise.

The design patterns presented in [14] demonstrated that it is possible to build a language to talk about VUI designs. In this paper we continue building this new pattern language, focusing on the different dialog strategies, auditory design, and usage scenarios.

The structure of the paper is as follows: we first provide an overview of the whole pattern language. Afterwards, we introduce the discovered patterns grouped by the VUI design aspect they address: Interaction Style, the organization of a spoken response and a final set of patterns that address voice usability issues in large scale settings.
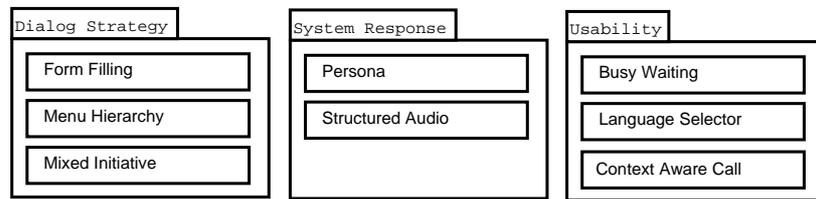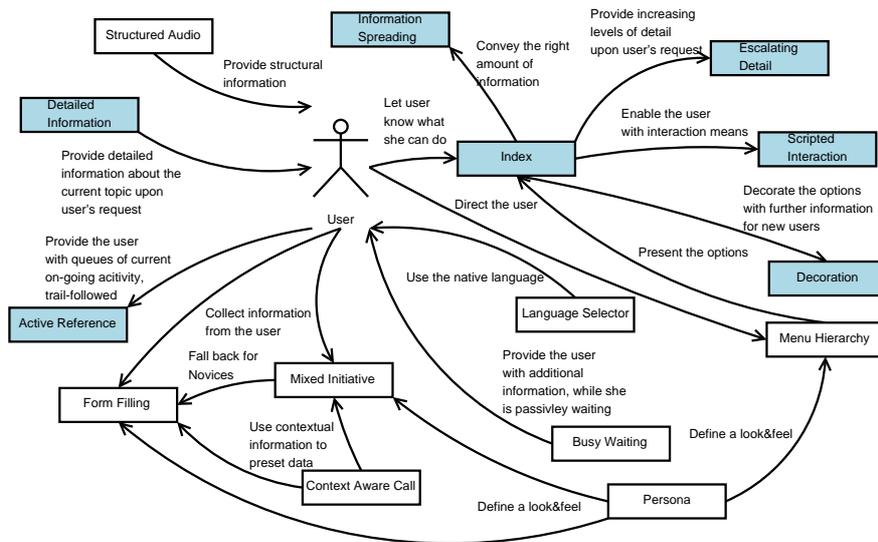
Figure 1: Categorization of Audio Patterns



Figure 2: Relation of Audio Patterns

## 2 Overview

In this section we present patterns we have mined and categorized according to their main purpose. An overview is given in figure 1. The existing patterns in the language (taken from [14]), marked with a grayed box, are not described in this paper.

The patterns belong to a language we started in [14], as shown in figure 2.

These new patterns expand the language in important areas of VUI design, closer to how the actual process of VUI design development actually takes place. In this context, the election and development of an appropriate *dialog strategy* is a first step designers must take when building a VUI interface. MENU HIERARCHY serves as general organizational scheme for many Audio Patterns. Novel designers would naturally apply it, although it is just one characteristic of a dialog strategy. Further dialog strategies are explained in section 2.1. Another major design concern described later in section 2.2

| Problems inherent to audio | Technical problems |
| --- | --- |
| Speech is one-dimensional | Speech synthesis quality |
| Speech is transient | Speech recognition performance |
| Speech is invisible | Recognition: Flexibility vs. Accuracy |
| Speech is asymmetric | |

Table 1: Challenges of an audio only application

focuses on the system output or rather, the *system response*. Note that the system response structure will be underlying the dialog structuring and interaction mechanisms implemented later, thus becoming the cornerstone of how the overall user interface will be perceived.

Nowadays, the design of efficient speech interfaces is considered by many to be more an art than an engineering science. This thesis is also supported by Junqua in [10] who concludes his book with the challenge:

> Today building the dialog component of a conversational system is more an art than engineering or science. We need to develop techniques for dialog design that are grounded on strong theoretical frameworks.

"Patterns and pattern languages offer an approach to design with much potential" [3]. The patterns described in this paper continue to build a pattern language to talk about VUI design.

An application designed for audio only input and output faces several challenges, shown in figure 1, before it can be named easy to use. Some of them are named in [11].

**Speech is one-dimensional** 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.

**Speech is transient** Listening is controlled by the short term memory. Listening to long utterances has the effect that users forget most of the information that was given at the beginning. This means that speech is not the ideal medium for delivering large amounts of data. An advantage of the transient nature of speech is that people can perceive visual and aural information in parallel. Because of this property, speech is ideal to deliver information without forcing the user to switch context.

**Speech is invisible** It is difficult to indicate to the user what actions she may perform and what words and phrases she must say to perform these actions. This invisibility may sometimes leave the user with the impression that she does not control the system.

**Speech is asymmetric** People can speak faster than they type, but can listen much more slowly than they can read. This has a direct influence on the amount of audio data and the information being delivered. This property is extremely useful in cases where we have the possibility of using additional displays to supplement the basic audio interface. We can use the displays for delivering information which is unsuitable for audio due to its length, and focus on using the audio device for interaction and delivering short pieces of information.

**Speech synthesis quality** The quality of modern text-to-speech engines is still low. In general, people prefer to listen to pre-recorded audio because it sounds more natural. However, for this to work, the data to be delivered has to be known in advance and recorded as audio, which consumes additional memory. For dynamic documents, where the content depends on the user's actions, text-to-speech may be the only feasible solution.

**Speech recognition performance** Speech is not recognized with an accuracy of 100%. Even humans are not able to do that. There will always be a doubt in the recognized input which has to be handled somehow.

**Flexibility vs. Accuracy** Speech can have many faces for the same issue and natural language user interfaces must serve many of them. This has a direct impact to recognition accuracy. To illustrate this trade off between flexibility of the interface and its accuracy, consider the following example for entering a date. A flexible interface would allow the user to speak the date in any format the user desires (e.g., "March 2nd", "yesterday", "2nd of March 2004", etc.). Another possibility would be to prompt the user individually for each of the components of the date (e.g., "Say the year", "Say the month", etc.). Obviously, the first method is much more flexible for the user, but requires much more work from the recognition software, and is far more error-prone than the second approach.

The first four items, one-dimensionality, transientness, invisibility, and asymmetry, are inherent to audio. It will be impossible to solve them completely, but it is important to know them and to find a workaround. The following items, quality of TTS, recognition performance, and flexibility, are technical problems by their nature. We assume that they can be solved somehow, see figure 1.

Existing guidelines [2, 5, 8] are essentially restricted to giving hints to master these challenges. This paper continues the development of a pattern language [14] integrating many guidelines and solution approaches developed during the past 15 years. The goal is to document and share this knowledge with new designers (see figure 3) as well as providing a language to talk
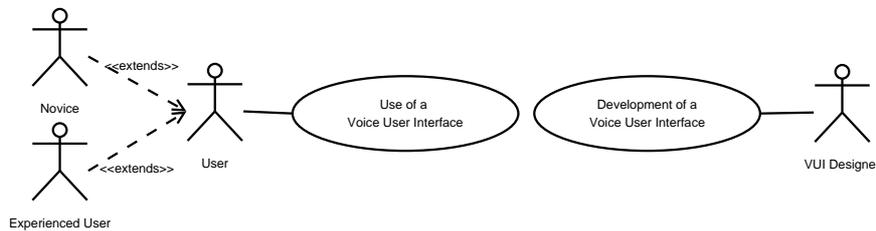
Figure 3: Simplified view on roles involved in VUI design

about *Voice User Interface* (VUI) designs. The notation of a pattern language allows for such a required theoretical framework that was requested by Junqua. 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 [18].

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.

Figure 3 shows a simplified overview of the roles involved in VUI design. *Novices* are users that have not used the application before, and are neither familiar with the concepts of the application nor the grammar they can use. *Experienced users* have a systematic understanding of the application and its functionality. The *VUI designer* develops the application and the concepts on how to present it to the user.

For some of the patterns we suggest to use a *wizard-of-oz* experiment [6]. This is a well known strategy taken from the field of human-computer interaction. Test persons interact with an application which they believe to behave autonomous. Actually it is operated by an unseen human being, the *wizard*. Instead of real voice recognition, the user's input is manually entered into the system and processed as a text stream rather than an audio stream. These experiments should reveal information about use and effectiveness of the application.

The name of the experiment comes from *The Wonderful Wizard of Oz* story, in which an ordinary man hides behind a curtain and pretends to be a powerful wizard.

## 2.1 Dialog strategy

As GUI-based applications, voice applications interaction consists of *user input* and *system output*. User input can be [4] an audio recording of a user, touch tone, or more commonly voice recognition of a user's speech. Similarly,

system output can be text-to-speech or pre-recorded sound.

This section addresses various interaction styles, also known as *dialog strategies* [2]. All these patterns feature a common set of forces:

- Dialogs have to to be efficient and short

- Dialogs have to be clear and structured

- Novices have to be guided. They need to know what kind of information to provide

- Experienced users know what to say and need a fast way to enter the data

### 2.1.1   Form Filling

**Intent**   Gather information from the user.

**Context**   The user has to provide data that can't be gathered through a selection process. Forms are a well known concept of the real world to provide information in a sequential order, one at a time.

**Problem**   How to collect structured information from the user?

**Forces**

- Forms forces users to provide one datum at a time

- Natural language communication allows to provide more than one datum at a time

- Speech is invisible: Users might forget what they entered in the beginning

- Speech recognition performance: Users need feedback about the entered data

**Solution**   Identify a short description or label of the field to be filled in and prepare a variable to store the entered information. Use comprehensive instructions for the label [15] using a consistent, brief, and clear terminology with which the user is familiar. Prepare a grammar to retrieve the valid values for the fields. Ensure that the label contains clues about the format of the user's utterance. To ensure the correctness of the data, an own error handler for each field may be used.

Present the label to the user, followed with an optional input prompt and silence to let the user enter the data. as if she were filling out a form.

Present a summary of all entered data to let the user confirm the data or re-enter them.

**Structure**   This is illustrated in the following figure. The user is requested
for an utterance in node *Field 1*. If she does not make an utterance within
a certain time, or made an erroneous input, she will be reprompted. If all
goes well, the dialog proceeds to the next field until she reaches the node
*Summary*, where she is asked to confirm a summary of her inputs. If she
rejects the summary, the dialog proceeds with the first field for another try.

   In an alternative approach, the user can be prompted to confirm each
field separately.
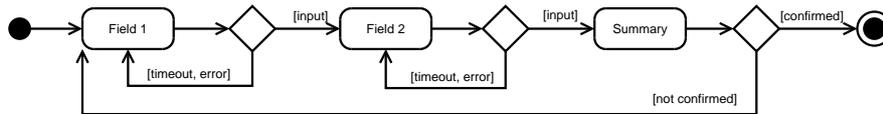


**Consequences**

  + Dialogs are clear and structured

  + Good for beginners/novices

  + System messages are not to be confused

  o System controls dialog flow

  - Dialogs appear rigid and inflexible

  - Dialogs become lengthy

  - Does not solve the problem to provide more data at a time

**Known Uses**   The Poldi Gewinnspiel of the 1. FC Köln used FORM FILL-
ING to request the desired information about the caller, like membership of
the 1. FC Köln and to enter the answers to the quiz. The application went
offline in 2005, but was callable at +49 (180) 5 29 00 29.

   The dating line L.U.C.Y. from Com Vision uses FORM FILLING to request
the data of the caller, i.e. *sex*, *age* and *ZIP code*. Having entered all data,
the system summarizes all data in one go. The application can be called at
+49 (12) 345 662 662.

   The Citiphone Banking application uses FORM FILLING to enter the
data, i.e. *account number* or *bank code* for a bank transfer. After the caller
entered a value for a field, she is asked for confirmation. The application can
be called at +49 (180) 33 22 111.

**Related patterns**   FORM PATTERN [17] solves a similar problem for graph-
ical user interfaces.

   ESCALATING DETAIL as a means of error recovery.

   MIXED INITIATIVE is an alternate approach and enables the user to pro-
vide more than one datum at a time.

**Sample code**   In a car inspection scenario, the worker enters the data after a car has been repaired. He enters his worker *id*, the *order number* and other data that belongs to the order such as *part numbers* in a form before the back-end system continues with the billing process.

Note: The VoiceXML `<form>` element must not be confused with the form concept that is described in this pattern. `<field>` elements are just one side of the elements which a `<form>` may contain.

```
<form id="prepare_bill">
    <field name="id" type="digits">
        <prompt>Please say your id.</prompt>
    </field>

    <field name="order" type="digits">
        <prompt>Please say the order number</prompt>
    </field>
    ...
    <block>
        <submit next="http://www.example.com/servlet/bill"
                namelist="id,order,part_number"/>
    </block>
</form>
```

**Also Known As**   Directed Dialog, System Initiative

### 2.1.2   Menu Hierarchy

**Intent**   Guide the user to the information of interest

**Context**   The user has to provide data that can be gathered through a selection process. The options to be presented to the user are interrelated in a hierarchy, or can be made to appear that way.

**Problem**   How to guide the user to a certain piece of information?

**Forces**

- Hierarchical structures are easy to understand

- Not all options can be grouped in a hierarchical structure

- Long menus more efficient [16]

- Speech is one-dimensional: Long menus force the user to wait until the wanted option is presented

- Speech is one-dimensional: The system's response time to present a menu is very high

- Speech is invisible: Users cannot see the structure of the data

- Speech is transient: The user has to remember the path

**Solution**  Form categories of similar option items to create a tree structure [13]. If this is not clearly solvable, the initial design can be improved upon the user's feedback. Use a natural sequence of the items to determine the presentation sequence [15]. In case there is no natural sequence, choose an order that is obvious to the user, such as most frequently used items first or alphabetic sequence of terms. Present these options in a menu to the user. Avoid listening to long prompts by using barge-in. Aid the user by naming the appropriate commands in the prompt [20]. Enable the user to deviate from prompted words by specifying alternative words in the the grammar or use the touch tone key-pad. Conduct *wizard-of-oz* experiments to evaluate the menu with respect to the wording of the prompts and the grammar. Once, the user makes a selection, the system offers another menu until the application settles on the one item or action the user wants. Remember the user's path to allow for going up in the hierarchy.

The limitations of the short-term-memory has been studied by several authors and is still discussed in the HCI field. Some authors like Miller [12] suggest that a person can only remember 7+/-2 information chunks. In an ongoing project we conducted a user study were we found that more than 6

options are rejected by most users. Since the path must also be kept in the memory, we assume, that this limit is also applicable for the depth of a menu tree. Therefore, the length and the depth of such a menu tree should be as flat as possible, and should also not exceed the *magical number 7*.

**Structure**  This is illustrated in the following figure. The user is requested for a choice in node *Option 1*. Depending on her answer the dialog proceeds with the appropriate next option until it reaches a leaf.
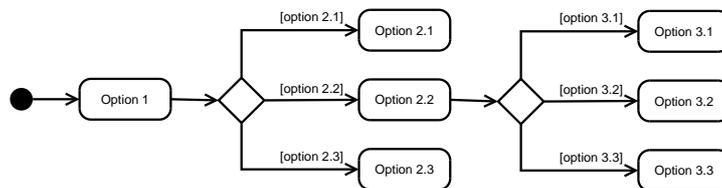


**Consequences**

+ Dialogs are clear and structured

+ Good for beginners/novices

+ The user is guided by the system and gets more details on demand

+ The user is able to control the leaves she wants to enter or leave

o System controls dialog flow

- Dialogs are too structured

- The *lost-in-space* problem increases when the number of options or the depth of the hierarchy are too high

- Poorly chosen categories force the user to re-enter the menu to find the category that matches their reason for calling [16]

**Known Uses**  The traffic information InfoTraffic from Viasuisse uses a hierarchical structure to enter the route, in which the caller is interested. Since Switzerland is multilingual, the application first asks for the desired language. It continues in the chosen language to enter the desired kind of traffic, like highways, train, etc. Having chosen i.e. highway the application turns to FORM FILLING to enter the id of the highway. The application can be called at +41 (900) 400 500.

**Related patterns**  Typical applications use FORM FILLING and MENU HIERARCHY at different points, depending on the information and the user's mental model [2].

INDEX or SCRIPTED INTERACTION may be used to present the available options.

ACTIVE REFERENCE can be used to help the user stay oriented in the hierarchy.

HIERARCHICAL SET [17] solves a similar problem in visual oriented applications.

**Sample code**  At the car inspection the worker browses for information about a part that needs to be exchanged. The parts are sorted in a hierarchical structure. Having navigated to the exhaust system, she is been asked for the sub-part in which she is interested.

```
<menu>
    <prompt>
        Say the part of the exhaust system <enumerate/>
    </prompt>
    <choice next="http://www.example.com/pipe.vxml">
        Exhaust pipe
    </choice>
    <choice next="http://www.example.com/converter.vxml">
        Catalytic converter
    </choice>
    <choice next="http://www.example.com/rear_muffler.vxml">
        Rear muffler
```

```
    </choice>
    ...
</menu>
```

### 2.1.3  Mixed Initiative

**Intent**   Collect information from the user.

**Context**   Users with a different level of experience have to enter some piece of information that may have interdependencies. The dialog should imitate a natural human-to-human conversation.

**Problem**   How to collect information from the user seamlessly?

**Forces**

- Users want to enter data without too many restrictions

- It is possible to say the same thing in *endless* different ways.

- Combination of user data grows by a factorial order

- Grammars can cover only part of the ways to capture user input

- Speech is invisible: Users may not know what to say

**Solution**   Start the conversation with an open ended prompt, i.e. *How may I help you?*. Create a field, also known as *slot* for each piece of information to obtain from the user. Allow the user to provide more information than the current question expects, i.e.

**System:** Where do you want to go?

**User:** I want to go to Darmstadt by train at 17:00.

   Also allow filling or corrections of field that are out of focus, i.e.

**System:** At what time do you want to go to Darmstadt?

**User:** No. I want to go to Mannheim.

   Use references in the dialog context, i.e.

**System:** Bayern München lost 2:3

**User:** What is their current rank?

Use barge-in to enable the user to interrupt system prompts. If the application's task can be divided into multiple sub-tasks, do not stick to the principle of *Say what you want at any time you want to* described above. This reduces the occurrence of the system's misbehaviour in the presence of recognition failures.

Use suitable follow-up prompts, once the field gets filled. Present the collected information in a final prompt and ask the user for confirmation. Use ESCALATING DETAIL to aid the user in case of a timeout (the user did not say anything) or in case of a reject (the utterance could not be mapped to the grammar). Use FORM FILLING as the last escalation level. Conduct *wizard-of-oz* experiments to evaluate the grammar.

**Structure** An example of a structure is shown in the following figure. The user is requested to provide information for *field1* and *field2* after the initial *InputPrompt*. If she provides information for *field1* she will be prompted for *field2* in *PromptField 2*, whereas if she provides information for *field2* she will be prompted for *field1* in *PromptField 1*. Having provided the missing piece of information she will be asked to confirm the entered information in *ConfirmAll*. As an alternative she can provide information for both fields at once and will be asked to confirm all in the next dialog step.
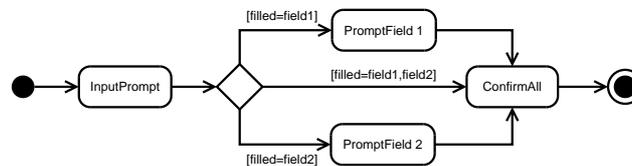


## Consequences

+ Dialogs are more natural

+ Users can provide more than one information item at once

+ Dialogs can become more efficient and shorter

+ Users can take control of the dialog flow

- Recognition rate decreases since the user is free to make any utterance

- User is fooled to think she is in a real free form entry, which will cause her to use words that might not belong to the grammar

- *Turn taking* (When should or can the user speak?) and *intention recognition* (What are the goals of the user?) come up as new problems [7]

**Known Uses**   The cinema Cinecitta in Nuremberg offers a voice portal as a cinema information. After calling the system gives a short overview about its capabilities. The user is free to interrupt the system at any time. If the user does not make an utterance or calls for help, the system falls back into a FORM FILLING like dialog strategy, by asking first for the hour and then presents a list of films that are shown at the given time. The user is free to resume the initiative again at any time. The application can be called at +49 (911) 20 66 67.

**Related patterns**   ESCALATING DETAIL is used to aid the user to get back on track.

If the user fails to enter the data, the dialog strategy may fall back into FORM FILLING.

PERSONA might be used to make the dialog appear more natural.

**Sample code**   In a car inspection scenario, the worker enters the data after a car has been repaired. He enters his worker *id*, the *order number* and other data, that belongs to the order, such as *part numbers* into the system before the back-end system continues with the billing process. The worker can enter all data at once or in a sequential order.

```
<form id="repair_done">
    <grammar src="repair.grxml"
             type="application/srgs+xml"/>

    <initial name="get_info">
      <prompt>Say the data for the bill</prompt>
      <catch event="nomatch">
         <prompt>
             Let us try getting each field separately.
         </prompt>
         <reprompt/>
         <assign name="get_info" expr="true"/>
      </catch>
    </initial>

    <field name="id">
        <grammar src="reapair.grxml#id"/>
        <prompt>Please say your id.</prompt>
    </field>

    <field name="order">
        <grammar src="repair.grxml#order"/>
        <prompt>Please say the order number</prompt>
    </field>
    ...
```

```
    <block>
        <submit next="http://www.example.com/servlet/bill"
                 namelist="id,order,part_number"/>
    </block>
</form>
```

## 2.2  Design of the system response

Developing voice based applications with VoiceXML is straightforward. The design of quality speech interfaces on the other hand, is a non-trivial task. This section introduces some patterns that can help to design the system's response to the user input.

### 2.2.1  Persona

**Intent**   Define a *look & feel* for voice based applications.

**Context**   Users of voice based applications build their own mental image of a personality or character that they infer from the application's voice and language. Such mental image relates certain properties to the virtual dialog partner, where systems responses should fall within a foreseeable range of possibilities.

**Problem**   How to realize a *look & feel* for voice based applications?

**Forces**

- Interests of the target groups are different

- Interaction with voice based application should provide an underlying coherent and stable personality

- The system's responses must be consistent. New characters distract the user.

- Dialogs should appear more natural

- Users know that they are talking to a machine

- The character must match the user's mental image of the application

**Solution**   Identify the audience and enumerate the benefits to members of the intended audience. Specify what impressions and feelings the application should convey to the audience. Describe the fictional persona by writing a short biography of her to get hints, how to phrase messages and prompts. Define the wording of each message and prompt so it conveys the persona's message and personality. Compare several synthesized voices and select the voice with the tone, accent and energy that matches the persona's. Determine the speed, tone and prosody of each prompt. Use markup languages like SSML [19], to encode the voice characteristics or a voice actor emulating the persona, when recording the messages and prompts. Assure congruence of synthesized and prerecorded audio if both techniques are combined. In addition, choose the appropriate non-verbal sounds, background music or pauses to improve the clarity of prompts and messages. Make sure, that the usage is consistent throughout the whole dialog.

**Consequences**

+ VUI anticipates the caller's needs

+ Callers of the target group are satisfied

+ Common *look & feel* for the application

- Callers outside the target group feel sidestepped

- Users are distracted if the the persona is chosen wrong

- Not applicable if the target group varies widely

**Known Uses**   BERTI, the soccer information service from Sympalog uses the fictive person BERTI to interact in a MIXED INITIATIVE dialog with the caller. The application can be called at +49 (9131) 61 00 17.

The Poldi Gewinnspiel of the 1. FC Köln used a very famous person to create their PERSONA: Lukas Podolski. The application went offline in 2005, but was callable at +49 (180) 5 29 00 29.

**Related patterns**   PERSONA is often used as an extension to the chosen dialog strategy, FORM FILLING, MENU HIERARCHY, or MIXED INITIATIVE, to make the dialog appear more natural.

**Sample code**   In our car inspection scenario, the customer can call to get information about the progress of the repair. He has to identify herself with her customer id to retrieve the current repairs that are associated with this customer.

The sample code provides an example, how SSML tags can be used to use prosodic information within an application to control the TTS output.

This example can give only an idea about this pattern. The decision for a persona is more or less a philosophical question, that has to be consistent throughout the whole dialog.

```
<field name="customer">
  <prompt>
    Hi, my name is Bill. I can give you some information
    about the status of your car repair.<break/>

    Please say your
    <emphasis level="strong">customer id</emphasis> that
    is printed on your copy of the
    <emphasis level="strong">order</emphasis>.
  </prompt>
  <grammar mode="voice" root="customer_id">
  ...
  </grammar>
</field>
```

### 2.2.2 Structured audio

**Intent**   Provide structural information in audio.

**Context**   Graphical documents offer many possibilities to structure the contents by means of headlines, different fonts and many other. This structure helps the user to stay oriented and allows for easy and fast access to the desired information.

**Problem**   How to provide information about document structure in audio?

**Forces**

- Structural information helps the user for easy and fast access to information

- Visual content is a fundamental part of any document

- Structural meta information gets lost while reading the document

- Voice applications have no direct means to represent structural information

**Solution**   Use sounds in form of *auditory icons*, *earcons* or *background sounds* to represent the structure. As an alternative, different speakers may also be used [9].

**Structure**   The following figure shows an example, how an auditory icon *item.raw* is used to introduce the distinct items, *Item 1* and *Item 2* read by the TTS engine.
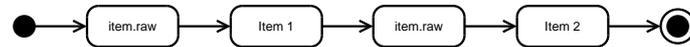


**Consequences**

+ Use of structural information in audio

+ Easier for the user to concentrate on the content

- A sound may have a different meanings to users

- Exhaustive use may distract user's attention

**Known Uses**   The AHA framework [9] uses STRUCTURED AUDIO to render structural information of web pages in audio.

**Sample code**   At our car inspection scenario, the worker listens to a list of repairs that have to be done. Each item is marked with an item sound to indicate the list structure.

```
<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>
    ...
</field>
```

## 2.3   Usability in Business Scenarios

We present some of the most common usability issues in publicly available voice applications. In particular, these problems and cardinal aspects that we consider should be addressed in voice applications that are expected to become a central part of the communication and information strategy of any organization.

### 2.3.1   Language Selector

**Intent**   Support language selection for users.

**Context**   Voice applications usability is affected by a number of objective factors such as speed, noise and attention. But also by other more subjective, such as vocabulary, expressions, intonation and pronunciation. All these factors introduce different degrees of complexity when accessing content through a voice interface. However, the language remains as the biggest barrier.

**Problem**   How to provide users with an effective language selection mechanism?

**Forces**

- Global companies and current increasing migration within multilingual economic regions such as the EU or Mercosur force services to be offered in multiple languages

- Users want to access a service in their native language

- Service has to be translated

- The default language of the company that offers the service is unknown to users

- Recognition of multiple languages concurrently is not supported by all recognizers

**Solution**   Order the languages to be supported according to their importance in the host country or the area of the country. Use the list of official languages of the country in case of doubt. Greet the caller in all languages that are supported or in the language to be considered dominant. Ask the user for the language to use in each language in the determined order. If the recognizer does not support concurrent recognition of multiple languages, use SCRIPTED INTERACTION to ask simple *Yes/No* questions in each language

**Structure**   This is illustrated in the following figure. After the *Greeting*, the application prompts the option of interaction with the language *L1* in that language and the option for *L2* in language *L2*. After the user has chosen the language, the application continues with the selected language in *L1 Application* or *L2 Application.*
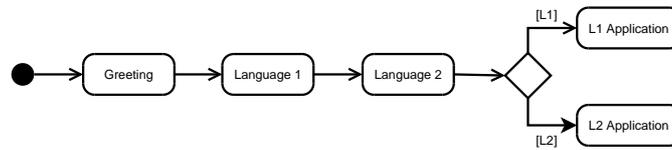
## Consequences

+ Allow users to easily recognize an option

+ Enable users with little foreign language skills access the system

- Listening options in many different languages can be confusing

- Problem of translating the application remains unsolved

**Known Uses**   An example can be found in major airlines services such as the United Airlines customer service: +49 (69) 50 07 03 87.

The traffic information InfoTraffic from Viasuisse asks the user in all official languages of the Switzerland in the language to determine in which language the user want to interact with the system. The application can be called at +41 (900) 400 500.

**Related patterns**   SCRIPTED INTERACTION can be used to ask for the language. CONTEXT-AWARE CALL can be used to automatically suggest the language from the caller's ID.

**Sample code**   If the customer calls our international garage, if her car is ready, the application asks for the language, sh wants to use.

```
<form id="language">
    <field id="language">
        <prompt>
            Welcome to the car inspection.
            <p xml:lang="en">Do you want to use English?</p>
            <p xml:lang="de">
                Moechten Sie Deutsch verwenden?
            </p>
            <p xml:lang="es">Desea acceder es Espanol?</p>
        </prompt>
        <grammar src="language.grxml"
                 type="application/srgs+xml"/>
        <filled>
            <if cond="language='english'">
                <goto next="en/car.vxml"/>
            <elseif cond="language='deutsch'">
                <goto next="de/car.vxml"/>
            <elseif cond="language='espanol'">
```

```
                <goto next="es/car.vxml"/>
            </if>
            <reprompt/>
        </filled>
      </transfer>
</form>
```

### 2.3.2   Busy Waiting

**Intent**   Provide a meaningful interaction to the users waiting to be attended.

**Context**   Personalized attention is a scarce resource. More and more, companies are rapidly moving to provide their customers to centralized callcenters with an increasing number of self-service functionality in order to cope with the ever-growing demand. However, human intervention to resolve conflicts cannot be totally replaced. In these scenarios, bottlenecks are more likely to appear as human attention (operator) does not scale well but only through the addition of more operators. As such, customers looking for human attention often find themselves waiting to be attended.

**Problem**   How to transform the passive user wait in a phone line into a more active, engaging experience (and keep them waiting!)?

**Forces**

- Waiting is avoided by customers

- Avoid users leaving the service (possibly in favor of a competitor)

- Minimize the time users spend waiting to be attended

- Physical limitations of the telephone system or used services to attend more people at the same time

- The company wants to offer the service, but adding more operators is costly

- Users expect a 24x7 service

- Not all call centers are available 24x7

**Solution**    The solution needs to address the uncertainty of the waiting status (Am I going to be attended? Will it take long? How many people are already waiting? Why keep waiting?). There are number of information that can be conveyed to the user to bring clarity and sense of the time spent *queued*:

Clearly state if the service is available 24h or not. Mention the time-zone, if the application covers an area over multiple time-zones. At regular intervals of time provide some information.

This information can be one or a combination of the following modules. This can vary, depending on the target group.

- Let the user know that the system is aware of the waiting, apologize and reinforce the message that the user will be attended as soon as possible.

- Provide information about how many people are on the queue (as a subjective delay measure) or better, provide an estimate of the waiting time

- Offer alternative ways to perform a request. For instance, offering the user self service options or directing users to the customer website.

- Provide short advertisements about new services, products and last minute offers.

- Play a smooth and calming music in the background. This can be combined with brief corporate or branding messages and intervals where only the music is heard.

**Structure**    This is illustrated in the following figure. In *Connect* a caller should be connected to a technician. During the transfer she hears music. If the connection was successful, she gets transferred to the technician. If a timeout occurs, she can be provided with some other information, i.e. numbers of persons waiting, before she hears the music again.
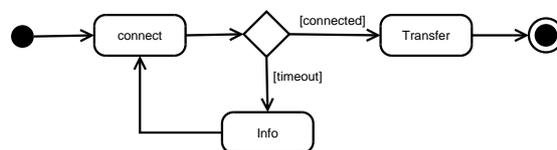


**Consequences**

+ Users become more satisfied because they know more.

+ Changes the waiting period into informative moments

- It doesn't solve the need of more operators

- Users might feel caught by the application if they get information in which they are not interested

- Telephones usually have a lower sound quality than a radio. Therefore, playing advertisements originally prepared and recorded for radio emission may not be easy to follow and understand

**Known Uses**  The customer service at T-Online informs the user upon alternative request possibilities, if all technicians are busy. While the user is in the waiting queue, she gets notified that she will be connected to a technician as soon as possible from time to time. The application can be called at +49 (180) 5 30 50 00.

**Related patterns**  A Place To Wait [1] with the constraints of the audio domain and a virtual environment

**Sample code**  If the customer calls the garage, if her car is ready, she may end up in a queue. Each 60 seconds, she will be informed about the number of persons to be served first.

```
<form id="xfer">
    <transfer name="mycall" dest="tel:+1-555-123-4567"
              transferaudio="music.wav" connecttimeout="60s"
              bridge="true">
        <prompt>
            Say cancel to disconnect this call at any time.
        </prompt>
        <grammar src="cancel.grxml"
                 type="application/srgs+xml"/>
        <filled>
            <object name="size"
                classid="method:///queue.size"
                data="http://www.example.com/queue.jar">
            </object>
            <prompt>
                There are only <value expr="size"/>
                persons to serve before you.
            </prompt>
            <reprompt/>
        </filled>
    </transfer>
</form>
```

Note, that this example would not really work, since the caller would be disconnected and enqueued again, if the connection timeout expires. The reason is the limitation of the `<transfer>` element in VoiceXML. A workaround

for VoiceXML is to dynamically create the audio file *music.wav* and synthesize the announcements to this file. Proprietary implementations with a CTI connection to the telephony system do not have this limitation.

### 2.3.3 Context-aware call

**Intent**   Provide more meaningful interaction to the users.

**Context**   Requesting services and goods over the phone is a common part of our lives. These voice applications are continuously applied everyday into the most diverse purposes. From appointments and accessing bank services to ordering a pizza or a taxi, spoken requests are an every day activity.

A certain amount of context information is often required for any request in order to be meaningful. In voice interfaces, such information must be provided by the user.

**Problem**   How to minimize the overhead of requesting information to perform a request?

**Forces**

- The process of conveying directions and specifications of what is being asked can be time consuming when the amount of information is large

- A request cannot be fulfilled unless all the required information is available

- When a service must be repeatedly requested, providing the same information is time consuming

- Users do not feel annoyed, if they do not use the service frequently

- Users need feedback about the information which the system already has, and temporal orientation

- A low speech recognizer performance makes that data entry more error prone

- Context information is not available for third parties yet

**Solution**   Context information can help voice interaction systems be more responsive and better adapt their response to the users. A common source of context information in voice applications is brought for instance in each phone call by the caller ID service. Use this information to determine actions and default values for data that the user has to provide to use the service.

**Structure**   This is illustrated in the following figure. If contextual information *ctx* is available, it is obtained in *GetContext*. If *ctx* is not available, the user has to enter it manually in *Enter* before the information can be processed in *Process*.
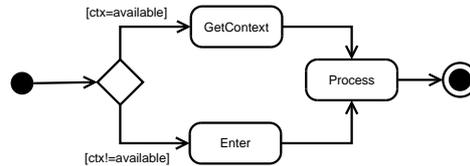


**Consequences**

  + Personalized service. The Information about who's calling and potential needs and preferences is already available

  + Less mistakes on the overall process of attending a request. Elimination of repetitive data entry favors accuracy

  + More efficient interaction between the user and the operator. Substantially more calls can be attended in the same amount of time

  + Other planning and logistic systems can be integrated and use the contextual information to schedule service according to other criteria such as logistics or customer rate

  - Cannot be applied for services that requires authentication, since a caller ID might not be enough to identify the caller for critical applications

**Known Uses**   A very popular application is the Memobox implemented by communication providers, which uses the user ID in order to automatically enable the access to the private recorded messages. In this case, the telephone (either fixed or mobile) becomes an identification (not authentication) token.

Another example is the case of a taxi request system available in Argentina (+54 221 427 4500), which by default uses the caller ID information to determine the address where a Taxi should pick up its passenger, unless otherwise instructed. When the call is made from mobile phone, this fact is reflected to the operator who in that case asks the customer if the pick up location is the same as last time or should be found somewhere else. The use of context information helps the taxi company improve its efficiency keeping the average customer call very short (8-12 secs), just enough to acknowledge the existing address is correct and confirm the request.

The Israeli Electric company has started printing a *Contract Number* field on the electricity bills as a single *handle* that is typed when calling their VUI system.  This handle replaces a whole group of numeric entry fields that were required to bill with a credit card.

**Related patterns** In FORM FILLING, context-aware call can help with automated filling of fields with known information about the caller.

**Sample code** In a car inspection scenario, the worker has to get a list of which repairs have to be done for the next car. This information can be entered manually by reading the license plate to the system or by a location system, that automatically detects that the worker is standing near to a certain car.

```
<form id="get_order">
    <object name="location"
            classid="method://Locator/locate"
            data="http://www.example.com/location.jar">
        <param name="worker" expr="id"/>
    </object>

    <block>
        <if cond="location">
            <submit next="http://www.example.com/read.jsp"
                    namelist="location"/>
        </if>
    </block>

    <field name="enter_location">
        <grammar src="order.grxml"
                 type="application/srgs+xml"/>
        <prompt>Please read the license plate</prompt>
        ...
        <filled>
            <submit next="http://www.example.com/read.jsp"
                    namelist="enter_location"/>
        </filled>
    </field>
    ...

</form>
```

# 3 Summary

We have presented in this paper 8 new patterns in the area of Voice User Interfaces (VUI). Several known guidelines for VUI design are covered. The design for VUI applications has its roots in the one-dimensional, transient and invisible nature of the audio medium, enhanced by technical limitations, such as speech synthesis quality and speech recognition performance. 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.

The new patterns consistently build on the previously mined VUI patterns and show non-trivial design decisions in three different aspects of VUI applications such as Dialog Strategy, System Response and Usage Scenarios.

FORM FILLING, MENU HIERARCHY and MIXED INITIATIVE are the fundamental decision of the dialog strategy that the designer follows in her application.

PERSONA and STRUCTURED AUDIO help to design the system response.

BUSY WAITING, LANGUAGE SELECTOR, and CONTEXT AWARE CALL are some common usability issues in business scenarios.

These patterns are the continuation of a new pattern language, introduced in [14]. We hope to continue growing it with the help of the VUI community.

# References

[1] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language: Towns, Buildings, Constructions.* Oxford University Press, UK, 1977.

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

[3] Andy Dearden and Janet Finlay. Pattern Languages in HCI; A Critical Review. *Human-Computer Interaction*, 21, 2006.

[4] Bryan Duggan. Revenue Opportunities in the Voice Enabled Web. Technical report, School of Computing, Dublin Institute of Technology, Kevin St. Dublin 8, Ireland, 2002.

[5] James A. Larson et. al. Ten Guidelines for Designing a Successful Voice User Interface. *Speech Technology Magazine*, January 2005.

[6] Norman M. Frazer and G. Nigel Gilber. Simulating speech systems. In *Computer Speech and Language*, volume 5. Academic Press Limited, 1991.

[7] Marti A. Hearst. Mixed-initiative interaction. *IEEE Intelligent Systems*, pages 14–16, September 1999.

[8] Speech Science Institute. Guidelines for Developing Voice User Interfaces. `http://www.ssi-interactive.com/vui_guidelines1.htm`, 2004.

[9] Frankie James. *Representing Structured Information In Audio Interfaces: A Framework For Selecting Audio Marking Techniques To Present Document Structures.* PhD thesis, Stanford University, 1998.

[10] Jean-Claude Junqua. *Robust Speech Recognition in Embedded System and PC Applications.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.

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

[12] George A. Miller. The magical number seven, plus or minus two: Some limtis on our capacity for processing information. *Psycological Review*, 63:81–97, 1956.

[13] Kent Norman. *The Psychology of Menu Selection: Designing Cognitive Control at the Human/Computer Interface.* Ablex, Norwood, NJ, 1991.

[14] Dirk Schnelle, Fernando Lyardet, and Tao Wei. Audio Navigation Patterns. In *EuroPLoP 2005 Conference Proceedings*, 2005.

[15] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[16] Bernhard Suhm, Barbara Freeman, and David Getty. Curing the Menu Blues in Touch-tone Voice Interfaces. In *CHI '01: CHI '01 extended abstracts on Human factors in computing systems*, pages 131–132, New York, NY, USA, 2001. ACM Press.

[17] Jenifer Tidwell. *Designing Interfaces.* O'Reilly, 2005.

[18] VoiceXML. `http://www.w3.org/TR/voicexml21/`.

[19] W3C. Speech Synthesis Markup Language (SSML) Version 1.0. `http://www.w3.org/TR/speech-synthesis/`, September 2004.

[20] Nicole Yankelovich. How Do Users Know What to Say? *ACM interactions*, 3(6), November 1996.