A Pattern Language for Error Management in Voice User Interfaces

Dirk Schnelle-Walka
Telecooperation Group
Darmstadt University of Technology
Hochschulstraße 10
D-64283 Darmstadt, Germany
dirk@tk.informatik.tu-darmstadt.de
phone: +49 (6151) 16-4267

June 28, 2010

Abstract

Error handling is a crucial aspect of the success of voice based interfaces since the underlying technology always leaves some uncertainty about the recognized input. In this paper a pattern language for error management in voice based interfaces is presented, enabling designers and developers to deal with this limitation. It integrates into our pattern language for voice user interfaces.

1 Introduction

Speech is not recognized with an accuracy of 100%. Even humans are not able to do that. There will always be some uncertainty in the recognized input, requiring strategies to cope. This is different from the experience with graphical user interfaces, where keyboard and mouse input are recognized without any doubts. Speech recognition and other errors occur frequently and reduce both the usefulness of applications and user satisfaction. This turns error handling into a crucial aspect of speech applications. Successful error handling methods can make even applications with poor recognition accuracy more successful. In [9] the authors show that the task completion rate increased from 86.4% to 93.4% and the average number of turns reduced by three after a better error handling method had been installed. On the other hand, poorly constructed error handling may bring unwanted complexity to the system and cause new errors and annoyances.

The patterns presented in this paper integrate into a pattern language for voice user interface design [11] where the pattern language that was presented previously in [13] and [12] is extended. This pattern language made only few approaches to error handling by introducing the Escalating Detail pattern which was first discussed in [13]. This paper introduces a more detailed concept of error management by means of a dedicated pattern language. The referenced patterns of this pattern language were previously published in [11].

Most patterns are designed to be applied in telephony environments. As error handling is a key factor in the design of voice based dialogs, the concepts are relevant for multimodal interaction as well. Some known usages are taken from this context. In this case it is also possible to use the additional modality for error correction which is not exploited in the presented patterns since the pattern language restricts to voice-only interfaces.

Error management is usually separated into error detection, error reasoning and error correction which will be explored in more detail in the following sections.

1.1 Error Detection

Errors can only be corrected if they are detected. Since errors can have different causes an automated error detection is not always possible. Semantic errors are hardly noticed by users thus requiring automatic error detection methods. Automatic error detection is not trivial, however. Recognition confidence scores can help but may not be sufficient indicators. In 1994 Kamm found that many applications left error detection to the user [5], but both the user and the system should be able to detect errors. This became of greater importance in the development during the recent years. The following categorization of different error types taken from [3] still serve as a basis for error handling.

Level 0 Missing input.
The user was requested to provide an input but did not say anything.

Level 1 Recognition rejection.
The user said something that did not match an expected input.

Level 2 Recognizer returns something that cannot be interpreted (makes no sense at all).
This can happen if the user's utterance was falsely mapped to an allowed sequence of words. Imagine a voice automated train schedule where the user says e.g. *The weather will be fine today* which is falsely recognized as *What five today.* Generally this is often referred to as the *out-of-vocabulary problem* (OOV).

Level 3 Recognizer returns something that is not semantically consistent.
In this case an utterance was accepted that matches an allowed sequence of words but has semantic faults. In our train schedule scenario, the user may say something like *I want to leave at 7 kilos.*

Level 4 Recognizer returns semantically well formed, but impossible to fulfill sentences.
Here, the user requests some information that is consistent and makes sense to her, but the system is not able to provide that information. An example would be, if the user asks the train schedule system for a train connection from Darmstadt to Frankfurt, but the connection is currently not served due to ongoing repairs.

Level 5 Same as 4 with the exception that the impossibility is due to the dialog context.
An example for this level is that the user selects the fifth available train connection, although there are only four of it.

Level 6 The back-end system fails to fulfill the command
If e.g. the database connection is lost it will not be possible to deliver any information.

Level 7 User initiated error correction.
This level summarizes all kinds of errors that are not covered by the levels above and can not be identified automatically.

Each of the levels need different strategies to handle the error. For instance, semantic errors like those from Level 5 need automatic error detection methods while others can only be identified by the user herself.

1.2 Error Reasoning

The analysis of the cause of an error is the key to better error correction. Without analysis, it is possible that both the system and the user repeat the same errors, thereby causing what Turunen calls the *error spiral* [14]. In most cases the cause of the error is related to the recognizer, see figure 1, but other reasons should also be considered. A user model and dialog history are helpful means in resolving error reasons. Imagine the situation where a system detects a possible error because the recognized utterance did not match the current dialog state. If the confidence level is high, the presence of a semantic error can be assumed. This means that the user is either disoriented or simply used the wrong commands. Because of the need for application specific information, error reasoning is often a highly application dependent task.

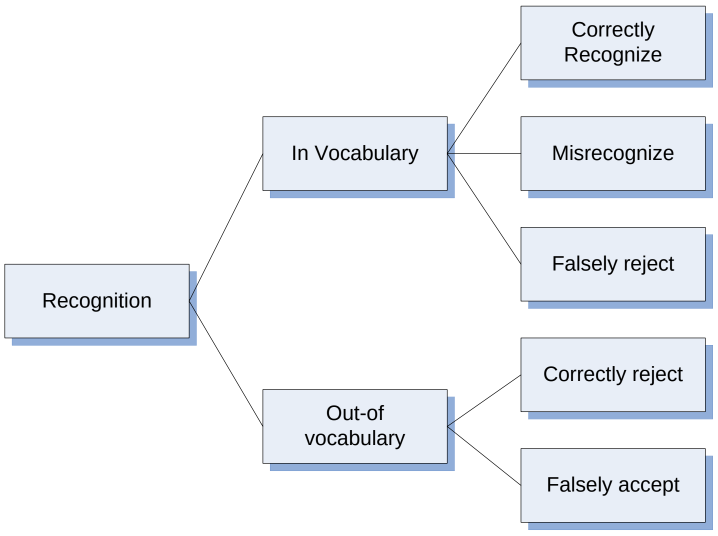

Figure 1: Categorization of Recognition Errors

1.3 Error Correction

Error correction takes place after the cause of the error has been deduced. The system decides how to handle the error, by selecting a suitable error correction strategy. Error correction should be considered carefully, since too liberally used error management can lead to inefficient and unusable interface. In particularly, if EXPLICIT CONFIRMATION is used, or the application has real-time needs, the interaction may become too clumsy. Another important aspect of error correction is to avoid error spirals, in which an error correction procedure leads to new errors, making the interaction an endless loop of errors. Hence, badly chosen or even missing error correction prevents the user from entering her data at all.

Different dialog management strategies may also be used as a means for error handling. As pointed out in [13], e.g. MIXED INITIATIVE may fall back into FORM FILLING if the user fails to enter the data.

When the actual error correction takes place, the system initiates an error correction dialog, such as a ESCALATING DETAIL or GLOBAL ERROR CORRECTION. It is important to realize, that error correction is a dialog itself, and carried out like any other dialog, and more errors can occur during the correction dialog.

The following phase model was introduced by Turunen [14]. It is based on the model of Luperfoy and Duff [6].

1. **Error detection** Either the system or the user detect an error and the system categorizes it as described in section 1.1.

2. **Diagnosis of cause** The system analyzes the causes leading to the error so that it can better correct the error and ideally prevent further errors. Without knowing these causes, both the system and the user may repeat the same errors, leading to an error spiral.

3. **Planning a correction** The system selects a suitable error correction strategy to handle the error. The estimated error correction costs and the errors costs should be compared to decide if error correction is justified. If the error correction costs precede the error costs,

error correction should be omitted. An automated initiation of an error correction dialog should be avoided, since this can lead to inefficient and unusable interface.

4. **Executing the correction** The system initiates an error correction dialog, such as Selection from a List, to correct the error.

5. **Informing the user about the error correction** Error correction should be noticed by the user. The system informs the user about the error correction, if necessary, and about the reasons that led to it. The system also informs the user about what is going to happen next.

6. **Closure and return to the primary dialog** After error correction the system chooses how to return to the primary dialog. This can lead to a totally new dialog state, since the error or its correction may have changed the context.

7. **Preventing errors** In some cases, the system can adapt dynamically to the user and prevent further errors from occurring. Examples include the change from the Mixed Initiative dialog strategy to e.g Form Filling as mentioned above. Another strategy to prevent errors is to offer more help to the user. These are examples of dynamic error prevention. Error prevention can also take place in the design phase of an application. For example, the vocabulary should be acoustically different to reduce recognition errors.

Especially the last point of preventing errors is important. Junqua says in [4]:

> The cost of an error is application dependent, and so is the error correction technology used. Errors have serious consequences because they can destroy the user's mental model.

Consequences of an error are e.g. the navigation to some place unintended or the triggering of unwanted actions. This increases the lost in space problem and the feeling of not being in control. This has to be considered when planning an error correction.

2 Additional Background Information

Figure 2 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 for presenting it to the user.

The main problem with these roles is, that all actors have a direct relation to the voice application, but all have different levels of understanding.

The VUI designer knows all details of the application. In order to design the application she also has to think like an experienced user and like a novice. Since experienced users also have a detailed understanding of the application, it is relatively easy for the designer to take the experienced user's view. The role of the novice is more difficult to understand for the designer. She has to provide the user with the means to learn the concepts and structure of the application to become an experienced user. Additionally the designer has to take care that both, experienced users and novices find a comfortable way to interact with the application. Hence, we have a set of core forces for the development process that are shared among all patterns and reflected in the patterns described below.

Some patterns feature an example. The examples are taken from a telephone banking application. In this example, a user calls her bank and is served by an automated telephone system, where she can hear her e.g. current account balance or transfer money.

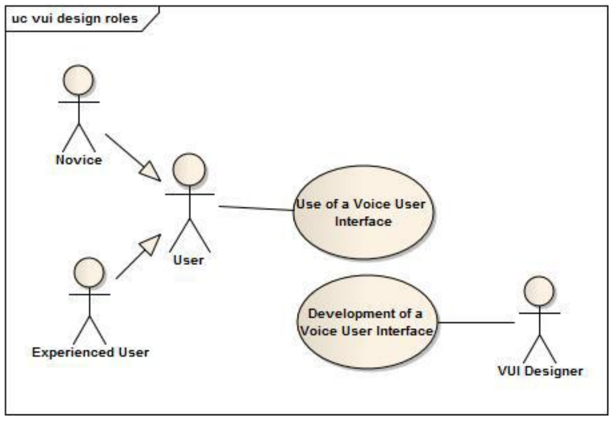
uc vui design roles
Novice
User
Experienced User
Use of a Voice User Interface
Development of a Voice User Interface
VUI Designer

Figure 2: Simplified view on the roles involved in VUI design

3 Error Management Patterns

In this section the pattern language for error management in voice user interfaces is presented. The language comprises two parts, patterns for error recovery and patterns to prevent errors that are presented in the following sections.

3.1 Error Recovery

After an error has been detected the application initiates an error recovery strategy. The following patterns provide the means to recover from an error with respect to the cause of the error.

Escalating Detail

Intent

Provide an error recovery strategy when the speech 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*, because no speech was detected at all.

Problem

How to guide the user in an error situation with the right amount of information?

Forces

- In case of a missing input, it is likely that users did not listen carefully and do not know what utterances they may perform. They need more guidance to provide 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 want to have a faster system response.
- Users do not know 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 handle those errors appropriately.
- Low recognition performance and background noise result in a misinterpreted type of error.
- An error can only be detected, if the recognition result is not allowed in the current context.

Solution

When a user's speech is not recognized correctly or the user did not speak at all, provide responses that give increasing amounts of detail with each subsequent error with respect to the source of the error. These responses should be designed to help the user in making less ambiguous verbal responses. To implement this strategy, consider the following:

1. Write handlers for different types of errors.
2. Use a counter for the occurrences and levels of the affected type in a row.
3. Write prompts to increase the amount of provided details of help provided at each level.
4. 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.

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.

Related Patterns

DECORATION can be used to reuse the initial prompt. RAPID REPROMPT provides a faster way of entering the data.

Example

In our banking scenario, a caller is about to transfer money. At this step she has to provide the amount. The dialog should behave as follows:

System: How much do you want to transfer?
User: *no input*
System: I did not hear you. Please, name the amount you want to transfer
User: *unrecognized input*
System: I did not understand you. Please, name the amount you want to transfer
User: *unrecognized input*
System: Sorry, I still did not understand you. You have to tell the amount of money you want to transfer. Please, use only a number. Please, name the amount you want to transfer.
User: 500$

Also Known As

Yankelovich et al. [16] and Weinschenk et al. [15] name this pattern PROGRESSIVE PROMPTING.

Variances

RAPID REPROMPT

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, because no speech was detected at all. The structure of the dialogs are obvious to the user or with repeat users.

Solution

The solution is similar to ESCALATING DETAIL and provides more detailed information detailed responses at subsequent errors of the same type. However, the first response is very short assuming that the user knows what to say and simly needs another try. The following response messages are exactly the same as in ESCALATING DETAIL. To implement this strategy, consider the following:

1. Write handlers and use corresponding counters for different types of errors as for ESCALATING DETAIL

2. Use a short prompt such as "I'm sorry" or "What was that?" at the first level as a notification of the error, thus enabling the user for a rapid way of another try.

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.
- ☺ Preferred by users if all variables are controlled and when only the prompt wording is varied
- ☹ Does not provide users with the detailed information right away
- ☹ Users who are unsure of what to say may need an extra step to recover from an error
- ☹ Difficult to find the right amount of help at each level.

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

Example

Think of the same scenario as described in the example for Escalating Detail.

Here, the dialog should behave as follows:

System: How much do you want to transfer?
User: *no input*
System: I'm sorry?
User: *unrecognized input*
System: What was that?
User: *unrecognized input*
System: I did not understand you. Please, name the amount you want to transfer
User: 500$

Known Uses

A survey by Nuance compared the two strategies. In the test scenario, all variables are controlled and only the wording was varied. The results showed, that Rapid Reprompt is preferred by approx. 80% of the users [2].

Global Error Correction

Intent

Keep the user although a lot of recognition errors occur.

Context

The recognition accuracy is poor or the caller population varies widely, e.g. the application has to support native and non-native speakers.

Problem

How to enable the user to provide her data if the overall error rate is high?

Forces

- Users want to achieve their goal for calling
- Too many recognition errors make it impossible for users to achieve their goal for calling
- Correcting errors is time consuming and annoying
- Hang-up rates are very high after a few rejects
- Live agents are not available for all applications
- Handicapped users must be able to use the application

Solution

The solution takes into account the respect to the global amount of errors of this call but also to the subsequent error count for entering a data item. If one of these counters is greater than a predefined threshold, transfer the user directly to an agent or name other means of achieving her goal for calling. To implement this strategy, consider the following:

1. Establish three thresholds:
   (a) the maximum number of successive errors in a single dialog state (typically 3)
   (b) the maximum number of errors counted globally
   (c) the maximum number of global disconfirmations. This is the responses of "No" in confirmation states. (typically 2)
2. Prepare a counter for all of the above mentioned error types in an application.
3. Increment the counters accordingly
4. When one of the thresholds is reached, send callers directly to an agent if one is available or name other means of achieving their goal. For example, name the presence of a web site but do not advertise for it.

Consequences

☺ Users reach their goal for calling

☺ Users are not annoyed since they are not forced to keep on trying until they give up.

☹ Additional effort is needed to correctly count the errors

☹ Does not leverage the provider from maintaining a call center

☹ Users may loose their trust in a system if they are transferred to an agent each time they use the application

☹ Users may use this feature to fool the application.

Related Patterns

Explicit Confirmation is the basis for the threshold of maximum number of disconfirmations. Escalating Detail or Rapid Reprompt can be used to handle successive errors in a dialog state.

Example

Imagine a user who spent a lot of time to enter their amount for a transfer and who is now at a confirmation step to initiate the transfer. A sample dialog may look as follows.

System: Do you want to transfer 500$ to account number 12345678?
User: No
System: Sorry. Which part did I get wrong?
User: The account number
System: What is the correct account number, then?
User: 13345678
System: Do you want to transfer 500$ to account number 12245678?
User: No
System: Sorry. You will be transferred to a call center agent for clarification.
<*transfer user*>

Known Uses

This pattern is implemented in nearly all telephony applications in Germany by law. As a result an advice was given to fool the application by not using words to enter the data, but to sing e.g. *lalalalala...* in order to get transferred to a call center agent [7].

SPELLING

Intent

Recognize the user's input although all attempts of the user to deliver her data fail due to recognition errors.

Context

Users must enter their data and have no means to use alternative input methods or they are allowed to use an unconstrained vocabulary.

Problem

How to enable the user with means to enter her data although the recognition errors are high in s specific dialog step?

Forces

- Users must enter their data
- Users prefer to deliver their data using unconstrained spoken language
- Users want a fast way to deliver their data
- A limited vocabulary allows for a high recognition accuracy
- Unconstrained (or dictation) grammars have a low recognition accuracy and it can still happen that a word is not in the dictionary (OOV problem)
- Single letters are too short for good recognition results
- Users may not know established alphabets like the NATO Alphabet
- Users might have difficulties in telling the correction if the recognition fails at all

Solution

When a user's utterance is not recognized because it is pronounced differently (e.g. due to a dialect) or the user used a word that is not in the dictionary enable the user with other means to enter her data by voice, e.g. let her spell the misrecognized word. To implement this strategy, consider the following:

1. Identify the misrecognized word or let the user identify the wrong word
2. Ask the user to spell the word
3. Accept established phonetic alphabets to allow for higher recognition accuracy

Consequences

☺ The user can enter her data although all other approaches fail
☹ Interrupts from the usual flow
☹ Time intensive to enter a word letter by letter
☹ Might fail as well since there are already difficulties with recognizing this user

Related Patterns

The use ESCALATING DETAIL or RAPID REPROMPT should be considered before this pattern is used as a final error handling strategy. SELECTION FROM A LIST provides an alternative way of entering the data if there are several recognition hypotheses.

Example

Imagine a user of a banking application who wants to enter the reason for transfer. Once the system asks the user to spell the reason for transfer, she can use the ICAO spelling alphabet [8]. The dialog should behave as follows:

System: What is the reason for transfer?
User: Present
System: I did not understand you. Please, name the reason for transfer
User: Present
System: Sorry, I still did not understand you. Please, spell the reason for transfer
User: Papa Romeo Echo Sierra Echo November Tango

Known Uses

The Windows 7 speech recognition system uses this technology to allow for an error correction using voice.

Selection from a List

Intent

Allow the user to enter her data if there are multiple recognition results with a similar confidence score

Context

Users enter their data and the recognizer returns with a list of different recognition hypothesis. None of these can be used to identify the user's utterance with a satisfactory confidence for sure.

Problem

How to accept the user's input if there is no concise recognition hypothesis?

Forces

- A user input may require words that sound similar and are hard to distinguish by the recognizer
- Repetions of similar sounding input may result in repetetive errors of the same kind or in iterative attempts to enter the data
- The intended input may not be in the list of recognition hypotheses
- Not all recognizers are able to deliver multiple recognition hypothesis
- A dynamic list may have items that are not easily distinguishable by a recognizer
- A small vocabulary, such as *yes* and *no* can be recognized with an accuracy close to 100%

Solution

The solution exploits the hypothesis list of the recognizer to ask the user for confirmation of each of them using simple yes-no questions. If the intended option is not among them, give the user another try to enter her data. To implement this strategy, consider the following:

1. Evaluate the confidence score of the recognized utterance.
2. If the confidence score is low, ask the user for clarification by prompting the different possibilities as a list.
3. Accept only *yes* and *no* as answers to select from the list.
4. If the last item was rejected, reprompt the user to enter the data anew.

Consequences

☺ The user can deliver the data without the need to repeat the whole utterance.

☺ Intuitive way of selecting the right data, similar to what humans do in their conversation when they ask for clarification.

☺ An erroneous input is corrected by a mechanisms with a higher recognition accuracy.

☹ An additional step is needed to enter the data.

☹ Not applicable if there are too many items in the list.

☹ The force that some recognizers do not deliver multiple recognition hypothesis remains unsolved.

Related Patterns

Related Patterns

The use Escalating Detail or Rapid Reprompt should be considered before this pattern is used as a final error handling strategy. SPELLING provides an alternative way of entering the data if the recognizer did not provide a recognition hypothesis or is not able to do that. THREE-TIERED CONFIDENCE provides a solution if the recognizer comes up with a single recognition hypothesis and a low confidence score.

Example

Imagine a user of a banking application who is about to select the service she wants to use during this phone conversation. The dialog should behave as follows:

System: Welcome to ABC bank. How may I help you?
User: I want to transfer money
System: Sorry, I am not sure if I got you right. Do you want to lend money?
User: No
System: Do you want to transfer money?
User: Yes

Three-Tiered-Confidence

Intent

Keep the confidence in recognized utterances high.

Context

The recognizer returns with a recognition hypothesis that has a low confidence score.

Problem

How to get the right data for a recognition hypothesis with a low confidence score?

Forces

- Not all recognizers are able to deliver confidence scores
- Confirmation is not always needed, especially for data with a low critical relevance or which can be corrected easily in subsequent dialog steps
- The recognizer's hypothesis may be correct although the confidence score is low
- Users need a fast way to enter their data
- Confirmation is time consuming
- Users prefer applications with a human-like dialog flow

Solution

The solution imitates the behaviour of humans in a dialog who accept the data if the other party said something that was understood without any doubt but who ask for confirmation if there was some uncertainty about the heard words. If the other party was not able to understand what was said at all, humans simply ask again. Therefore, establish three tiers for each of these cases and assign an error recovery strategy for each tier. Use the confidence score to select the relevant tier. To implement this strategy, consider the following:

1. Define three confidence level thresholds (tiers) low, medium and high.
2. If the confidence is high, do not confirm.
3. If the confidence is medium, confirm.
4. If the confidence is low, reject.

Consequences

☺ Confirmation is only performed when it it needed.
☺ Does not lengthen the conversation unnecessarily.
☹ May lead to an explicit confirmation of all inputs, if the overall recognition accuracy is low.
☹ Not applicable if there is critical information that must be confirmed.

Related Patterns

Selection from a List may be used to offer more guidance to the user. Explicit Confirmation can be used for confirmation.

Example

A user tries wants to transfer money and has to select the appropriate service. With Three-Tiered-Confidence the dialog could behave as follows:

System: Welcome to ABC bank. How may I help you?

User: I want to transfer money
<*Confidence score was low*>
System: Sorry, I did not understand you. Which service do you want?
User: Transfer money
<*Confidence score was medium*>
System: Do you want to transfer money?
User: Yes
System: How much do you want to transfer?
User: 10 dollars
<*Confidence score was high*>
System: What is the reason for transfer?
User: ...

Known Uses

San-Segundo et al. describe the usage of this pattern in [10] in a railway information system. They use the confidence scores to select an appropiate confiramtion strategy. They use actually four levels but discover that the fourth level that was introduced did not make much sense.

Cohen descibes THREE-TIERED CONFIDENCE in [2] as a strategy to deal with errors.

3.2 Error Prevention

Instant feedback is a vital component of every user interface. It serves to update the user's mental image of the state of the application as shown in figure 3. Since speech is invisible, the

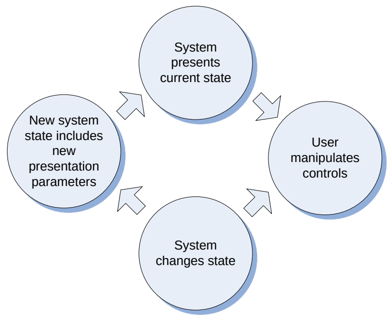
System presents current state
User manipulates controls
System changes state
New system state includes new presentation parameters

Figure 3: The feedback loop according to Ballentine [1]

absence of feedback leads to errors of all kinds but the presence of feedback leads to [1]

- less errors,
- the prevention of state errors,
- removal of time from the interface and
- allows users to understand and incorporate delays

These aspects are the core of the following patterns.

Immediate Feedback

Intent

Provide feedback about an accepted input.

Context

The user made a request resulting in an uncontrollable and variable delay before the next prompt is presented.

Problem

How to manage uncontrollable and variable delays?

Forces

- Speech is invisible: Users do not know if the system accepted their request
- Users do not know what is going on behind the *scenes*.
- An announcement of a duration with an immediate continuation may destroy the user's mental model
- Long pauses may cause the user to believe that the system is inactive and will hang up

Solution

After the user entered her data additional processing may be necessary before the dialog can continue. The duration of these processing steps is not always known in advance. Hence, the system must notify the user that the input before the processing step to indicate a pause of uncertain length. To implement this strategy, consider the following:

1. Request information from the user as usual.
2. Once, the user made her request, signal the user immediately that her request was understood by either
   (a) playing back a beep,
   (b) outputting a short prompt, like *Thank you*
   (c) or by giving explicit information that the following request may take some time.
3. After that confirmation, initiate the request with the uncontrollable delay.
4. Play back some good music to signal the user that the system is still active.
5. Continue with the result of the request.

Consequences

☺ The user knows that her request was accepted by the system.
☺ The user gets an understanding of what is going on.
☹ The term *Thank you* does not sound as the logical continuation of a conversation.

Example

In our banking scenario, the user enters her account number to hear the current account balance. The balance is retrieved from a host system. Additionally, the system must confirm that the current user is allowed to retrieve the data. In a demo setup the latency varied between 0.5 seconds and 20 seconds.

Computer: What's your account number?
User: 133-552-01
Computer: Thank you
<*Uncontrollable and variable delay*>
Computer: The balance…

Explicit Confirmation

Intent

Ask for the user's confirmation about the correctness of the recognized input.

Context

The user provided some data initiating an action. The consequences of an error are costly or impossible to undo.

Problem

How to be sure that the user provided the right information?

Forces

- Users forget the exact wording of the choices by the time
- Experienced users know what to say and need a fast way to enter the data
- Speech is invisible: Users cannot see what the system understood
- Excessive use of confirmations let the dialog appear rigid and unnatural
- Users prefer applications with a human-like dialog flow
- Processing steps that can not be easily undone or that can have serious consequences for the user require a perfect recognition rate
- Speech recognition technology is error prone

Solution

The invisible and transient nature of voice based interfaces require an active feedback of the system about the recognized data, which may actually be different to what the user said. In order to be sure that these two match, ask explicitly for confirmation. If the data is confirmed by the user continue processing the data, otherwise give the user another try to input her data. To implement this strategy, consider the following:

1. The user enters some data.
2. Play back the recognized data and ask explicitly for confirmation, e.g.

   Computer: What's the departure city?

   User: Frankfurt.

   Computer: Frankfurt. Is that correct?

   User: Yes.
3. If the use confirms the data, continue with the next field.
4. If the user rejects the data, reprompt the user for the data.

Consequences

☺ Danger of triggering unwanted and not recoverable errors becomes low.

☹ Dialog becomes lengthy and rigid.

☹ Excessive use may hinder users from using the application at all.

Related Patterns

Usually, feedback falls along a continuum from explicit to implicit. Consider using IMPLICIT CONFIRMATION if the consequences for the user are not high in case of an error. This pattern can be used to validate the data that was collected by FORM FILLING.

Variances

Group the confirmation items to no more than four items before confirmation. If the user rejects the data, ask for the field that was wrong, e.g.

Computer: Now let me confirm that: Hamburg to Darmstadt on March fifth. Is that correct?
Computer: Which part did I get wrong?
User: The departure city

Consequences

- ☺ Grouping leads to more fluid and efficient dialogs.
- ☹ An additional dialog step is needed to determine the wrong data.

Implicit Confirmation

Intent

Ensure the correctness of the entered data.

Context

An erroroneous input has no serious consequences for the user or can easily be undone. The dialog aims at mimicing a conversation between two humans, thus having an unconstrained grammar. The user did not provide all data and there is a semantic relationship between the entered data and the next item to collect.

Problem

How to increase the confidence of the recognized information?

Forces

- Flexible grammars tend to increase error rates
- Dialogs should be efficient.
- Combinations of input fields grow by a factorial order.
- It is possible to express the same thing in endless ways.
- Not all speech snippets can be pasted together to create a meaningful sentence.
- Speech is invisible: Users cannot see what the system understood
- Excessive use of confirmations let the dialog appear rigid and unnatural
- Users prefer applications with a human-like dialog flow
- Processing steps that can not be easily undone or that can have serious consequences for the user require a perfect recognition rate
- Speech recognition technology is error prone

Solution

The solution mimics the behaviour of the human to insure the correctness of the entered data. Therefore, integrate the entered data into the output of the subsequent output that asks for the next piece of information. The user can now easily be sure that the system understood her correctly. If the user then continues entering the next data, the previously entered data is implicitly confirmed. Alternatively, the user can reject the data by simply correcting it when the system asks for the new data. To implement this strategy, consider the following:

1. Once the user provided data, integrate the entered data into the prompt for the missing data.

2. Expect a user initiated correction of the previous data in the next field as well as the missing data.

   Computer: What's the departure city?

   User: Frankfurt.

   Computer: When do you want to leave from Hamburg?

   User: No. I want to leave from Frankfurt

   Computer: Sorry about that. When do you want to leave from Frankfurt?

Consequences

☺ Dialog becomes more natural.

- ☺ Dialogs are more efficient and shorter
- ☹ Can't be used to summarize content.
- ☺ More lightweight than Explicit Confirmation.
- ☹ Require bandwidth -do not overuse!
- ☹ Cannot be applied if there is only a single data to enter or for the last piece of information if there is more data.

Related Patterns

Usually, feedback falls along a continuum from explicit to implicit. More lightweight than Explicit Confirmation. Highly suitable for Mixed Initiative dialogs but also for Form Filling

Known Uses

San-Segundo et al. use this pattern in a railway information system [10]. They observed that approx. 60% of the confirmations were implict, the rest was explicit.

4 Conclusion

Speech technology is still error prone. Hence, designers and developers of voice user interfaces must be prepared to integrate error management as a fundamental concept into their solutions. In this paper we extended the pattern language for voice user interface design that was presented e.g. in [11] with a focus on error management techniques. The patterns cover aspects such as error recovery and error prevention on the basis of a proven error categorization and error handling theory.

In this paper, six patterns for error recovery have been presented. Escalating Detail and Rapid Reprompt enable the handling of Level 0 to Level 3 errors. The errors are detected and corrected with limited respect to the cause of the error. Global Error Correction also handles Level 7 errors if the overall disfirmation of the user is high. The appearance of errors is expected to be due to a bad recognition performance.

Spelling, Selection from a List and Three-Tiered Confidence are suitable to handle Level 1 errors. They also provide the user with a way to correct her input.

A better approach to deal with errors is to prevent them. The pattern language introduces three patterns with this goal. Immediate Feedback shows a way how to deal with uncontrollable, variable delays. Explicit Confirmation and Implicit Confirmation provide feedback about the entered data which is an important issue due to the invisible nature of voice based interfaces.

Summing up, these patterns enable developers and designers of voice based user interfaces with a solid fundament to deal with errors.

References

[1] Bruce Ballentine. *It's Better to Be a Good Machine Than a Bad Person: Speech Recognition and Other Exotic User Interfaces at the Twilight of the Jetsonian Age.* ICMI Press, March 2007.

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

[3] D. Duff, B. Gates, and S. LuperFoy. An architecture for spoken dialogue management. In *Proceedings of the 1996 International Conference on Speech and Language Processing (ICSLP)*, 1996.

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

[5] C. Kamm. *Voice Communication Between Humans and Machines*, chapter User Interfaces for Voice Applications, pages 422–442. National Academy Press, Washington D.C., 1994.

[6] S. Luperfoy and D. Duff. Disco: A four-step dialogue recovery program. In *Proceedings of the AAAI workshop on Detection, Repair, and Prevention of Human-Machine Miscommunication*, 1996.

[7] Matthias Münch and Georg Schnurer. Hotline-Sprachcomputer durch Singen überrumpeln. `http://www.berlin-bookmarks.de/nw/article/Vermischtes/1195503795\_Hotline-Sprachcomputer\_durch\_Singen\_ueberrumpeln.html`, nov 2007. Last accessed on 02/15/2010.

[8] L. J. Rose. Aviation's ABC: The development of the ICAO spelling alphabet. ICAO Bulletin, 1956.

[9] Hirohiko Sagawa, Teruko Mitamura, and Eric Nyberg. Correction grammars for error handling in a speech dialog system. In *HLT-NAACL '04: Proceedings of HLT-NAACL 2004: Short Papers on XX*, pages 61–64, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[10] R. San-Segundo, J.M. Montero, J. Ferreiros, R. Córdoba, and J.M. Pardo. Designing confirmation mechanisms and error recover techniques in a railway information system for spanish. In *Proceedings of the Second SIGdial Workshop on Discourse and Dialogue*, pages 1–4, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[11] Dirk Schnelle. *Context Aware Voice User Interfaces for Workflow Support.* PhD thesis, Technische Universität Darmstadt, 2008.

[12] Dirk Schnelle and Fernando Lyardet. Voice User Interface Designt Patterns. In *EuroPLoP 2006 Conference Proceedings*, 2006.

[13] Dirk Schnelle, Fernando Lyardet, and Tao Wei. Audio Navigation Patterns. In *Proceedings of EuroPLoP 2005*, pages 237–260, July 2005.

[14] Markku Turunen and Jaakko Hakulinen. Agent-based error handling in spoken dialogue systems. In *Proceedings of the Eurospeech 2001*, pages 2189–2192, 2001.

[15] Susan Weinschenk and Dean T. Barker. *Designing effective speech interfaces.* John Wiley & Sons, Inc., New York, NY, USA, 2000.

[16] Nicole Yankelovich, Gina-Anne Levow, and Matt Marx. Designing speechacts: issues in speech user interfaces. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 369–376, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.