Designing Common Functions for Speech-Only User Interfaces: Rationales, Sample Dialogs, Potential Uses for Event Counting, and Sample Grammars

TR 29.3287

February 22, 2000

James R. Lewis
Joseph E. Simone
Mariusz Bogacz

Speech Development and Customization

West Palm Beach, Florida

Abstract

This document describes six functions that every speech-only user interface should include (help, repeat, spoke-too-soon, cancel/backup, start-over, goodbye) and one that many clients will want to have in their system (operator). The document includes descriptions and sample grammars that should help developers design usable and consistent user interfaces. It also contains analyses of how to use event counts associated with these functions to control dialogs in the user interface. One interesting result of these analyses is that, with some exceptions, there doesn't seem to be much reason to count these events as a component of mechanisms for controlling dialogs.

ITIRC Keywords

Speech recognition
Telephony
Speech user interface design

Contents

Introduction

The purpose of this document is to provide guidance to developers of IBM speech-only user interfaces (such as telephony user interfaces) for the design of certain functions that should be common to the user interface of most systems, and should be available at every user turn in the user-system dialog. Those functions are:

- Help
- Repeat
- Spoke-too-soon
- Cancel/Backup
- Start-over
- Operator
- Goodbye

It is possible to categorize the functions into three broad groups:

- Help
- Remain in system (repeat, spoke-too-soon, cancel/backup, start over)
- Leave system (operator/disconnect, goodbye)

The help style we are promoting is implicit, self-revealing help. It has a number of desirable properties, including no requirement for an explicit help mode (or grammar, although it does not preclude the use of a help grammar). It supports efficient use by experts, but its self-revealing property supports ease-of-use by novices. If the help is not helpful to a user, the system moves the user to an operator (if available).

The four functions that remain in the system differ primarily in the extent that they move the user back through the system. Commands from the repeat grammar and spoke-too-soon errors result in the repetition of the most recent prompt. Cancel/Backup moves the user to the prompt that preceded the most recent prompt. Start-over moves the user back to the main menu (or, if the system has no menu, to the initial explicit prompt).

The two functions that leave the system are operator and goodbye. The operator function lets the user get out of the recognition system and talk to an operator (if available). The goodbye function results in the presentation of a closing prompt followed by disconnection.

Our approach is based on the guidelines of Balentine and Morgan (1999), but in this report we extend the Balentine and Morgan guidelines by providing for each function:

- a rationale for why it is important to the user interface
- sample dialogs relevant to the types of systems we expect to build
- an analysis of what to count and how to use those counts in managing the use of the function
- sample grammars (where appropriate) to support the functions.

The analysis of what to count and how to use those counts is of particular interest because it has not, to our knowledge, been addressed in any other work. Balentine and Morgan (1999) typically recommend against using event counts to control the user interface. Our analyses demonstrate that this is correct in most cases, but not in some.

Help (Implicit, Self-Revealing Help Style)

Rationale

A common characteristic of speech-only user interfaces is that they have no documentation. In many cases, they also need to be able to support use by both novices and experts. For both classes of users, it is important to force users to hear no more than they need to hear, and require them to say no more than they need to say. The ideal way to accomplish this is to make the interface self-revealing through the appropriate use of recognition windows (Balentine and Morgan, 1999), which depends on appropriate usage of silence timeouts[1]. It is possible that a user might experience momentary confusion or distraction, which would lead them to explicitly request help. If the system has been designed to be self-revealing, then these explicit requests for help could receive the same treatment as a silence timeout or any out-of-grammar utterance in the recognition window, allowing use of the same code and prompts. This is a style of help referred to as 'implicit' because the system never enters an explicit help mode. The theory is that implicit help provides more of a sense of moving forward (better, more satisfying UI) and is simpler to code than explicit help.

Sample Dialog

System:	Welcome to the IBM automated directory dialer. […]
User:	<rapid silence timeout or any out-of-grammar utterance or anything from the help grammar>
System:	You can call any IBM employee by speaking the employee's name and location. […]
User:	<rapid silence timeout or any out-of-grammar utterance or anything from the help grammar>
System:	You can hang up or say "Help" at any time. […]
User:	Help. <or rapid silence timeout or any out-of-grammar utterance>
System:	Please state the desired name and location. [!!!]
User:	I need more help. <or full silence timeout or any out-of-grammar utterance>
System:	For example, to call Joe Smith in Kansas City, say, "Joe Smith in Kansas City". [!!!]
User:	I still don't get it. <or full silence timeout or any out-of-grammar utterance>
System:	Sorry for the confusion. Transferring to an operator.
User:	Thanks.

Note the self-revealing nature of the dialog, and the way it works whether the cue to the system is a silence timeout, an utterance explicitly recognized as from a Help grammar, or any out-of-grammar utterance. Also, note the pattern for the introduction (first four system prompts), which has the general sequence Welcome-Purpose-MoreInfo, followed by an ExplicitPrompt. In most cases the explicit prompt at the end of the introduction will get the user on the path to successful use of the system. If user inputs after that explicit prompt are (1) in the Help grammar, (2) silence timeouts, or (3) out-of-grammar, then the system falls through the general sequence of ProvideExample-Bailout. Note that the user never hears

[1] The symbol for a recognition window in this paper's dialogs is […]. If the recognition window should be signaled with a beep, indicating to the user a requirement for user input at that point, the symbol is [!!!]. One key difference between […] and [!!!] is that the silence timeout for […] should be quite short, between .5 and 2.0 seconds, depending on the linguistic context in which the window appears. The silence timeout for [!!!] should last from 3-10 seconds, depending on the complexity of the task and the expected length of user utterance.

the same prompt simply repeated in this sequence, so the system always seems responsive to the user, always moving forward[2]. This help style should promote efficient use by most users. One potential drawback is that it does not necessarily reveal to the user the existence of the other common functions described later in this document (repeat, backup/cancel, start over, operator and goodbye). If deemed necessary for a particular application, the system's dialog design could reveal these features just before bailing out. Consider the end of the sample dialog above:

System: For example, to call Joe Smith in Kansas City, say, "Joe Smith in Kansas City". [!!!]

User: I still don't get it. <or full silence timeout or any out-of-grammar utterance>

System: Would you like to hear a list of the things you can say at any time?

User: Yes.

System: At any time, you can say:

Repeat […]
Backup […]
Start over […]
Operator […]
Goodbye […]

Please say one of these options, or state the desired name and location. [!!!]

User: Operator.

System: Transferring to an operator.

Potential Uses for Counting 'Helps'

It doesn't cost much to record the number of explicit requests for help, but it also isn't clear how to use these counts, as long as the system is self-revealing with a smooth bail-out, as described above. For controlling the bailout prompt, it would probably make sense to keep track of the return codes that come back as the user goes through the help-to-bailout sequence rather than counting helps. Some applicable return codes and bailout prompts are those associated with:

In grammar with high confidence	Sorry for the confusion. Transferring to operator.
In grammar with low confidence	Sorry for the confusion. Transferring to operator.
Out of grammar	Sorry for the confusion. Transferring to operator.
Silence timeout	Sorry, but I can't hear you. Transferring to operator.

The only time that the bailout prompt for silence timeout is appropriate is if all of the opportunities for user speech have resulted in silence timeout. Otherwise, the 'sorry for the confusion' prompt is more appropriate. Note that if the system doesn't have the capability to transfer to an operator or the designer decides that transferring to an operator is inappropriate, the designer can replace the phrase 'Transferring to operator' with 'Disconnecting … please call back and try again', or can adopt a strategy of repeating the last prompt in the sequence some programmatically-determined number of times before disconnecting.

[2] If user testing indicates that two chances at providing on-task input is insufficient, then it might be reasonable to allow the system to produce the final prompt more than once. This should, however, be the exception rather than the rule.

A Sample ‘Help’ Grammar

```
<help> =
    <preface>? some? <help-noun> please? |
    <preface> (some | more) help please? |
    some <help-noun> please? |
    <help-noun> please? |
    what can i do |
    what can i say |
    tell me how i can use this system |
    can you please <help-verb> me |
    i really need some <help-noun> badly? |
    what does that mean |
    i don't understand that |
    that makes no sense (to me)? |
    that makes no sense (at all)? |
    (could you)? please? repeat that? |
    (can you)? please? repeat that? |
    huh i didn't get that |
    what |
    i <do-not> understand |
    i think i need some <help-noun> |
    i <do-not> know what to say |
    i <do-not> know what to do |
    can i start over |
    how do i start over |
    what (do i do)? now |
    what else (can i do)? |
    what things can i ask for |
    what things do you know about |
    what do you know about? |
    what can i ask for? |
    what can i do here? |
    how does this work |
    help me |
    how about helping me (with this)? |
    please help me
    .

<help-noun> =
    help |
    assistance |
    guidance
    .

<help-verb> =
    help |
    assist |
    guide
    .

<preface> =
    <preface-norm> |
    <preface-ing>
    .
```

```
<preface-norm> =
    i would like                     |
    i would like to <acquire>        |
    i'd like                         |
    i'd like to <acquire>            |
    i want                           |
    i want to <acquire>              |
    i need                           |
    i need to <acquire>              |
    please? get me                   |
    can you please? get me           |
    please? give me                  |
    can you please? give me          |
    please? provide me with          |
    can you please? provide me with |
    can i please? <acquire>          |
    can i <acquire>                  |
    may i please? <acquire>          |
    i wish to <acquire>
    .

<preface-ing> =
    how about giving me              |
    I feel like
    .

<acquire> =
    have  |
    get
    .

<do-not> = do not                    |
    don’t
    .
```

Functions That Leave the User in the System

Repeat

Rationale

In a speech-only user interface, the user needs to hear and carefully attend to the output of the system because system output is temporal and ephemeral. Careful design of prompts and feedback with regard to clarity and prosody can help, but if a user didn't get all of the information in the system message, either from being distracted or due to the amount of information in the system message, then the user might well ask the system to repeat what it just played.

In general, if asked to repeat, the system should repeat all of the system messages and prompts from the end of the previous explicit prompt to the current explicit prompt, as shown in the sample dialog. In the example, the user is more likely to need to have the conversion rate repeated than to need to have just the current explicit prompt ("Would you like to do another conversion?) repeated.

Sample Dialog

System: The conversion rate from French francs to Australian dollars is 1.5678 dollars per franc. […] Would you like to do another conversion? [!!!]

User: Please repeat that.

System: The conversion rate from French francs to Australian dollars is 1.5678 dollars per franc. […] Would you like to do another conversion? [!!!]

Potential Uses for Counting 'Repeats'

Because it doesn't cost much to count things, it's reasonable to count repeats. If the user is in an attention-demanding environment or is trying to write down or memorize the system message, he or she might well ask for several repetitions of the message, so typically the system should simply provide them. There might be cases in which it would make sense to escalate the prompts on repeated repetition – making them less telegraphic, for example. Certainly, the system shouldn't permit infinite requests for repetition. As a rule of thumb, we'd recommend limiting repetition sequences to no more than five consecutive repetitions. The system should, however, allow the designer to override any globally set repetition limit so he or she can tailor the repetition limit to be appropriate for an especially long (potentially requiring a larger repetition limit) or short (potentially requiring a smaller repetition limit) system message.

A Sample 'Repeat' Grammar

```
<repeat> =
    <sorry> |
    <sorry>? <missed-it> |
    <sorry>? <missed-it>? <repeat-it>
    .

<repeat-it> =
    <request>? give <it-that> to me    <again> |
    <request>? run  <it-that> by me    <again> |
    <request>? say  <it-that> (to me)? <again> |
    <request>? say  <it-that> over     <again>? |
    <request>? repeat <last-prompt> <again>? please? |
    <request2>? have <it-that> repeated <again>? |
    <again> please?
    .

<missed-it> =
    i <didnot> quite? <get> (you | <last-prompt>)? |
    i missed <last-prompt> |
    i am not following     (you | <last-prompt>)? (too? well)? |
    i am not hearing       (you | <last-prompt>)? (too? well)? |
    i am not understanding (you | <last-prompt>)? (too? well)? |
    you spoke too <fast> and  i missed <last-prompt> |
    you spoke so  <fast> that i missed <last-prompt> |
    <last-prompt> went right? by me  |
    <last-prompt> went right? over my head |
    what did you just? say (to me)? |
    what did you just? tell me |
    what was <it-that> (that? you just? said (to me)?)? |
    what is  <it-that> you said (to me)?
    .

<last-prompt> =
    <it-that> |
    what was said |
    your response (to me)? |
    what you just? said (to me)? |
    what you are saying (to me)? |
    what you are trying to (tell | say to) me |
    what you are telling me |
    (the | your | that) last prompt |
    what you just? told me
    .

<didnot> =
      did not |
      didn't  |
      do not  |
      don't   |
      could not |
      couldn't |
      can not |
      can't
      .
```

```
<again> =
    again |
    one more time
    .

<it-that> =
    it |
    that
    .

<fast> =
    fast |
    quickly |
    quick |
    hurriedly
    .

<sorry> =
    pardon me |
    (i'm | i am)? sorry |
    excuse me |
    i beg your pardon
    .

<get> =
    get |
    understand |
    hear |
    comprehend |
    follow |
    catch
    .

<request> =
    i need you to  |
    i want you to |
    i'd like you to |
    i would like you to |
    can you please? |
    could you please? |
    will you please? |
    would you please?
    .

<request2> =
    can i please? |
    could i please? |
    may i please? |
    i'd like to
    .
```

Spoke-Too-Soon

Rationale

According to Balentine and Morgan (1999, p. 120):

"A spoke-too-soon error is a short 'stumble' that is a natural consequence of user turn taking. It should not be treated as a complex problem that requires extensive coaching. In fact, the user should be allowed to speak again as soon as possible after the spoke-too-soon."

Sample Dialog 1

System: Do you want to do another transaction? [!!!]

User: (interrupting the beep) Yes.

System: Remember to wait for the tone. Do you want to do another transaction? [!!!]

Sample Dialog 2

System: Do you want to do another transaction? [!!!]

User: (interrupting the beep) Yes.

System: <beep> <beep> [!!!]

"The application uses non-speech audio to cue the user that the machine was not ready. This solution is even shorter than spoken feedback. Users learn to respond to such cues quickly – within their first-use session (Balentine et al., 1997). In the example, the user repeats "Yes" after hearing the triple beep." (p. 121)

Potential Uses for Counting Spoke-Too-Soons

Again referring to Balentine and Morgan (1999, p. 123), you should track spoke-too-soon errors separately from other speech errors:

"The spoke-too-soon error is a natural consequence of the half-duplex turn-taking protocol. Human interactions often include dovetailing – in which the new talker begins just as the old talker is finishing. This 'overlapping' turn-taking behavior is perfectly normal for human speech and can therefore be expected to occur often when interacting with a speech-enabled application.

For this reason, spoke-too-soon errors should not be blown out of proportion. If a quick re-prompt captures a user repetition, then the error does not seriously impeded the flow of the dialogue. The application should be tolerant of this stumble.

Bear in mind that spoke-too-soon conditions – unlike most speech errors – can be expected to increase as users become expert and comfortable with the application. This is because they come to anticipate machine prompts and queries, and because they become interested in moving more quickly toward goals. This phenomenon can be a feature if spoke-too-soon recovery methods are quick and graceful. Conversely, it can be a problem if such methods are clumsy and slow. Poor spoke-too-soon recovery tends to slow down users as they become expert, causing them to adopt a turn-taking rhythm that diminishes their preferred (natural) pace. This increases anxiety and irritation, and distracts users from their primary goal."

We probably don't need to keep a running count of spoke-too-soon errors, but should present the short version (Sample Dialog 2) the first time a user speaks too soon in a specific sub-turn. We should then use Sample Dialog 1 until the user gets back on track, returning to Sample Dialog 2 for the next first occurrence of spoke-too-soon for a prompt. At any given prompt, the system should not allow for infinite recoveries from a spoke-too-soon error, so it is necessary to keep a local count at each prompt.

Backup and Cancel

Rationale

In any reasonably complex non-GUI system (for example, speech-only or DTMF), users (especially first-time users) might need to explore the interface. In exploring, they might go down an unintended path, and will need commands to backup through the menu (dialog) structure. If a system has sufficient complexity, it might be necessary to distinguish among three desired user actions – backing up to the immediately preceding prompt (BACKUP), returning to the most recent submenu (CANCEL), and starting over at the main menu (START-OVER). If a system has a simpler structure (only one function or one menu), then the backup and cancel actions should have the same system consequences. For the systems we are currently designing, we can treat backup and cancel identically, but we should be prepared to treat them differently in the future.

Sample Dialog

System: Say one of the following choices: […]

Leave message […]
Camp […]
Forward call [!!!]

User: Forward call.

System: Forward to which 4-digit extension? [!!!]

User: Cancel.

System: Say one of the following choices: […]

Leave message […]
Camp […]
Forward [!!!]

Potential Uses for Counting 'Cancels'

Because cancellation moves the user back through the dialog structure one step at a time, there doesn't seem to be much reason to count cancels with one exception. If the user keeps issuing a cancel command at the highest-level prompt in the system, where the system can't back up any farther, then it probably makes sense to stop after three repetitions of that prompt and bail out using the bail out sequence described under Help:

System: Sorry for the confusion. Transferring to an operator.

Or, if the system has no operator:

System: Sorry for the confusion. Disconnecting … please call back and try again.

A Sample ‘Cancel’ Grammar

```
<cancel> = <cancel-action> please?  |
    please? <let-me> <cancel-action> |
    <how-about> <cancel-acting>     |
    <i-would-like-to> <cancel-action>
    .

<i-would-like-to> =
    i would like to     |
    i'd like to         |
    i want to           |
    i need to           |
    i wish to           |
    (<can-could-would-will> you)? please? |
    (<can-could-may>          i)? please? |

<how-about> =  i am interested in         |
    how about                             |
    would you mind                        |
    do you mind                           |
    I feel like
    .

// works with "You"
<can-could-would-will> =
    can |
    could |
    would  |
    will
    .

// works with "I"
<can-could-may> =
    can |
    could |
    may
    .

<cancel-action> = cancel      |
    backup                    |
    go back (<oneA> step)?
    .

<cancel-acting> = cancelling  |
    backing up                |
    going back (<oneA> step)?
    .

<oneA> = one                  |
    a
    .
```

Start Over

Rationale

In any reasonably complex non-GUI system (for example, speech-only or DTMF), users (especially first-time users) might need to explore the interface. In exploring, they might go down an unintended path, and will need commands to backup through the menu (dialog) structure. If backing up one step at a time seems to be too cumbersome, the user might want to just start over.

Sample Dialog

System: Say one of the following choices: […]

Phone options […]
Fax options [!!!]

User: Phone options.

System: Say one of the following choices: […]

Leave message […]
Camp […]
Forward call [!!!]

User: Forward call.

System: Forward to which 4-digit extension? [!!!]

User: Start over.

System: Say one of the following choices: […]

Phone options […]
Fax options [!!!]

Potential Uses for Counting 'Start overs'

It would be possible to count the number of times the user starts over, but such a count would probably not be useful because a user might start over for either of two reasons. A start-over might indicate the user got lost in the system, but it might also indicate nothing more than that the user has accomplished one goal and wants to start using the system again. Thus, counting these events would usually have little value in taking the user down alternate paths in the dialog structure. It might make sense to keep track of these in the system log, however, because consistent start-overs from non-terminal task areas could indicate a usability problem.

A Sample ‘Start over’ Grammar

```
<start-over> = <startover-action> please? |
    please? <let-me> <startover-action> |
    <how-about> <startover-acting>  |
    <i-would-like-to> <startover-action>
    .

<i-would-like-to> =
    i would like to     |
    i'd like to         |
    i want to           |
    i need to           |
    i wish to           |
    (<can-could-would-will> you)? please? |
    (<can-could-may>          i)? please? |

<how-about> =  i am interested in         |
    how about                            |
    would you mind                       |
    do you mind                          |
    I feel like
    .

// works with "You"
<can-could-would-will> =
    can |
    could |
    would  |
    will
    .

// works with "I"
<can-could-may> =
    can |
    could |
    may
    .

<let-me> = let me                  |
    allow me to
    .

<startover-action> = start over     |
    start again                    |
    begin again                    |
    go to the (beginning | top)
    .

<startover-acting> = starting over  |
    starting again                 |
    beginning again                |
    going to the (beginning | top)
    .
```

Functions That Remove the User from the System

Operator

Rationale

Designers of telephony systems try to create systems that allow their clients (purchasers of the system) to reduce the cost of their call centers while still maintaining a high level of user (user of the system – usually employees or customers of the client) satisfaction. One current area of uncertainty in telephony system design is whether to make it easy or difficult for users to transfer out of the recognition system to a human operator. This could vary widely from client to client, with some clients wanting to make it very easy and others eliminating all human operators from their organization. If the client has no human operators or doesn't want to make it easy to connect to an operator, then it doesn't make any sense to include a global operator function in the system. It does makes sense to include such a function for clients who want to make it easy for their users to leave the recognition system and talk to an operator. Unless the designer knows that a client will not have an operator or operators, he or she should design the system to make it easy to configure the system to include functions that move the user from the automated system to an operator.

Sample Dialog

System: Welcome to the AussieBank currency conversion system. […]

User: Operator, please.

System: Transferring to operator.

Potential Uses for Counting 'Operators'

Because the operator function removes the user from the recognition system, it isn't even possible to use an operator count as a guide for managing a specific user. Keeping track of the number of operator commands issued in the system log and noting any consistency in where the user was when he or she issued an operator command could be useful for identifying usability problems in the system.

A Sample 'Operator' Grammar

```
<operator> = operator please?       |
    please? <give-me> an? operator  |
    <i-want-to-talk-to> an? operator
    .

<i-want-to-talk-to> =
    i would like (to <talk-to>)?    |
    i'd like     (to <talk-to>)?    |
    i want       (to <talk-to>)?    |
    i need       (to <talk-to>)?    |
    i wish        to <talk-to>      |
    (<can-could-would-will> you)? please? <give-me>  |
    (<can-could-would-will> you)? please? provide me with  |
    (<can-could-may>         i)? please?  get       |
     <can-could-may>         i   please?  have      |
    i am interested in getting   |
    how about <giving-me>        |
    would you mind <giving-me>   |
    do you mind <giving-me>      |
    please? <let-me> <talk-to>
    .
```

```
// works with "You"
<can-could-would-will> =
    can |
    could |
    would  |
    will
    .

// works with "I"
<can-could-may> =
    can |
    could |
    may
    .

<giving-me> =
    giving me |
    getting for? me |
    telling me
    .

<give-me> =
    gimme  |
    get me |
    give me
    .

<let-me> = let me                   |
    allow me to
    .

<get-operator> = operator           |
    <talk-to> an operator
    .

<talk-to> = talk to                 |
    transfer to                     |
    connect to                      |
    communicate with                |
    talk with                       |
    get
    .
```

Goodbye

<u>Rationale</u>

Most users will probably just hang up when they have finished using the system. Some users, however, might feel more comfortable that they have completed their desired task by saying goodbye to the system, then hearing the system confirm task completion with a closing message. By allowing this type of grammar, if a user has unintentionally left the task incomplete, the system can notify the user of this and prompt them to complete the task. A goodbye grammar also provides an opportunity to engage users in post-usage satisfaction surveys.

<u>Sample Dialog 1</u>

User: Goodbye.

System: Goodbye. Thank you for using our automated appointment system.

<u>Sample Dialog 2</u>

User: Goodbye.

System: You haven't specified an appointment date. Do you want to cancel this appointment? [!!!]

User: Yes.

System: Goodbye. Thank you for using our automated appointment system.

<u>Potential Uses for Counting 'Goodbyes'</u>

Because the goodbye function removes the user from the recognition system, it isn't even possible to use a goodbye count as a guide for managing a specific user. Keeping track of the number of goodbye commands issued in the system log and noting any consistency in where the user was when he or she issued an operator command could be useful for identifying problem areas in the system. For example, goodbye commands issued after a successful task completion point would indicate normal operation, but goodbye commands issued at other points in the dialog flow could indicate usability problems (as when users hang up during a call, but before completing a task).

A Sample ‘Goodbye’ Grammar

```
<goodbye> =
    goodbye                                 |
    adios                                   |
    bye? bye                                |
    no more                                 |
    so long                                 |
    <thk>                                   |
    that is it                              |
    that's it                               |
    that is all              <thk>?  |
    that's all               <thk>?  |
    that will be all         <thk>?  |
    that'll be all           <thk>?  |
    i am done                <thk>?  |
    i'm done                 <thk>?  |
    i am through             <thk>?  |
    i'm through              <thk>?  |
    i am finished            <thk>?  |
    i'm finished             <thk>?  |
    it has been a pleasure  <thk>?  |
    okay <thk>?
    .

<thk> =
    thanks |
    thank you |
    thank you very much |
    thanks very much
    .
```

Discussion

In this document, we have described six functions that every speech-only user interface should include (help, repeat, spoke-too-soon, cancel/backup, start-over, goodbye) and one that many clients will want to have in their system (operator). These descriptions and sample grammars should help developers design usable and consistent speech-only user interfaces[3].

One interesting result of the counting analyses is that, with some exceptions, there doesn't seem to be much reason to count the usage of these functions as a component of mechanisms for controlling dialog flow. The implicit, self-revealing help strategy doesn't rely on counts for control, but falls through a predefined series of prompts, ending with the removal of the user from the system (either to an operator, if available in the system, or by disconnection otherwise)[4]. The system should not allow infinite repeat commands, infinite recoveries from spoke-too-soon errors, or infinite cancels/backups when there's nowhere to go (which means the system is acting like a repeat). Therefore, it does make sense to count these events and use an excessive number as a guide to transferring the user out of the system (either to an operator, if available, or disconnecting).

---

[3] For absolute consistency, speech applications developed using these always active commands should include in the always-active grammar, as a minimum, the commands HELP, REPEAT, BACKUP, CANCEL, START OVER, and GOODBYE. The use of the larger grammars is optional, and their use should be limited to applications that have this type of command flexibility throughout the entire system. Systems that allow users to transfer to an operator should also include in this set the command OPERATOR.

[4] The only exception to this in the help strategy is if you want to present a different bail-out prompt if the user has fallen through the sequence of prompts exclusively with silence timeouts. Even in this case, though, it should be possible to track this with a single flag rather than counting the events.

References

Balentine, B., Ayer, C., Miller, C., and Scott, B. (1997). Debouncing the speech button: A sliding capture window for synchronizing turn-taking. *International Journal of Speech Technology*, *2*, 7-19.

Balentine, B., and Morgan, D. P. (1999). *How to build a speech recognition application: A style guide for telephony dialogues*. San Ramon, CA: Enterprise Integration Group.