Moving the Conversation Forward

Susan Hura
Juan E. Gilbert
Silke Witt-Ehsani
Karen Kaushansky

Table of Contents

Introduction

When you have suboptimal error handling, the response of the system can cause a downward spiral and create the appearance to the user that they are in a bottomless pit. Error prompts can magnify this situation; instead of moving the conversation forward and closer to completing the user's intended task, the user feels stuck. One example is when an error prompt itself generates user responses that are not recognized and causes an increase in user frustration. Reprompting with the same wording may encourage users to rephrase their response and continue to be misrecognized without understanding the cause of the misrecognition and how to get back on track. As well, this occurs with turn taking errors, and false starts, which can build error upon error resulting with the user ending up in this downward spiral.

Why is the bottomless pit a problem? Firstly, it diverts the user further away from their task, getting stuck in one small step of a longer process. Secondly, it does damage to the user's confidence and willingness to use the system, increasing frustration levels and potentially hampering the rest of the interaction between system and user.

So how do you move the conversation forward or give the user perception of progress?

Analyzing human – human conversation shows that with such conversations, error can occur just as likely as with a speech recognition application, but the big difference is that humans have developed a number of strategies to repair errors and move the conversation forward. Based on patterns that can be observed in human – human conversations, we present here several strategies to give speech applications the same capability to get past errors and to successfully continue a call.

Give the user visibility to what is going on

For critical pieces of information that are needed, give the user visibility into where they are in the process and how to achieve their intended goal. For example, instead of asking for the date, inform the user of why it's needed: "Before I can make the reservation I first need to get the date."

Ignore errors in non-critical parts of the dialog

For non-critical pieces of information, or information that could be collected later in the call, consider ignoring the error state. First, be sure that every piece of information you ask of the user is actually needed. Next, if asking a non-critical question such as repeating a website address, then on a nomatch or timeout, there is no need to reprompt and instead take the opportunity to move on:

> "For more information, go to our website at blahblah.com. Would you like me to repeat that?"
>
> [silence]
>
> "Okay, next to complete your transaction…"

Even if the information is critical, if a nomatch occurs, consider reprompting later in the call to collect the information, especially if there is a chance that the caller will need to talk to an agent or the information being asked for is only required for certain paths of the application. For example, in a pizza ordering application, if a nomatch occurs on coupon recognition, consider moving on and collecting it later or with an agent. This will help move the conversation forward and will build confidence with the user by what can be accomplished in the dialog:

> "What's the coupon code?"
>
> "9-7-g-4-g-5"
>
> "Hmm, I didn't quite make out that coupon code but let's get your order first and then we'll be sure to get that coupon from you."

The other point to make here is that it is important to build user confidence at the beginning of the call. Choosing more recognition-safe interactions up front, or moving on if nomatches occur at the

top of the call, will build trust between user and application while forward progress is being made in the call. Once the user has invested a certain amount of effort in a call, it is possible that a user is more likely to attempt to correct an error later in the call knowing how far they've gotten.

Here are some other strategies for moving the conversation forward:

Smart reprompting for partial information

A typical human discourse element is that if someone only understands half of a sentence, they phrase a question asking for the missing piece of information while also telling the other person which part they had understood. The same principle can be used in speech applications.

Using some conversational markers and dialog, reprompt only for missing slots when asking for multiple slots in an utterance. For example, when asking for the month and day reprompt with "Great, what day in February?" instead of "I'm sorry, please repeat the day in February". Or when asking city and state, if the state is recognized then use that information to move forward:

> "Please say a city and state"
>
> "Fresno, California"
>
> "What city in California?"

Along the same lines, there may be ways to reprompt users smartly when collecting digit strings or alphanumerics, so that the entire utterance does not need to be repeated.

> "Please say your account number."
>
> "6-4-3-2-4-5-3-2"
>
> "What were the last two digits of that account number?"

Use barge-in off where possible

Another strategy is to ensure that false barge-in does not happen at the beginning of a call since when it does happen, the first prompt the user may hear is a nomatch prompt. This is not a recommended way to build user confidence at the start. By not allowing barge-in for some portion of the initial prompt, or detecting false barge-in and reprompting the user with the initial prompt instead of the nomatch prompt, errors are avoided and the conversation progresses without the threat of the bottomless pit. Here is an example:

> "Welcome to XYZ, what is your account number?" (this prompt receives a false barge-in because the caller coughed before the prompt began)

"?*%&.." (user cough with very low confidence score)

"I'm sorry, I didn't get that, please repeat your account number." This becomes the first prompt the caller hears instead of the intended prompt because of the false barge-in. Had barge-in been turned off, the caller would only have heard the no-input prompt.

Special NoMatch handling for noise and sidespeech

Another alternative would be to have separate error handling for very low confidence nomatches that are typically caused by coughs, noise and sidespeech. The strategy for those errors would be to not play any error message but simply repeat the prompt. So in the above example, the interaction would become:

"Welcome to XYZ, what is your account number?" (this prompt receives a false barge-in because the caller coughed before the prompt began)

"?*%&.." (user cough with very low confidence score)

"Sorry, WHAT is your account number?"

This approach has the purpose of not stopping the call due to errors that are not user errors but errors created by a sub-optimally performing recognition engine.

Use fallback strategies

Another strategy is to use a fallback mechanism. That is, if a caller has problems with providing a particular piece of information, don't keep trying to get that piece of information multiple times, but rather switch to collecting an alternative piece of information. For example, if a caller has problems with an account number, it is often possible to switch to asking for a social security number or last name instead.

Conclusions

In summary, when designing a speech application, designing for error handling is a crucial component in creating successful applications. We presented a series of techniques to address different error root causes which will serve as a toolbox for voice interface designers.