

From Touchtone to Speech

Six Steps for Speech-Enabling Your DTMF IVR

Chris Biber, Andrew Kozminski
Sonia Langenberg

Pronexus Inc.
Spring 2005

Introduction

Organizations of all sizes have used Touch-Tone interactive voice response (IVR) systems to automate various customer-facing business processes. While these Touch-Tone IVR systems can save money, they also have inherent drawbacks such as complex menu structures and difficult caller navigation. Speech-recognition technologies can resolve many of these shortcomings. Transforming a dual tone multi-frequency (DTMF) system into a speech-enabled solution can boost automation levels, raise customer satisfaction, and increase operational efficiency. Speech-recognition technologies also open the door to entirely new applications, such as an automated address change system, that are not possible in Touch-Tone IVR systems.

Articles abound regarding the merits of speech applications and their impressive return on investment, but more information is needed about the planning that should precede the migration from Touch-Tone to speech recognition. This document addresses that need.

While getting an existing Touch-Tone application "up to speech" (adding speech recognition capabilities or converting an application to speech recognition) is not a trivial undertaking, it is well worth the investment in time and resources. Proper planning, design approaches, and tools can greatly simplify the task.

While this document is written primarily for organizations that have customer premises equipment (CPE) applications, these guidelines apply equally well to hosted applications.

Choosing the Application Most Suitable for Conversion

Which application is the best candidate for speech recognition? This is a common question in any organization where more than one Touch-Tone application is deployed. To find the right answer, consider the number of calls each application receives and how many callers engage with the application. While these two gauges alone can identify potential conversion candidates, be sure to consider other measures as well. For example, analyzing the call completion rate of the existing Touch-Tone application can help you to understand current usage patterns and determine the best possible candidate for an initial conversion project. In general, a low call completion rate with a highly used application might point to a good candidate for an upgrade to speech. Several other key metrics are discussed later in this document.

Tip If possible, start small. Do not make your most business-critical Touch-Tone application your first conversion project.

Six Steps to Speech-Enable a Touch-Tone Application

Designing a good voice user interface (VUI) is a demanding process that requires the interdisciplinary talents of computer science, linguistics, and human factors psychology. The collection and analysis of past caller interactions is also extremely valuable. Consider the following steps when making the transition from Touch-Tone to speech.

1. [Select Your Speech Components](#)
2. [Redesign Your Application](#)
3. [Design Application Prompts, Grammars, and Dialogues](#)
4. [Increase Application Success with Proper Error Handling](#)
5. [Tune, Tune, and Tune Again](#)
6. [Measure Application Success](#)

Each of these six steps is described in detail in the following sections.

These six steps are needed because speech applications are more complex than DTMF solutions. While successful speech applications typically make for a simpler caller interaction by flattening DTMF menu

trees, speech application development requires additional effort to compensate for different levels of recognition accuracy.

With Touch-Tone interfaces, you always limit caller input to one of ten digits, and DTMF detectors guarantee almost 100% accuracy. Speech is different. In addition to lower recognition accuracy, developers need to consider background noise and other factors that make recognition even more difficult. In a speech application, a caller can say anything. This all leads to increased application design complexity. Speech application developers commonly expend more programming effort compensating for errors (such as low accuracy, ambiguity, and out-of-grammar vocabulary) than on the actual call flow and business-related features.

Unlike Touch-Tone IVR systems, speech systems are also highly dependent on the demographics, local accents, language mix, and the culture of the target audience.

1. Select Your Speech Components

Speech Engine

The first step is to decide on the speech engine that will power your speech-enabled application. Although you can start with call flow development and VUI design efforts without choosing an engine, the various engine features and design aspects have a significant influence over all other aspects of the final speech application. While most speech-recognition engines have similar performance characteristics (often with excellent recognition results), they differ in the number of supported languages, programming environment, and overall implementation.

Cost and complexity are two important factors that affect the speech platform decision.

Many organizations have rejected speech applications in favor of Touch-Tone, citing expensive proprietary platforms, spotty and inconsistent standards, and complex integration of software, hardware, and third-party components. This paper advocates the move to speech, but many applications are well-suited for Touch-Tone implementations. Imagine entering credit card and pin numbers using speech recognition in a public venue.

Speech-recognition technology has been implemented successfully since the early 1990s. However, only recently have cost and complexity been reduced to allow for mainstream adoption. The two critical developments in this regard are the evolution of open standards and the emergence of a new class of competitively priced platforms, namely Microsoft® Speech Server 2004.

Open Standards

The development of standards has greatly benefited the speech market. On the face of it, open standards, such as VoiceXML and Speech Application Language Tags (SALT), simplify speech development by making tools interchangeable and by creating a common programming paradigm.

Proprietary platforms, on the other hand, have the inherent downside of vendor lock-in, which reduces customer flexibility.

Before committing to a standard or considering a proprietary platform, you should consider the costs and benefits of each for the specific applications you want to develop.

VoiceXML is older and more widespread than SALT. It has been adopted as an official W3C standard and has found support among many industry players and their solutions. Its current implementation is VoiceXML 2.0. VoiceXML is designed specifically for speech-based dialogues in telephony-based applications. However, some inherent problems in VoiceXML have led to dramatically higher development and implementation costs. For example, the following problems negate many VoiceXML benefits:

- Lack of integration – VoiceXML engines, tools, and libraries all come from different vendors
- Lack of industrial-strength programming and debugging tools

- Incomplete support for telephony, which initially led to many proprietary extensions

SALT is an open standard that made its commercial debut with the release of Microsoft Speech Server 2004, along with tools and solutions of many Microsoft Speech partners. Sponsored by Microsoft, SALT is specifically designed for defining speech application logic. SALT supports both telephony and multimodal applications (such as desktop and tablet PCs, PDAs, and cell phones). SALT uses existing Web markup languages and standards for speech output, grammar formats, and semantic results. Through Microsoft Speech Server, SALT is integrated into the Microsoft Visual Studio® environment, which enables developers to take advantage of the strong programming and debugging tools of that platform.

However, probably the biggest contribution of SALT and Microsoft Speech Server is in the overall reduction of both cost and complexity for application development. Microsoft Speech Server is the first platform to integrate the following critical technologies that are required to develop speech-based business applications at an affordable price.

- Automated speech recognition (ASR) and text-to-speech (TTS) engines
- Grammar editing tools
- Authoring and debugging programs.

While the cost of traditional speech licenses can reach \$2,000 per line, bringing a 24-line deployment close to \$50,000, the new Microsoft Speech Server platform provides 24 lines of speech licenses for \$7,999. By simplifying application development, Microsoft Speech Server also helps to reduce the other large cost component of speech applications.

However, despite its affordability, the newness of Microsoft Speech Server might cause some to take a wait-and-see approach or to opt for competing technologies, such as Nuance or ScanSoft/SpeechWorks. Although these engines have been available for more than a decade, their cost is significantly higher and the application development efforts require the integration of various third-party hardware and software components.

Choose the best standard for the Touch-Tone application that you are converting to speech. For example, for a multimodal application, choose SALT instead of VoiceXML because SALT was designed to accommodate multiple modality.

Evaluate the openness of the standard against the unique requirements of the application. For example, if the IVR application relies on sophisticated call control, VoiceXML alone is not sufficient because it does not support the more advanced scenarios that might be required. Instead, the application would require proprietary extensions that in turn negate the benefits of vendor independence and application portability. As a result, a proprietary system might be the better and less expensive choice.

Tip If you are considering a move to an open standard, make sure the tools you choose have a well-defined migration path.

Platform and OS

Another important consideration is the underlying technology and its fit into the target environment. VoiceXML platforms tend to be stronger on UNIX and Java, which might be a better fit for carriers who do not want to migrate to Windows. On the other hand, because Microsoft technology typically dominates the enterprise area, an application that uses Windows to build an enterprise application would make integration with business back-end servers much easier. Therefore, SALT would be the natural platform choice for enterprise applications.

The importance of *five nines* reliability cannot be overstated. Five nines reliability refers to a technology or service that achieves 99.999% uptime. Telephone companies frequently tout their service as having five nines reliability. Practically speaking, this means that telephone service is down for only five minutes per year.

Many IVR systems tend to be one-of-a-kind projects, custom designed for the needs of a specific business. In my own experience (10 years of building such applications), I have found that problems with the operating system itself are rare and have concluded that both UNIX and Windows are reliable enough

for the job. The custom applications themselves are much more likely to fail, primarily due to bugs in the programs. From that perspective, choosing the proper development tools (programming languages, integrated development environment (IDE), and debugger) is more important to overall system stability than the operating system itself.

2. Redesign Your Application

If the business processes that lie beneath the DTMF application have not changed, you do not need to change the application logic during the conversion from Touch-Tone to speech. You will, however, need to make significant changes to the Touch-Tone call flow to take full advantage of the VUI.

Before you redesign your call flow, you should decide on which features to provide to callers. Will the application be entirely speech-enabled or will speech automation be restricted to a part of the interaction? Will the application offer Touch-Tone fallback throughout? Will it feature *barge-in*, allowing callers to speak while a prompt is playing?

Good practice mandates that you provide Touch-Tone fallback for those times when a caller is having trouble with spoken input, is concerned about security (for example, saying a credit card number, PIN, or password in a public venue), or simply prefers to use Touch-Tone.

As you redesign your Touch-Tone call flow, you should take advantage of the nature of speech recognition technologies. Rather than guiding a caller through an intricate system of nested menus, your speech application can offer a natural language request ("What type and size of coffee would you like?") The caller is then free to provide multiple pieces of information ("A double cappuccino, non-fat, one sugar.") This menu flattening is a key advantage of speech recognition interaction. Reducing the caller input steps can increase customer satisfaction while decreasing call duration and cost.

The design decisions you make when redesigning call flow will influence the level of complexity that the application developer will face:

- **Present callers with a main menu list of keywords.** A Touch-Tone developer will be immediately familiar with this approach and similar considerations apply. For example, you should limit the number of choices to fewer than five to avoid caller frustration. By specifying the accepted keywords in a spoken menu, the number of potential answers is minimized and grammar development is manageable.
- **Provide directed dialogue.** This method involves questions that are answered by Yes or No. (for example, "Would you like to pay your bill?"). Directing the caller and thus limiting the number of potential answers also simplifies application design and reduces the complexity of speech recognition grammars. Many speech applications are built around a directed dialogue design.
- **Interact with the caller through natural language.** This is the most complex approach because it enables a caller to say almost anything in response to an open-ended question such as "What would you like to do today?". To successfully implement natural language recognition, you need a thorough analysis and understanding of caller interaction. The resulting grammar set is comprehensive, covers all potential caller utterances, and detects keywords in sentences.

The design method you choose is driven by factors such as the complexity of the existing Touch-Tone application, the developer's knowledge of speech application design, and your audience. If your customers are experienced callers who prefer to shorten calls by responding during a prompt, your application should allow for barge-in and should encourage callers to respond at any time. By the same token, studies show that elderly callers often mistake an automated response for a live receptionist or agent and begin to engage in a conversation. Knowing your audience can help you plan your application wisely to avoid costly post-deployment problems.

A well-architected Touch-Tone application is modularized. That is, it separates the user interface from the business logic, making it easy to maintain and update. One way to achieve modularity is by using Web services to enable various applications (such as telephony IVR, databases, and billing systems) to share data and functionality. More importantly, however, a modular application invokes capabilities from other applications without regard to how those applications were built, what operating system or platform they

run on, and what devices are used to access them. Web services are invoked over the Internet by means of industry-standard protocols such as Simple Object Access Protocol (SOAP) and Extensible Markup Language (XML).

3. Design Application Prompts, Grammars, and Dialogues

Because this paper discusses the conversion of an existing Touch-Tone application, the planning and design processes herein are abbreviated. You already understand the automation task and the information that is required from the caller. At this point, you need to sketch out the conversation scenarios to capture the appropriate prompts to play and the proper responses to caller input.

Prompts are the short audio files that are played to the caller. They provide important navigation cues within the spoken dialogues. Prompts can cause the caller to speak, or provide them with the acceptable commands at the specific point in the application. Prompts can be prerecorded or dynamically generated by using TTS technology.

Prompts should be as short as possible and, in their first instance, should be preceded by a set of instructions. A prompt can end with a tone to cause the caller to speak or allow for barge-in during playback. The clear and unambiguous wording of a prompt is a key contributor to application success. Try to avoid compound questions and questions that allow multiple answers (not just Yes or No).

Prompts also provide feedback to the caller. Where possible, this feedback should embed confirmation of speech input. For example, after the caller has spoken a five-digit number, the feedback might incorporate that number and continue with the application, thus reassuring the caller that the interaction is proceeding correctly.

The application grammar defines which responses are acceptable to the application. It contains a list of the exact words the caller might say and the order in which these words might be said. Any caller utterance that is not contained in the grammar will not be recognized. In selecting the vocabulary, you should test callers so that you can identify commonly used words and find terms that might cause a problem for the speech recognition engine.

Try to anticipate all possible answers to a question. For example, a speech-enabled auto-attendant might ask callers to say the last name, the first name, or both names of the person they are looking for: "Smith," "John," "John Smith." When these variations are included in the grammar, the speech application recognizes any of the words or phrases. However, if the first name, "John," is not included in the grammar, the engine will only recognize the utterance "Smith" or "John Smith."

Most speech recognition engines provide prebuilt grammars, so you do not have to develop them entirely from scratch. However, for best results, you should tweak the grammar through testing and tuning (see [5. Tune, Tune, and Tune Again](#)).

After you define the grammar, you need to build the word pronunciations. While most speech engines supply a default pronunciation for each word in the vocabulary, it is often insufficient. Varying geographical, cultural, and gender accents and dialects require multiple pronunciations per word to account for the various ways in which a word will be spoken.

Grammar development is closely tied to the design of the call flow dialogues, which are the specific prompts or questions that the application will ask and the possible answers that the caller can say in response.

Touch-Tone application designers are familiar with DTMF menus, which can often be migrated to speech, provided that the developer adheres to a few guidelines:

- While DTMF menus commonly present a function–action pairing ("For Sales, press 1"), the speech application should present the choices in the most obvious form possible ("You can say 'Sales' or 'Support.'").
- Provide callers with navigational feedback to reassure them of their location within the call flow ("Main Menu. You can say 'Order,' 'Support,' or 'Operator.'")

- While DTMF menus often begin with the listing of available Touch-Tone options, speech dialogues are more effective when you let the caller speak first or after a short prompt. If the caller chooses not to speak, the system should then proceed to present the menu choices.
- Enable the caller to interrupt the prompt (barge-in) when they hear the appropriate command.

After you have built your grammar, you need to record the new prompts to inform callers that they can now speak their responses instead of pressing their Touch-Tone keypad. For example, a new prompt will inform the caller, "You can say 'Sales', 'Marketing' or 'Support' or simply say the name of the person you'd like to reach."

When choosing your speech application's voice (persona), be sure to choose a person whose voice talent reflects your company's brand. (Companies typically seek a friendly, welcoming voice to record grammar prompts.)

4. Increase Application Success with Proper Error Handling

Error handling and recovery with a speech interface is vastly different from error handling and recovery with a Touch-Tone interface. DTMF applications have a lower range of errors because typically the caller presses a single key that Touch-Tone detectors rarely misinterpret. Conversely, speech engines accept a wide range of input, leading to a greater possibility of error. The speech engine might misinterpret similar sounding words, or incorrectly accept a word that is not defined in the grammar. Speech engines might also incorrectly reject a word contained in the grammar or reject the last word of a phrase. The prevention and proper handling of errors is crucial to a successful speech application.

Prompt Escalation

One approach to enhancing the customer experience is prompt escalation, where the prompt is changed each time the application queries the caller for the same data item.

When input recognition fails (for example, due to low speech recognition engine confidence, words out of grammar, or no speech detected), you should provide a different prompt that gives the caller more information to resolve the problem and to move the dialogue forward.

Consider the following example without prompt escalation:

*<System prompt> What city are you calling from?
 <Caller> (noise in background) Toronto (more noise)
 (recognition fails because of background noise)
 <System prompt> What city are you calling from?*

In this scenario, the caller might not know why the system is reprompting with the same questions. (Is the system not working? Did I say something wrong?)

Now consider an example with prompt escalation:

*<System prompt> What city are you calling from?
 <Caller> (noise in background) Toronto (more noise)
 (recognition fails because of background noise)
 <System prompt> What was that? What city are you calling from?
 <Caller> (noise in background) Toronto (more noise)
 <System escalated prompt> Too much noise. Please remain silent for a moment. ... Let's try again. What city are you calling from?
 <Caller> (noise) Toronto...(more noise)
 <System> Are you calling from Toronto?*

In this scenario, the caller is given more information, including a method for improving the results. The system, in turn, uses the pause to change the recognition thresholds to account for the background noise. The caller is assured that the system is working and stays on the line.

If all fails, you might want to encourage callers to use Touch-Tone input when spoken entries have been recognized incorrectly. This not only avoids caller frustration, but also increases the number of completed interactions using the automated system.

Confidence Thresholds and Confirmation Interaction

In addition to developing prompt escalation strategies, you must also set confidence thresholds and zones.

When the speech recognition engine recognizes an input or utterance, it returns a value between 0 and 100 that indicates how confident it is of the match. The action that your application takes depends on this value and the thresholds you have set.

You should set two confidence thresholds: Rejection and Confirmation.

- Rejection threshold: An utterance with a confidence below this value is rejected as NoReco (not recognized).
- Confirmation threshold: An utterance with a confidence above this value does not require confirmation. However, the system might provide an implied confirmation, such as "Connecting you to John Smith." This simple feedback reassures the caller.

An utterance with a confidence value between the two thresholds requires confirmation. That is, the system will verify the caller's input to ensure that the caller intended to say what the system interpreted the utterance to be. ("Did you say John Smith?")

With most speech tools, you can customize the values of your confidence thresholds. Your speech application's confidence thresholds and the resulting confidence zones might look like this:

100 — Full confidence

ACCEPT

80 — Confirmation threshold

CONFIRM

40 — Rejection threshold

REJECT

0 — No confidence

5. Tune, Tune, and Tune Again

Planning your dialogue design, grammars, and prompts is critical to building a successful application. However, because of their complexity and ambiguity, speech-enabled applications require extensive testing and tuning to ensure ongoing accuracy.

Tuning is an iterative process of analyzing system performance based on system logs and recorded caller interactions and then applying the best design practices to achieve the most satisfying customer experience and to work around technology imperfections. As a result, the tuning phase can take several months and might require an interdisciplinary team of professionals, including developers, testers, linguists, and psychologists.

To improve customer experience and overall application performance, the tuning process should be based on actual caller data. Because it is difficult to predict the full range of input that callers are likely to speak in response to a given prompt, actual data enables you to examine what callers really said to the system, and update the grammars and dialogues accordingly.

What should be analyzed and tuned?

Prompts — Questions should be unambiguous to prevent unexpected caller responses. For example, a prompt that asks for a telephone number should be "Please say your telephone number, beginning with the area code." If the prompt were "Please say your telephone number," the caller might not know that they are required to provide the area code.

Dialogue — The structure of the interface or the call flow is generally addressed in the initial design, but you can often tweak it for more efficient task completion. In some cases, callers have expectations about the dialogue and what is possible to do at any point. If these expectations are not met by the grammars or the dialogue flow, misrecognitions can occur, and callers might be taken down unexpected paths.

Grammars — The grammar must include every relevant response to a question. If a caller's input is "out of grammar," yet it provides a valid answer, the grammar should be updated to include the caller's response.

Recognition — Core engine parameters, such as confidence thresholds can be optimized to minimize unnecessary confirmations.

Confidence thresholds — If the speech engine repeatedly recognizes words correctly, but rejects them or attempts to confirm them unnecessarily, the confidence threshold might be set too high. Lowering the threshold might be able to cover these issues, but still reject true misrecognitions.

Pronunciation — Although speech engines provide default pronunciation dictionaries, a caller's pronunciation might not match the one in the dictionary. Therefore, customized pronunciations should be added to the grammar in question. Applications such as auto-attendants necessitate the adjustment of the pronunciation dictionary to accommodate name pronunciations on a regular basis, as the grammar (for example, a list of employee names) changes over time.

6. Measure Application Success

Collecting and analyzing metrics is an ongoing effort. Whether the overall goal is to reduce operational costs or to increase customer satisfaction, the metrics below capture some of the key measurements for DTMF and speech applications, sorted loosely by call progression. A before/after comparison allows a developer to assess the success of a speech application and to quantify potential benefits, even before undertaking the effort.

Be sure to supplement these metrics with other factors, such as the cost of a speech application system, and the envisioned and realized savings. In some cases, Touch-Tone applications will do the job more cost effectively.

Metric (Rates in %)	Definition	DTMF	Speech
Call Volume	The number of calls that are reaching this application	X	X
Take Up Rate	The number of callers that engage with the application	X	X
Zero-Out Rate	The number of callers that hit 0 (zero) to transfer to an operator/agent. High zero-out rates indicate problem areas.	X	
Abandon Rate	The number of callers that hang up and abandon the call. High abandon rates indicate problem areas.	X	X
Recognition rate (per prompt)	The success of each individual prompt in accomplishing its tasks. This helps you pinpoint problem areas.		X
Call Completion Rate	The number of self-service calls that are completed as intended through the system.	X	X

Call Duration	The length of the average self-service call. Speech recognition can dramatically shorten call times and enhance cost savings.		X
Caller Satisfaction	The satisfaction level of the callers. Customer surveys and analysis of actual call recordings help you establish this level.	X	X

Some organizations use additional metrics, such as overall automation rates. Speech recognition enables new applications that would be impractical with Touch-Tone. Recording a large number of individual calls is necessary to accurately identify problem spots and pinpoint areas for improvement.

Accelerate Success with RAD Tools

Developing a successful speech application is fairly complex. It not only requires careful design of grammars and prompts, but it also involves the integration of speech engines, object libraries, call processing frameworks, and development tools, often from different vendors.

To facilitate and accelerate this development process, you can leverage the capabilities of a high-level Rapid Application Development (RAD) tool. RAD tools abstract the maze of multiple-vendor components into a uniform development environment, thus shielding you from low-level complexity so that you can focus on the business logic instead of the technology. RAD tools accelerate your application development as they provide an intuitive graphical user interface (GUI) that enables visual programming through drag-and-drop building blocks.

Some RAD tools also leverage the power of industry standard integrated development environments (IDEs) so that developers can take advantage of familiar programming languages. This reduces the learning curve and accelerates application development even further.

When you select a development tool, ensure that it supports the speech engines or platform that you chose for your speech application, whether Microsoft Speech Server or another speech engine. Also, make sure the development tool provides a flexible programming environment that enables you to customize and extend existing building blocks or incorporate other components not provided by the toolkit to take full advantage of new hardware and software. This capability is critical for most real-world telephony applications.

VBVoice is an example of a RAD tool that incorporates support for the traditional speech engines such as Nuance and ScanSoft. VBVoice is designed for Microsoft Visual Studio and Visual Studio .NET, the standard IDE for Windows platforms.

VBVoice enables you to take advantage of existing technology platforms, leverage in-house programming skills, and use familiar debugging tools—all integrated in the Visual Studio environment. Its drag-and-drop components can be fully customized by using industry-standard languages such as Visual Basic, J#, C#, and any other language supported in Visual Studio and Visual Studio .NET.

Recently, Pronexus has also introduced VBSALT, a rapid application development environment that complements Microsoft Speech Server. Based on the VBVoice design principles and features, VBSALT is specifically designed to maximize developer productivity and reduce the learning curve for new speech application developers.

Like VBVoice, programmers can use VBSALT to leverage existing development skills in the languages supported in Visual Studio .NET. VBSALT features a comprehensive range of telephony and speech building blocks. VBSALT further facilitates successful speech application development by dynamically generating the SALT and JScript code required to control Microsoft Speech Server and by encapsulating best practices for VUI design. Because the low-level control of Microsoft Speech Server is completely transparent, you can focus on the business logic and VUI aspect of your application.

Conclusion – Get Up to Speech

This paper presented some guidelines for creating successful speech applications. To learn more, consider taking a class, such as *"Speech Applications: Planning, VUI Design, and Maintenance"* (see www.microsoft.com/speech/training for more information), and read *How to Build a Speech Recognition Application* (Bruce Balentine & David P. Morgan, 1999).

For more information about how to make the move to speech or to discuss your individual requirements, contact our team at info@pronexus.com or (613) 271-8989.

About Pronexus

Founded in 1993, Pronexus Inc. is the developer of VBVoice™ and VBSALT™, the only Rapid Application Development toolkits for telephony and speech inside Visual Studio.NET that blend both GUI and sophisticated programming. Pronexus also provides VeoSuite, a range of turnkey applications for Microsoft Speech Server, and a complement of professional services that voice-enable a variety of industries. Comprehensive support and acclaimed training complete the firm's offerings.