Developing Voice Applications

An IBM White Paper

This white paper is intended for software developers who want to learn about the tools and techniques available to create new voice applications. It focuses on IBM's role in voice middleware technology -- the applications, products, and tools. We'll show which applications are getting the most attention, and how voice technology can justify its cost. We'll also describe the latest enabling software and tell you how you can learn more about it and get ahead of the curve with this leading-edge technology.

Table of Contents

Introducing voice applications

Voice applications -- a sure bet for the future

While computer voice technology hasn't yet reached the sophistication of *HAL* in the 1968 movie *2001: A Space Odyssey*, it is beginning to approach that level. Today, voice systems such as IBM's *WebSphere Voice Server* can be enabled for conversational interaction in a variety of practical applications, such as call center support, telephone directory lookups, banking and investment transactions. Everyone agrees on one thing: someday, in the not-too-distant future, we'll be talking with computers as easily as we do with humans -- on the telephone, over the Web, or through a variety of embedded devices. Voice technology promises to continue its exponential growth.

IBM's history in voice development

IBM's voice technology research originated back in the late 1950s, but it wasn't until the early 1990s that improved software and faster hardware made it practical.

In 1992, IBM's AIX-based *Speech Server Series* supported the dictation of reports and letters. Next, the IBM *Continuous Speech Series* provided continuous recognition of spoken commands and phrases. This series was initially developed for the AIX and OS/2 platforms, and then followed by a Windows 3.1 version. In 1993, a personal voice product (which was also a commercial high-accuracy voice recognition product), IBM's Personal Dictation System, was released for OS/2. The following year saw the announcement of IBM *VoiceType Dictation* for Windows and OS/2.

Not long afterward, *VoiceType Dictation version 3.0* was released for Pentium systems, and the new version didn't require the special digital signal processing hardware adapter required for earlier systems. In 1996, IBM *VoiceType Simply Speaking* introduced high-accuracy spoken dictation technology to the retail market.

1997 marked the introduction of *ViaVoice*, a dictation product using continuous voice technology. Users no longer had to pause between words, and could speak at a natural pace. Today, forty years and over 150 patents later, IBM leads the industry with voice technology products.

For the voice application developer

This paper was written for software developers who are considering or planning voice applications -- either traditional IVR solutions or versatile multi-channel access applications that can be used

over the telephone or from voice-enabled Web sites. It should also be of interest to current Web developers who want to add voice capability to their sites.

Voice *middleware* encompasses platforms and applications that run on servers, such as the WebSphere Voice Server, serving hundreds or thousands of telephone or Internet customers. Generally, these server-based voice applications are written to service a limited vocabulary and a large number of users, such as bank customers. No "training" of a caller's voice is required. (On the other hand, voice desktop applications are usually created for a single user and a much larger vocabulary, such as voice dictation applications, and voice training is employed to improve accuracy).

This paper is directed at application software developers who will be writing, testing and installing voice middleware applications.

Some important definitions

Engines, models, grammars and dialogs

The voice recognition process is performed by a core software component known as the *voice recognition engine*, which translates spoken words into text in a format that an application can understand. Of course the application could simply leave the words as text, as in a dictation application. But usually the application interprets the text as an instruction to do something, as in a *command and control* application, where the caller might say *"sell all shares"* and a transaction takes place.

Input comes into the voice recognition engine from a microphone as an audio stream, over the Internet, or from a telephone. The recognition engine may have to adapt to low volume or background noise, matching the input against an *acoustic model*. It then uses data, statistics, and software algorithms to convert the incoming audio signal into a data format that is suitable for further analysis. Once the audio data is ready, the engine searches for the best match, using the words and phrases it already has (active grammars and vocabularies), returning a text string.

A *grammar* in a voice application uses a particular syntax, or set of rules, to define the words and phrases that can be recognized by the engine. A grammar can be as simple as a list of words, or it can be designed with more flexibility and variability so that a more natural language capability is achieved. Each new phrase, or utterance, is compared with the words and phrases in the active grammars, which can define several ways to say the same thing.

The design of grammars is important to achieving accuracy. One big tradeoff is whether the grammar is designed for a few speakers using a large vocabulary or many speakers using a limited vocabulary, or some combination of the two. Speaker dependence describes the degree to which a voice recognition system depends on a speaker’s individual voice characteristics. The recognition

engine can be trained to an individual's voice in a *speaker-dependent* system, as in most desktop dictation applications. Server-based middleware applications are designed to service a wide variety of random callers, so they cannot be trained to each individual. It would not be practical to employ speaker-dependent models in the voice middleware environment. These are therefore known as *speaker-independent* systems.

There are three basic types of dialog used in voice recognition applications. In a *directed dialog*, the application *directs* the user to perform a specific task and is in control of the interaction. This menu-driven approach leads the user to complete a function by asking for information at each turn of the dialog, expecting a specific response each time. So if the system were to say *"What is your last name?"*, it would expect a match from a stored list of names.

In a *mixed initiative* dialog, the user can take control, providing more input than what is being currently requested, and in a different order than expected. So the caller can anticipate the next prompt and reply accordingly, as in *"Jones, Cincinnati, Ohio,"* and the system can handle this, even though it required only the name initially. A mixed initiative dialog also allows for the system to take control such as when the caller does not provide all of the information that is needed, the system will prompt the caller for the appropriate information. For example, if the caller said "*Cincinnati, Ohio*" but did not provide a name, the mixed initiative application would recognize the input as a location and prompt the user to provide a name.

Barge-in (also known as *cut-thru*) lets a caller interrupt a prompt as it is playing, either by saying something or by pressing a key on the phone keypad. This is a popular feature for experts seeking a "fast path" around long or multiple prompts.

Natural Language Understanding (NLU)

In a *natural language* dialog, grammars can be designed to allow a more conversational interaction with an application, to mimic the way two people might talk to each other. In this type of application, a caller could say *"My name is Jones, I am in Cincinnati and I need the telephone number of Michelle Smith."* This requires much more complex grammars as well as significantly more application logic, because the application needs to extract meaning and context from each spoken utterance.

A natural language application might also use *context* (what was previously said) to understand a command. So in "*I need her cell phone number*", the application would know from an earlier response that "*her*" refers to Michelle Smith. In an NLU system, the speaker might change her mind, as in "*No, make that Michelle Jones.*"

Statistical NLU defines an NLU system that operates on statistical models based on large samplings of recorded speech. This is an adaptive model because not every possible set of words or phrases has to be recorded or analyzed. It deals with sentence and phrasing structure based on statistical models of the language and task domain to allow more free form commands and queries.

Grammar-based NLU requires a huge set of grammars to anticipate all of the things a caller might say. Grammar-based NLU has been viewed by many as natural language recognition (versus natural language understanding). Very complex grammars are developed based on analyzed speech samples, and anything the user might say must be in the grammar, or the system will not know what to do with it.

When comparing Statistical and Grammar models, we see that NLU and mixed initiative capabilities extend system design complexity significantly. Voice recognition must allow a much broader variety of input. Additional technology must be employed to be able to resolve abstractions and glean the meaning of arbitrary inputs. Applications must allow complex transitions from one type of transaction to another, and at the same time conversational history must be used to "inherit" parameters. Although explicit finite grammars can be used to create such systems, creating such grammars requires a high level of linguistic expertise over a long period of time. Statistical methods, based on training recognition systems from actual conversational input, are a better approach for complex applications, since there is less dependency on matching exactly what the user says in order to be understood, and not as much dependence on highly skilled linguists to implement and maintain the system.

In the grammar-based model, for limited grammar requirements, an application can be created and piloted quickly with a high degree of accuracy. This results-oriented approach helps validate the model. The promise of high usability and recognition can be quickly observed, temporarily masking potential future problems when applications become more complex in both scale and content.

Multi-channel access

The basic concept of multi-channel access is this: customers expect to be able to contact businesses to obtain information and to conduct transactions over multiple communication channels (thus, *multi-channel access*). At a minimum, they expect to be able to make contact over the Web and by telephone. Customers also expect to be able to use mobile and pervasive devices such as SmartPhones and in-car systems to access the very same business information and services, and they expect to be able to access this information 24x7x365. In other words, customers expect to be able to conduct business anytime, from any place, using just about any device. Furthermore, no matter how customers choose to make contact, they expect to be provided with a common and consistent experience. This is perhaps the biggest challenge for businesses as they deploy multi-channel access solutions.

VoiceXML

The *Voice eXtensible Markup Language*, or *VoiceXML*, is an XML-based markup language for distributed voice applications, much as HTML is a language for distributed visual applications.

VoiceXML is defined and promoted by an industry forum, the *VoiceXML Forum*, founded by AT&T, IBM, Lucent and Motorola and supported by around 500 member companies.

VoiceXML was designed to create audio dialogs that feature text-to-speech, digitized as well as prerecorded audio, recognition of both spoken and DTMF key input, recording of spoken input, telephony, and mixed-initiative conversations. Its goal is to provide voice access and interactive voice response (e.g. by telephone, PDA, or desktop) to Web-based content and applications.

VoiceXML brings the power of Web development and content delivery to voice response applications, and frees the authors of such applications from low-level programming and resource management. It enables integration of voice services with data services using the familiar client-server paradigm, and it gives users the power to seamlessly transition between applications. The dialogs are provided by document servers, which may be external to the browser implementation platform.

The VoiceXML specification, version 1.0, is available at no charge, through its sponsors, the VoiceXML Forum. It can be found at http://www.voicexmlforum.org.

The demand for voice applications

Voice applications are growing in certain key markets, in particular, banking, finance, securities, and in the communications industry, especially customer call centers.

Contact Centers

For many years, the primary point of customer interaction, or *Customer Relationship Management* (CRM), was the traditional, telephone-based call center. In today's business environment, the Internet has rapidly assumed a role as an alternative to the call center. This shift has not replaced the call center, but instead has levied a whole new set of requirements on the enterprise, as CRM moves to eRM, or *electronic Relationship Management*.

Voice technologies have begun to change the way self-service systems work. No longer are these systems limited by the rigidly-structured, keypad-driven dialogs that characterize traditional interactive voice response (IVR) systems, or for that matter, most Web sites. Voice recognition frees customers from these constraints, enabling them to interact with an automated, self-service environment in a way that is very similar to a conversation with a human agent.

Customers want to be able to choose between an automated self-service system or a live customer service representative. Customers will expect to be able to use mobile and pervasive devices such as SmartPhones and in-car systems to interact with enterprises. They will expect to use more advanced user interface technologies such as voice commands using natural language understanding for self-service transactions. They will want to receive and send a variety of messages, including

e-mail and voice mail, from a variety of devices such as handheld companions, SmartPhones, car clients, and digital set top boxes (STBs). Voice technologies allow a common solution, with a single, integrated application base, to be accessed in any of several ways, using keyboard or voice, from a variety of different devices.

Voice-enabled Web sites

For many businesses today, the company Web site is the primary -- and sometimes the only -- customer interface. As they grow, businesses would like to continue to develop and expand on this interface, while minimizing the number and types of IT architectures. Voice-enablement allows voice access to be added to existing Web sites without replacing or making major renovations to them, by simply extending existing function. This brings benefits to several business areas.

To an information technology (IT) executive, voice-enablement means that new voice-enabled Web applications that share a common business logic with their visual brethren can be created as an extension of existing applications. Applications can be extended and implemented in less time, and are easier to manage. To a business executive, this means that new voice solutions can be introduced to new markets quickly, providing a strong competitive advantage. To the financial executive, this means that current and long-term development and maintenance costs can be controlled by using existing personnel and infrastructures.

The graphical user interface of many of today's Web sites will be supplemented with a speech-enabled, or conversational user interface. In a visual site, the browser runs on the client, or desktop / laptop computer. In IBM's WebSphere Voice Server, the browser resides on the server.

Here's how a voice-enabled Web site works. Telephone calls come in on an ordinary telephone line to a *connection environment*. The primary purpose of the connection environment is to transfer the telephone voice data to an IBM WebSphere Voice Server. The connection environment can be IBM WebSphere Voice Response, Cisco, or Intel Dialogic (with more to come).

Next it is received by the IBM WebSphere Voice Server. The server runs the IBM voice recognition and text-to-speech software, multiple instances of the voice browser, and call management software that stacks and controls the incoming calls as they go through the system.

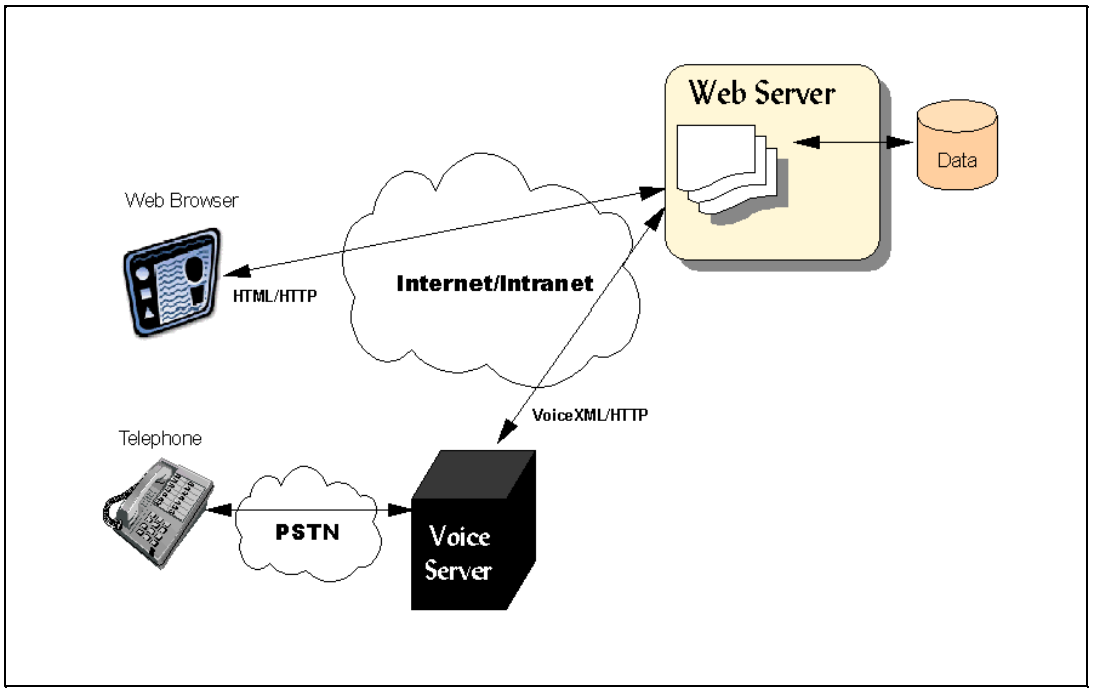
The recognition engine analyzes the audio stream and converts it to digitized text. The digitized text is then sent to the voice browser, which creates HTTP requests as necessary, and accesses the target information over the network. This is analogous to a visual browser, except that speech-enabled requests look for Web pages written in VoiceXML code, not HTML.

The application resides on a Web application server such as IBM's WebSphere Application Server, which contains pages of both visual HTML and VoiceXML code. Data is accessed from

various databases as needed. As each HTTP request is received, information is returned to the requesting server in the form of VoiceXML pages, which the IBM Text-To-Speech engine reads back to the caller.

The Voice Server in the Web Environment

The Voice Browser extends this paradigm by presenting the same Web information through a different media. Now, instead of displaying it visually (through HTML, graphics and text), the Voice Browser plays it to the caller audibly using VoiceXML. When the caller speaks a response -- the voice equivalent of clicking on a visual link -- the Voice Browser sends an HTTP request to the Web server, which may access the same back-end infrastructure, to return information, only this time it is VoiceXML instead of HTML.



It is very important to note that the Voice Browser and VoiceXML are not just providing a way of reading Web pages to a caller over the phone. Because a VoiceXML application is providing an audio-only interface, it needs to change the way information is presented. The point is that it changes the *presentation* of the information, not the information itself or the way it is generated by the Web server or the back-end system. VoiceXML provides a whole new way of accessing the same Web information, by providing voice access to Web data and services.

IBM WebSphere Voice Server

IBM offers a choice of voice products for enterprise customers, led by the **IBM WebSphere Voice Server.** This product provides server-based voice technologies that enable speech access to information, allowing anyone with a telephone to access e-business applications and data *simply by speaking.*

IBM WebSphere Voice Server V2.0 provides the base speech recognition, text-to-speech and telephony connector support for a Voice Server to integrate within the supported telephony platforms:

- WebSphere Voice Server connects to **IBM WebSphere Voice Response for AIX** and provides the speech technologies for customers using WebSphere Voice Response for AIX as their IVR platform. In this environment, voice applications can be developed in

any of the programming languages supported by WebSphere Voice Response for AIX (VoiceXML, Java, or State Tables).

- WebSphere Voice Server connects to the **Cisco** telephony platform and provides the speech technologies for customers wanting to implement VoiceXML applications within a Cisco Voice over IP (VoIP) environment on Windows 2000. In this environment, a separate Cisco VoIP gateway is required, which converts the incoming voice data to IP packets that can be transferred over the data network to the Voice Server.
- WebSphere Voice Server connects to the **Intel Dialogic** telephony platform. This solution provides the speech technologies for customers wanting to implement VoiceXML applications within an Intel Dialogic-based telephony environment on Windows 2000.

WebSphere Voice Server V2.0 supports the following languages:

- U.S. English
- U.K. English
- French
- German
- Spanish
- Italian
- Simplified Chinese
- Traditional Chinese
- Japanese

IBM WebSphere Voice Toolkit provides an *integrated development environment* (IDE) for developers to create voice applications. Using the Voice Toolkit, you can develop your voice applications on a desktop workstation before deploying them. Toolkit features include:

- Graphical VoiceXML application development environment
- Source VoiceXML and grammar generation
- Custom pronunciation generator
- National Language Support (NLS) to develop applications in any language the Voice Server supports
- Wizard for Reusable Dialog Components

IBM Reusable Dialog Components are a ready-to-use set of VoiceXML code objects that provide basic dialog functions needed by developers and make VoiceXML application development quick and easy. Examples include addresses, e-mail and credit card information.

IBM WebSphere Voice Server SDK enables developers to prototype and test VoiceXML applications on a desktop workstation using a microphone and speakers, without having to be integrated into a telephony infrastructure. Together, the WebSphere Voice Toolkit and the SDK provide a comprehensive, easy-to-use environment for developing and testing VoiceXML applications.

IBM WebSphere Voice Server Speech Technologies provide C APIs for integrating speech recognition and TTS into other telephony platforms. These APIs are provided on a per language basis for a subset of the supported WebSphere Voice Server languages**.**

IBM Natural Language Understanding (NLU) Technology provides the ability for callers to speak in a more natural, conversational style. IBM's NLU technology eliminates the limitations inherent in today's speech recognition systems. IBM can provide services to help you realize the benefits of this state-of-the-art technology.

The VoiceXML Browser

Up to this point, we haven't mentioned how the VoiceXML Browser fits into the WebSphere Voice Server. It is included in the WebSphere Voice Server product, but it is only used in those solutions where applications that are written in VoiceXML are deployed.

The VoiceXML Browser is a Java application. It uses HTTP over a LAN or the Internet to fetch VoiceXML application pages from a web application server. The Java console provides information on the prompts played, resource files fetched, and user input recognized, among other things.

The VoiceXML Browser contains a DTMF Simulator that enables you to simulate DTMF input (for example, PIN codes or passwords) on your desktop. The DTMF Simulator takes the place of a telephone and allows you to perform desktop debugging of voice applications without having to connect to telephony hardware and the PSTN. The DTMF Simulator and the Java console represent the only visual interfaces provided by the VoiceXML Browser.

When the VoiceXML Browser starts, it requests an initial VoiceXML document from the Web server. The VoiceXML document specifies an interaction (or "dialog") between the application and the user. The VoiceXML Browser interprets and renders the document, managing the dialog with the user by playing audio prompts (using text-to-speech or recorded audio), accepting spoken and DTMF inputs, and acting on those inputs. Each of these activities changes the state of the dialog. For example, a particular user response can cause the VoiceXML Browser to jump to another dialog in the same VoiceXML document, or to fetch and process a new VoiceXML document; as a result of this transition, the list of active grammars (and therefore the list of valid user utterances) changes. When a dialog does not specify a transition to another dialog, the VoiceXML Browser exits and the session terminates.

The VoiceXML Browser component of the IBM WebSphere Voice Server SDK is the implementation of the interpreter context as defined in the VoiceXML 1.0 specification.

The Voice Browser recognizes VoiceXML, which can be generated by WebSphere Studio, Java Server Pages (or any other server-side markup generation such as CGI or ASP) and/or manually written. The server is compatible with and plugs into a company's existing telephony / Web infrastructure.

Creating a Voice Application

Designing a voice user interface involves at least two levels of design decisions. First, you need to make certain high-level design decisions regarding system-level interface properties. Only then can you get down to the details of designing specific system prompts and dialogs.

High-level design decisions

There are several high-level user interface design decisions you need to make before getting into task or interface details. These include:

- The appropriate user interface - Will you use speech? DTMF? Both?
- The type and level of information - What kind of information will you present? How much information will you present?
- Barge-in - Do you need to support barge-in?
- Recorded prompts versus synthesized speech - Will you use text-to-speech prompts or prerecorded audio prompts? Or some combination of both?
- Use of audio formatting - Are there specific audio techniques that you can use to enhance the information your application is presenting?
- Simple versus natural command grammars - Do you want the caller to be able to say simple words and phrases or do you want the caller to be able to speak more conversationally?
- Prompt style - Will you use terse, concise prompts or more conversational, friendly prompts?
- Always-active navigation commands - Should you support a consistent set of commands (such as help, exit, cancel) that are active at all times during your application?
- Style of help - Will the user have to explicitly ask for help or will the application determine that the caller needs help and take the initiative to provide it?

Low-level design decisions

Once you've decided on the high-level properties of your system, several "low-level" issues are considered, especially regarding specific interaction styles and prompts. These include:

- Consistent sound and feel

- Consistent timing
- Consistent dialogs
- Appropriate prompt and menu construction
- Consistent error recovery

Advanced user interface topics

You don't have to stop there, though, if you have the resources and motivation to create a more advanced user interface. Some advanced user interface topics include:

- Customizable expertise levels
- Appropriate selection of user interface metaphors
- Approaches to controlling the "Lost in Space" problem
- Audio list management
- Additional opportunities for exploiting audio formatting

For more information about designing effective voice user interfaces, refer to the *IBM WebSphere Voice Server Programmer's Guide.*

Application development

As an application developer, you are probably very familiar with the steps involved in designing, developing and testing a graphical or even textual application. Voice application development follows a lot of the same principles, but it also brings some new things into the picture. Let's take a look at the elements that are new to the development of voice applications.

First of all -- and perhaps most importantly -- *you need to view your application as a dialog*. Your application interacts with the caller through voice recognition, DTMF, prerecorded audio and text-to-speech. You need to identify what your application is going to say to the caller (the application prompts) and you need to identify what the caller can say to your application (the grammars). Once you get the input from the user, your application needs to do something with it.

Identifying what a caller can say

One of the first steps in the voice application development process is deciding what the callers will say. Are there different parts of the application where different things will be said? Are there things callers will always want to say, regardless of where they are in the application (for example, "*help*" or "*cancel*")? Do you want to support synonyms - more than one way of saying a command? Do you want to support a more natural way of saying something versus providing specific command sequences (for example, "*Can I get my checking account balance please*" versus "*checking account*")? These are just some of the considerations you should make when identifying what callers will be able to say to an application.

Once you have decided what words and phrases the caller will be allowed to say at each point in your application, you define the grammar(s) for these words and phrases. You can and probably will have multiple grammars for your application. This helps you group and combine things the user will say to your application.

A grammar can be as restrictive or as flexible as the application and your users need it to be. Of course, there is a tradeoff of recognition speed and accuracy versus the size of the vocabulary. You may want to experiment with different vocabularies to validate a design that best matches the requirements and expectations of your users.

There are a few different ways you can define your application grammars, depending on the voice middleware platform you are using. Let's take a look at a simple example. Take the following very simple dialog:

***Application**: Would you like coffee, tea, milk or nothing?*
***Caller**: Coffee.*
***Application**: Your coffee is coming right up.*

You'd like the caller to be able to say either "coffee," "tea", "milk", or "nothing," so you need to define a grammar that contains those words. In the WebSphere Voice Response product, you might define the following grammar**:**

```
<drink> = coffee
        | tea
        | milk
        | nothing.
```

In a VoiceXML version of this application, your grammar could be written as:

```
<grammar>
coffee | tea | milk | nothing
</grammar>
```

Identifying what the application will "say"

A voice application is an audio-only interface. You need to identify what your application is going to say at any given point in the dialog with the caller. This is called defining your application prompts.

Prompts can be prerecorded and played to the user during the phone call. They can also be played using text-to-speech. Text-to-speech is a critical component of your application when you need to present "unbounded" information to the caller (say, the results of a backend database query). By its very definition, you must know the text of a prompt beforehand to prerecord it; however, you

don't always know what you need to tell the caller in advance. In these cases, you absolutely need text-to-speech to present this information to the user.

Handling the recognition results

Once your application receives text from the speech recognition software, it needs to do something with it. If the caller says "*checking account*," the application should read the caller's current checking account balance back to her. Based on the results of the recognition, your application may request another VoiceXML page to be fetched from the web application server, or a backend database query will occur and the results will be presented back to the caller, or the dialog may continue so that your application can get more input from the caller.

Furthermore, your application needs to consider "normal" speech recognition errors, such as when the user says something that is not in one of the active grammars, or the caller speaks indistinctly, or the caller doesn't say anything at all.

Application development models

Application developers typically have their choice of languages and tools with which to implement their applications. Voice application development is no different. IBM offers several application development models, including languages and tools, through which you can develop voice applications:

- **VoiceXML** - An industry standard language for providing conversational access to web-based data. This language is supported on all WebSphere Voice Server platforms.
- **Reusable Dialog Components** - A set of VoiceXML code objects that provide basic functions needed by developers and make VoiceXML application development quick and easy.
- **WebSphere Voice Response Beans** - Allow you to develop IVR applications that work with a WebSphere Voice Response product, on AIX or Windows or both. You can develop applications on platforms that support Java, using either a visual builder, such as IBM VisualAge for Java, or the Java Development Kit.
- **WebSphere Voice Response for AIX State Tables** - The "native" programming model of WebSphere Voice Response for AIX.
- **WebSphere Voice Response for Windows T-REXX** - The "native" programming model of WebSphere Voice Response for Windows.

Other IBM Voice Products

The IBM WebSphere Voice Server, IBM WebSphere Voice Response, and IBM Message Center are all part of the IBM WebSphere software platform -- a comprehensive set of integrated, e-business solutions. The WebSphere software platform is designed to deliver flexibility for

growth, supporting a variety of diverse, integrated e-business environments, enabling Web-based business applications with industry standards such as Java, XML, and VoiceXML. Multi-channel access solutions include customer relationship management (CRM) / Interactive Voice Response (IVR) products such as IBM WebSphere Voice Response and IBM Message Center.

IBM WebSphere Voice Response

IBM WebSphere Voice Response is a highly-scalable voice-processing platform with interactive voice response capability. A versatile, open platform, IBM WebSphere Voice Response can enable multiple, concurrent voice applications such as self-service access and updating of information, voice messaging and fax. Customers and employees have direct access to services and information 24 hours a day, 7 days a week, by telephone or through the Web.

IBM WebSphere Voice Response connects callers to Customer Relationship Management (CRM) applications to enable affordable self-service transactions by telephone. It can be used to provide new or out-of-hours services, and to reduce call center waiting times and costs. It is used by network service providers, CRM service providers such as banks, and by business support functions such as human resources, registration or appointment management. Voice recognition can be combined with WebSphere Voice Response for more comprehensive and flexible IVR solutions.

IBM WebSphere Voice Response supports open standards, including JavaBeans. VoiceXML and VoIP support are also available.

IBM Message Center

IBM Message Center is a unified messaging system that runs with IBM WebSphere Voice Response for AIX. With up to 500,000 mailboxes on a single system, it can manage employee and customer voice mail, e-mail and faxes, for access over the telephone or the Internet. Whether it's a service provider looking to deliver a high-quality unified messaging service, or a business looking to make life easy for customers and employees, IBM Message Center is a solution that simplifies messaging and improves responsiveness. Its robust high availability virtually eliminates system downtime.

Embedded Systems

Pervasive computing is driving the Mobile Internet, fueled by a variety of mobile devices. This trend will continue to gain momentum as devices get smaller and more embedded. Voice is the interface of choice for these mobile devices. Today IBM is enabling a wide range of embedded mobile solutions with its *IBM Embedded ViaVoice, Multiplatforms Edition* product. This toolkit provides developers with resources to create voice-enabled, embedded mobile solutions.

IBM's embedded platform can support a variety of real-time operating systems and microprocessors making the development of robust mobile voice solutions easy and practical for device and application developers.

Voice-embedded technology offers tremendous promise for automobile designers. In a mobile automobile environment, voice is the most natural interface for a driver or passenger to use when accessing information or interacting with their automobile. ViaVoice technology has been integrated into demonstration vehicles, monitoring conditions and reporting problems to passengers through a wireless connection to a service center. The service center can then give the driver advice about problems and, if required, dispatch emergency services. Passengers can also use their voice to call anyone in their address book. They can surf the Web and have information read to them, conduct e-business transactions, and create or receive e-mail.

Support for WebSphere Voice Server Developers

Education :

http://www-4.ibm.com/software/ad/aimclasses/

For more information:

WebSphere Voice Server Support:
http://www-4.ibm.com/software/webservers/voiceserver/support.html

WebSphere Voice Server Library:
http://www-4.ibm.com/software/webservers/voiceserver/library.html

WebSphere Voice Server for WebSphere Voice Response:
http://www-4.ibm.com/software/speech/enterprise/wsvs-dt.html

WebSphere Voice Server for WebSphere Voice Response library:
http://www-4.ibm.com/software/speech/java/vxml/docs.html

Notices