Introduction to Speech .NET

By Robert Delwood

Introduction

Microsoft's .NET Speech technologies add speech recognition and spoken output access to Web applications. The technologies combine telephony access and Web services into a single system and a single code base. There are three factors driving this initiative.

First, companies currently must maintain two separate functional areas: one for Web access and another for telephone access. The cost of operating and staffing a call center can be enormous and most companies now use an interactive voice response (IVR) system to reduce costs. However, it is clear that IVR systems offer only a small portion of the functionality of a Web site. The systems are usually proprietary, hard to program, and share no infrastructure with the company's Web site. In addition, the operating costs of such an IVR system are still significant.

Second, speech technologies are being offered as an enhancement to most IVR systems. Companies can automate more of the call center operations. However, current speech recognition systems are even more costly to develop and maintain than IVR systems alone.

Third, at the same time a long-term trend is developing: the proliferation of mobile communication devices. Today, there are as many mobile phones are there are traditional phones. Coupled with that is the fact that many of these mobile devices have significant computer processing capabilities. Many have powerful CPUs, extensive RAM, and some include Web browsers. These new devices blur the differences among existing technologies. Smart phones have a screen display; PDAs have telephony capabilities; laptops and Tablet PC computers have both in addition to traditional computer capabilities.

In the past, these differences may not have been an issue. Customers may have been satisfied using a phone to call an employee, retrieve an account balance, or check e-mail. The same customer may have been satisfied accessing the company's Web site from a desktop computer to check the status of a shipment, schedule an appointment, or purchase a product. The logical extension of these experiences is allowing customers the same access to information regardless of the actual device they use.

Microsoft is therefore building a .NET Speech SDK and an accompanying server addressing these trends by:

- Handling both Web browser access and telephony access (voice only) using a single Web application.
- Enabling the development of powerful speech interfaces using Web programming techniques. In addition to a robust development environment, a more natural speaking approach can be used. Customers can say "I'd like to book a flight from Seattle to Atlanta this Thursday leaving after 1:00 p.m." rather than clicking buttons for each of the city, time, and date.
- Enabling multimodal access. Customers with powerful devices may use more than one mode (voice, keyboard, stylus, and mouse, for instance) interchangeably to enter information. For example, the customer may speak to the device to request a list of available items and then use a stylus to scroll through that list and tap the desired item.

Technical Overview

Microsoft achieves these capabilities described above with a comprehensive system of four components.

- An ASP.NET server using ASP.NET speech controls. When a browser accesses the Web site, the ASP.NET server generates the appropriate code. A speech-enabled browser receives the HTML, embedded scripts, and Speech Tags. Speech Tags are Microsoft's implementation of Speech Application Language Tags (SALT), described in the SALT section below. SALT is an HTML-based tag set controlling speech capabilities. Embedded scripts control the dialog flow for voice-only interactions.
- A telephony server. This server's software works with a telephone board to connect to the telephone network. It also incorporates a voice browser for interpreting the voice-only related HTML, Speech Tags, and scripts. It can serve a large number of simultaneous calls with each call being given its own voice browser instance.
- A speech server. This server carries out requests for speech recognition and speech synthesis, and manages a prompt database. The prompt database is a collection of recorded human-spoken responses (rather than a purely synthesized voice) for a more natural sounding playback.
- A multimodal client browser. This add-on to Internet Explorer and pocket Internet Explorer interprets Speech Tags. If the host device is sufficiently powerful (such as a laptop PC or a Tablet PC), the browser add-on could access the speech engines locally (that is, loaded on the device itself) through Microsoft's Speech API (SAPI). However, if the browser is running on a lower powered device (such as smart phone or Pocket PC), or the device does not have the engines installed locally, the add-on passes a stream of features extracted from the audio over a wireless network to a speech server.

Microsoft is building a .NET Speech SDK that provides developers with four tools. These tools are Microsoft Visual Studio .NET add-ons and can be accessed using the Visual Studio .NET editor. One tool creates and manages the ASP.NET controls graphically rather than programmatically. Another tool records, edits, and manages voice prompts for speech output. The third tool provides debugging support. A fourth tool creates and edits speech recognition grammars. Grammars are described below.

An example at the end of the article will demonstrate the workings of the complete system. The full architecture is displayed in Figure 1.

(Insert Figure 1 here)
Figure 1: The .NET Speech applications architecture

[Material deleted]

Speech Application Language Tags (SALT)

The key to Microsoft's .NET Speech technologies is the newly emerging standard of SALT. The SALT standard will be specified by the SALT forum, which was founded in October of 2001 by Cisco, Comverse, Intel, Microsoft, Phillips, and Speechworks.

SALT is a small set of XML elements extending HTML or XHTML. SALT could be used to extend other languages such as cHTML, Wireless Markup Language (WML), or any Standard Generalized Markup Language (SGML). SALT adds speech recognition, text-to-speech, plays back prompt files and incorporates dual tone multi frequency (DTMF) and call control services for voice-only browsers. The tag set is intended to be a royalty-free and platform independent standard. Ultimately the SALT forum intends to submit the specification to a standards body such as the World Wide Web Consortium.

SALT extends HTML with only five top-level elements.

- <prompt> specifies output and plays back the prompt.
- <reco> executes and processes speech recognition
- <grammar> specifies the grammar files to the used. Grammar files are the list of words from which a valid recognition is selected.
- <bind> associates a speech recognition to a specific page.
- <dtmf> configures and controls the dual tone multi frequencies (DTMF) in telephony applications. This allows call controls and push buttons on the handset.

A sixth object is planned and will address call controls. The .NET Speech SDK will have additional information.

Until the SALT forum officially publishes the SALT standard, the .NET Speech SDK uses a set of elements called Speech Tags. Speech Tags is Microsoft's initial implementation of the SALT proposal.

Speech Tags

Developers may use Speech Tags in a combination of two ways: hand authoring or through speech controls. With hand authoring, they may write Speech Tags much as they would conventional HTML using the tags directly.

The following is a small code sample.

```
<!—- HTML -->
<html xmlns:stags=”urn:microsoft.com\speech”>
<input name="txtBoxCity" type="text" onpendown=”recoCity.Start()”/>

<!—- Speech Tags -->
<stags:reco id=”recoCity”>
     <stags:grammar id=”g_city” src="city.xml" />
     <stags:bind targetElement="txtBoxCity" value="//city" />
</stags:reco>
</html>
```

Sample 2: Sample speech control code

The actual recognition begins by an OnPenDown event. The sample collects a destination city from the user's speech and assigns it to a textbox (named txtBoxCity using a grammar file named city.xml.

Speech Controls

As an alternative to hand authoring Speech Tags, developers can author using ASP.NET speech controls. Speech controls are a special form of Web controls. In general, Web controls are those that have built in events, methods, and properties. Web controls encapsulate common tasks such as displaying calendars or validating user input. The Web server maintains state information and persist data during the control instance's lifetime. Web controls can then provide consistency with standard methods, events, and properties. Microsoft's .NET Speech SDK extends their use with companion controls known as Speech Controls. Speech Controls modify the behavior of existing controls by way of document object model (DOM). There is no need to rewrite large code segments. As a result, the page can define more precise speech interactions.

For example, for a multimodal application, the designer intends to add speech recognition to a Web control such as a textbox. The user simply clicks inside the textbox and speaks while holding down the mouse button, stylus, or similar device. After releasing the device, the recognized text appears in the box. This approach is also called tap and talk. The page would already have a Web control for the textbox. A speech control is then added to the page. Among the control's properties is `TargetElement`. Add the textbox ID as the property value in order to speech-enable that textbox. See Sample 3 for a code example. Other properties include events such as OnMouseDown() and OnReco(). If the default handling is not sufficient or applicable, these events may be overwritten for the individual control.

It is important to note that the basic Web control is not modified directly by the companion control. The basic speech companion control in .NET Speech SDK is called QA (indicating question and answer). This control has five basic properties.

- The Question object adds voice output.
- The Answers object allows speech recognition.
- The Statement object is for fixed output such as "Welcome," or for reading e-mail.
- The Command object issues commands, such as Help, Repeat, or Cancel.
- The Confirm object verifies user responses.

The following code demonstrates programming a textbox control (named *`Textbox1`*) with speech. It asks the user for the type of credit card. The grammar source (the `<Grammar>` tag) is the XML file list containing valid responses for credit card names.

```
<Speech:QA id="creditcardQA" runat="server">
     <Prompt>What type of credit card will you be using?</Prompt>
     <Reco id="reco1">
     <Grammar src="c:\creditcardlist.xml"/>
     </Reco>
     <Answers>
     <Answer id="A1" XpathTrigger="/SML/cardName"
TargetElement="Textbox1" />
          </Answers>
</Speech:QA>
```

Sample 3: Sample speech control code

The speech QA control is a separate component directly attaching to existing controls rather than modifying those controls directly. For example, in Sample 3, the *`Textbox1`* control is a standard Web control for a textbox. However, the speech control supplements it with a grammar. The speech result is returned to the *`TargetElement`*. The companion control approach allows for seamless integration with the ASP.NET framework and at the same time allows custom controls to be built without restrictions. The Web application developer adds these features during the page design rather than relying on hard coded features in the ASP.NET controls.

Recognition Results: Semantic Markup Language

The speech recognition results are returned to the application in an XML format called the Semantic Markup Language (SML). SML includes the complete recognizes text as well as a representation of the semantic information it contains. Using the simple grammar from Sample 1, a travel application might receive a recognition result string of “please fly to Houston.” The application needs to know which word is the city. SML marks this for the application. The following is an example of a possible recognition result for that utterance.

```
<SML text=”Please fly to Houston” confidence=”80”>
     <CityName text=”Houston” confidence=”90”>Houston</CityName >
</SML>
```

Sample 4: Sample SML results.

The element is identified based on the rule name or property name. In this case, the property name is *`CityName`*. After the speech server returns the recognition result, the application parses this result, specifically looking for the *`CityName`*.

[Material deleted]

An Example User Experience

To illustrate the power of the speech-enabled Web, here is a simple (and hypothetical) user experience covering both a simple telephone, and a smart device with a browser that combines speech and GUI.

Amy is a busy software executive who is always on the road. She uses her PDA, notebook PC, cell phone, and many other devices to coordinate her schedule.

Amy logs onto her corporation’s travel site from her Pocket PC to begin planning her trip. With her stylus, she can tap an icon that allows her to speak to that section of the Web page. The first icon she taps on is next to the label “Where do you want to travel?” She speaks to the Pocket PC, “leaving from Seattle going to Dallas-Fort Worth.” She then taps a speech icon labeled “When do you want to travel?” and speaks again to the PC, “leaving on December 13th.” The computer updates the appropriate fields on her screen. Amy sees that she forgot to say the time of day she wishes to travel. She taps the speech icon next to the time field and says, “Leaving Seattle at 8:00 a.m. and arriving in Dallas at 4:00 p.m.” She continues to fill out the rest of the page for all of the required information right down to airline and seating preference. After Amy enters all the necessary information, she taps Submit. Several flight options appear on the screen. After reviewing the options, she selects one, but decides that she is not ready to purchase the tickets and saves the itinerary in her “my trips” folder until she is ready to buy the tickets.

A few hours later, the Amy decides she is ready to purchase her tickets. Unfortunately, she is driving a car and will not have access to her computer for some time. Instead of using her Pocket PC, she calls the travel site from her hands-free cell phone in her car. The following conversation would be typical:

Speech server: Hello Amy, Welcome to corporate travel by phone, please say “search flights” or “my trips.”

Amy: My trips

Server: Please say or enter your passport Identification number.

Amy: 555555

Server: Amy, you have one itinerary on hold for you. To review this itinerary before …”

Amy: Review the itinerary.

Server: The itinerary is for one ticket from Seattle to Dallas-Forth Worth flight number 9000 on North by South West Airlines. Departing Seattle at 8:00 a.m. and arriving Dallas-Forth Worth at 4:00 p.m. Would you like to order the tickets or cancel the itinerary?

Amy: Order the tickets

Server: That will be $900. Would you like to use the credit card data and mailing address stored in your Passport account?

Amy: Yes

Server: Your tickets will be sent to you in three working days.

Behind the Scenes in Amy’s User Experience

The Web server application is written using server side code. A conventional GUI-only Web browser accessing the Web site would have the ASP.NET server generate a standard HTML page. However, Amy’s Pocket PC is running pocket Internet Explorer with the Speech Tags add-on. Because of the new capabilities, the server generates an HTML page along with Speech Tags, SALT and scripts.

When Amy taps the speech icon on her Pocket PC, the event connects her browser to a speech server running at the corporate travel site. Data representing her speech utterance is streamed to the server. At the same time, the browser sends the speech server a pointer to a grammar file specified in the Web page containing the phrases Amy could speak.

The speech server attempts to find the best match to what she said in the grammar. For example, it might determine that she asked for origin airport of SEATAC and destination airport of DFW. The results are returned to the browser with an XML representation of the semantic information in her utterance. The browser then populates the relevant fields. When Amy eventually submits her plan, the browser sends the information back to the Web server. Whether Amy spoke, handwrote, or typed the information does not matter; the Web server simply receives the information.

When Amy disconnects, the ASP.NET server stores session information to use later. For example, when Amy calls her corporate travel’s toll-free phone number again, a server connected to the incoming telephone line answers the call and creates an instance of a Speech Tags voice browser. Recognizing Amy’s mobile phone number and knowing that the toll-free phone number corresponds to the corporate travel site, the browser requests a customized home page from the Web server of the corporation’s travel site. The code on the Web server recognizes that this is a voice-only browser and sends HTML, Speech

Tags, SALT, and ECMAScript to the browser. The ECMAScript directs the voice browser to the correct prerecorded announcement to play first (i.e., which form to activate first). When Amy replies, the server sends a request to the corporate travel speech server along with a pointer to a grammar file. When the voice browser receives the results, it populates the relevant field, and the ECMAScript program determines whether to play another prompt or send the result back to the Web server.

The developer at corporate travel department wrote the application using Visual Studio .NET tools provided in the Microsoft .NET Speech SDK. One tool allowed the developer to create the speech controls, set their properties, and associate them with general ASP.NET controls. To make the output speech more natural sounding, the corporate travel site decided to record their own prompts and used a .NET Speech SDK tool record, edit, and manage those prompts. And lastly, the developer used the grammar editing tool to create, edit, and manage the grammar files.

Conclusion

The Microsoft .NET Speech SDK can be used to create a variety of speech-enabled Web sites. With the technological promise of mobile devices and the proliferation of wireless capabilities, it will be important to unify the user experience. Customers are expecting to access information independent of the type of device they use. .NET Speech technologies addresses those needs in ways no other solution can. The expected release of .NET Speech SDK is November of 2002. In anticipation of these new capabilities, companies may wish to start implementing Web sites with, or gain experience with, ASP.NET as this will be the core of .NET Speech applications.

Comment: Not Speech Tags?