Guidelines for a VoiceXML Solution Using WebSphere Transcoding Publisher

Marshall Lamb, WebSphere Transcoding Publisher

Brenda Horowitz, WebSphere Voice Server

Overview

IBM WebSphere Transcoding Publisher (WTP) provides the means for making existing, Web-based content authored in HTML and XML available to various clients devices which do not accept HTML and XML as input. Employing components called transcoders, WTP does not modify the original document, but instead mutates a copy of the document as requested by the client into a new markup language, such as WML (wireless markup language) for WAP (wireless access protocol) phones, or even more simplified HTML for slower networks with lower bandwidth.

The Voice eXtensible Markup Language (VoiceXML) is an XML-based markup language for creating distributed voice applications in which the input and/or output are through a spoken, rather than a graphical, user interface. VoiceXML is an emerging industry standard that has been defined by the VoiceXML Forum (http://www.voicexml.org) and accepted for submission by the World Wide Web Consortium (W3C) as a standard for voice markup on the Web. Users can access deployed VoiceXML applications anytime, anywhere, from any telephony-capable device.

In most situations, you will want to code your voice applications natively in VoiceXML using the IBM WebSphere Voice Server SDK or an equivalent product. If, however, you fall into one of the following categories, you may find using WTP to be a more productive solution, even in combination with a native VoiceXML application:

- You already have an existing visual web site (in HTML) or data in XML
- You want to present the same basic data on multiple output devices, one of which is a telephone.

To WTP, VoiceXML is just another format, and the voice browsers which interpret VoiceXML into spoken text are just new devices to support. There are two ways to use WTP to transcode information into VoiceXML:

1. Using XSL stylesheets to convert XML directly to VoiceXML
2. Using the HTML-to-VoiceXML transcoder for converting entire HTML pages, or portions of pages, into VoiceXML

The HTML-to-VoiceXML transcoder does what its name implies: converts HTML to VoiceXML. When a VoiceXML-capable client requests an HTML document through WTP, WTP understands that the device expects VoiceXML content by the AcceptHeader portion of the HTTP request, converts the requested HTML document into VoiceXML using the transcoder, then sends the resulting document back to the client. The HTML-to-VoiceXML transcoder does not actually render the VoiceXML document into spoken text; that is the responsibility of the voice browser. When you are ready to deploy your VoiceXML applications, we recommend that you use the IBM WebSphere Voice Server. However, other vendors' products with VoiceXML 1.0-compliant browsers should also work. To access the applications, users simply dial the telephone number that you provide and are connected to the corresponding voice application.

Challenges of voice versus visual

Most dynamic content that is being considered for a voice application has either already been rendered visually, such as through HTML pages, or has no interface representation at all, such as a collection of XML documents or data from a database. The first thing to understand is that a voice interface implies a completely different set of rules and expectations for the end user than traditional visual interfaces. For instance, visual elements which may make the document easy to use (navigational bars, "twisties" or expand/collapse nodes) or visually appealing (embedded graphics, animation, color scheme) may not work at all in a voice application. Also, visual elements which are not germane to the subject of the document, such as advertisements or company banners, get in the way of the end user's ability to get to the core content quickly and easily. The elements of a good interface are the same, though, regardless of whether it is viewed or listened to: it must be clean, intuitive, efficient, and productive.

Dynamic data which is in some other, non-visual representation, such as XML, saves you from these visual elements. Even so, an interface must still be built for this data. Understanding the capabilities of the voice browsing system and the user's expectations for the interface is paramount to building a usable voice interface, especially if your background is in building visual Web applications.

Let's explore in more detail some of the most important differences between a visual application and a voice application.

Mental Load

One of the largest and most important differences between a visual and audio interface is the amount of data which can be represented in, and consumed from, each. Web (HTML) pages often offer a gross amount of information, but we have been trained to exploit common navigational elements to drill to the desired data. The page is in front of you for as long as you need it. You can spend as much time as you need analyzing the information. You can even quickly move back through previous pages to review what you've seen.

Contrast this with a voice application. Users typically access such an application through a phone, meaning that once they've heard it, it's gone, and is usually difficult or time consuming to get back. Information in a voice application is transient just like any human-to-human spoken conversation. Couple this with the fact that the user's eyes are typically focussed on something else, which means that they are easily distracted and will find it very difficult to concentrate on large amounts of data.

Voice application designers need to limit the amount of information presented to the user, avoiding overpowering short term memory. Remember, there's no document to view and no back button. For example, where a large list of options may be presented in a visual form, common usability guidelines for voice dictate that around 7 (+/- 2) options in a list is best. Thus when transcoding HTML to VoiceXML using WTP, extra care needs to be taken in reducing the complexity of the data. We'll explore methods for doing this later in the section titled **Using annotation to simplify the page**.

Navigation and Error Recovery

We are all familiar with navigating HTML pages. The common use of navigation bars, self-describing hyperlinks, and the Forward and Back buttons make it easy to get where you need to go and understand where you are. If there is an error, the Back button gets you back to a safe location where you can try again.

For voice applications, it is not obvious to the user where he is in the application. Where navigation is concerned, you need to provide a consistent approach for traversing menus and pages of data, such as always offering a link to content first, followed by a link back to the main menu, then by a link to exit the application. The link wording needs to be concise (one to three words) yet descriptive and representative of the current area of the application. In visual Web pages, you can intersperse links to distantly related material and not cause the user to lose focus of the subject matter. Try this in a voice application, the user can quickly lose track of what he has heard and forget where he is, causing him to suffer through the rest of the links to find the one that repeats the options or returns to the previous menu. You need to be restrained in how much data is presented to the user and ensure that all of the data is closely related in subject.

Error recovery is often simpler in a visual application. You recognize yourself typing something incorrectly and fix it before it is submitted, or resubmit input by going back and trying again. You click on the wrong link, so you use the Back button to try another one. With voice applications, you not only have to contend with providing the wrong input, but with the system's ability to understand what you say.

Use of voice applications is more error prone. They need to be designed in such a way to make it easy for user to retry an input, or limit the user's choices for input to minimize the risk of error. In a visual representation, data is randomly accessed, and input can be given at most any time. In a voice application, data is read sequentially and input (voice recognition) can only be processed at certain times. "Barging in", which refers to the ability for a voice application to allow the user to interrupt the data being read with a request, as well as controlling when voice input is allowed in general, is a function of the voice browser, not VoiceXML (and thus the HTML-to-VoiceXML transcoder). Even so, how (and how much of) the data is being converted to VoiceXML is under the control of the HTML-to-VoiceXML transcoder and of WTP in general.

Some of the navigational difficulties related to voice applications are taken care of by the HTML-to-VoiceXML transcoder as it keys off certain HTML tags for deciding navigation points, but it is up to you to ensure that the result is usable and the visual elements that the transcoder builds the navigation from are present in your source documents. More on this later in the **HTML to VoiceXML transcoding capabilities** section.

Wording and Input Processing

Modern speech recognition processing is optimized around recognizing known words and phrases. It is easier for the software to distinguish between spoken phrases made up of several syllables and varying phonetic characteristics than between spoken letters, such as when attempting to spell a word. It is easier, for example, to distinguish between "Closest theatre" and "Closest restaurant" when spoken as a phrase than it is trying to spell "theatre" or "restaurant". It can be difficult for voice recognition software to tell the difference between "T" and "E", especially if the user is on a cell phone in a public place or a moving vehicle, but it is easier to recognize a specific phrase.

When a voice application requires text input, that input must be limited to a well-understood and bounded grammar, or specification of valid input. When the grammar can be limited to a (short) list of words from which to choose, such as through a navigation menu or selection list, then the voice browser needs only to "concentrate" on understanding those set inputs. That grammar must either be generated as part of the page, or dynamically generated as part of the application logic, such as from a database.

Requiring the user to spell an entry, such as a name, part number, restaurant name, can increase the rate of unrecognized input or erroneous results. The possible exception is numeric input, where the grammar is limited to the numbers 0 through 9. These ten characters are more or less phonetically unique and most voice recognition systems have less difficulty differentiating between them.

Unfortunately, most HTML form input is not as bounded. Because all visual Web browsers have some type of keypad or keyboard attached to them, almost anything can be presented as input. When transcoding HTML to VoiceXML, one of the most daunting tasks is to ensure that possible input is well bounded and defined so that accurate grammars can be produced and error tendency reduced. For example, if an HTML page requires that the user type in their state of residency, you can use WTP to turn that text input into a select list of 50 entries, which can have an associated grammar of 50 words and phrases. The list of possible entries are used as the basis for the grammar in recognizing the response. Better yet, if you have access to the HTML source, you can rework the original HTML page to have a select list to begin with, which improves both the VoiceXML produced by WTP as well as the usability of the original HTML page. We'll cover more in the **HTML design guidelines** section.

Voice application design assumptions

Because of the limitations in speech recognition and the requirement for simplicity, voice application designs tend to be well-defined and bounded. You don't, for example, provide a URL to browse over the phone since the system's ability to understand you while you spell the address and your ability to correct mistakes are severely limited. Therefore, navigation is restricted to a finite set of pages and a finite set of links between them. This makes it easier for the voice application designer to predict input and control the grammars used to recognize that input.

When accessing HTML source to build, or incorporate within, your voice application, the same assumptions should be made that you will only access a finite number of pages. If you have access to the source behind these HTML pages, then you have control over the layout of the page and can apply some of the techniques described later in the **HTML design guidelines** section to improve the "transcodability" of these pages. You also will have a greater chance of knowing or predicting when this source could change and can plan alterations in WTP as a result.

If you do not have access to the source, such as if you are mining data from external Web sites, then you run the risk of having to react to changes to these sites, which if they happen without warning, could mean long downtimes for your voice application as you catch up. In this case, it is in your best interest to focus on data sources that have a well designed format with a history of preserving document structure. You'll need this since you don't have the luxury of cleaning up the HTML source yourself and will be relying on WTP to do all of the work.

We'll now focus on the HTML-to-VoiceXML transcoder's capabilities, as well as those of WTP in general, for generating voice application content, to be followed by HTML style guidelines for filling in the gaps that WTP cannot cover.

Fitting Transcoding Publisher into the picture

Before describing how WTP transcodes source into VoiceXML, let's explore the different ways WTP can be employed to do so.

WTP usage scenarios

Transcoding Publisher can produce content for voice applications, or even the voice application itself, in three ways, or scenarios: converting XML data directly to VoiceXML through the application of XSL stylesheets, converting a Web site into a VoiceXML application, and mining Web content for reuse in a VoiceXML application.

Converting XML to VoiceXML

For XML source, WTP can employ its XMLHandler transcoder to apply user-developed stylesheets to XML data to convert it to VoiceXML. This process does not involve the HTML-to-VoiceXML transcoder. If you decide to render your XML data directly into VoiceXML, you do not have to be concerned with the HTML style guidelines and the annotation function covered later.

A twist to this approach which does make use of the HTML-to-VoiceXML transcoder is when the stylesheet applied to the XML produces HTML as a result, which is then converted to a device-appropriate format such as VoiceXML. This approach uses HTML as a common intermediate format from which more specific formats can be more easily derived.

Without a common intermediate format, converting XML data to several possible presentation formats implies having to generate a different stylesheet for each resulting markup language, and a new set of such stylesheets for each XML document. If there are M source XML documents and N devices to support, M x N stylesheets must be created! This effort can be simplified by creating a single stylesheet which converts the XML to an HTML representation, then allowing the various HTML transcoders in WTP to render the desired resulting markup language, such as i-mode, HDML, WML, or even VoiceXML (see Figure 1). This greatly simplifies the

implementation by requiring only one stylesheet per document, or M stylesheets, or even fewer if generalized stylesheets can be used for multiple XML documents, perhaps incorporating parameterization to alter how a single stylesheet behaves based on preference information gathered during request processing (for example, to alter how a stylesheet behaves based on the requesting device.)

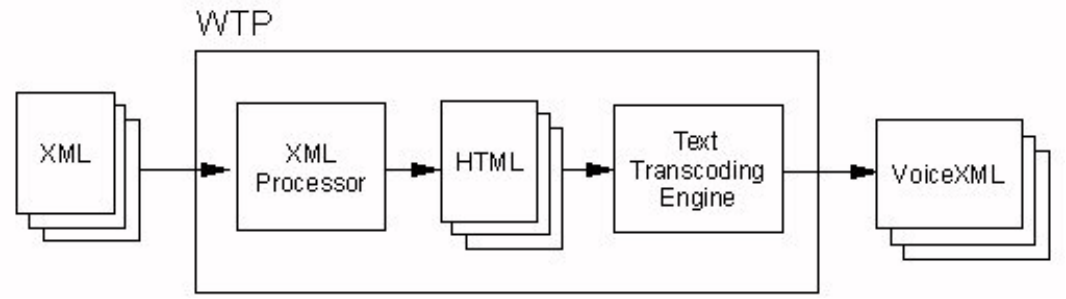


Figure 1: WTP applies XSL stylesheets to convert XML data to HTML which is then transcoded into device-specific formats, such as VoiceXML

Obviously, the guidelines we will present in this document on how to build HTML for VoiceXML transcoding apply in this scenario since you are first building a generic form of HTML from your XML. You need to consider carefully how the HTML will be formed if VoiceXML is one of your target formats.

Full HTML to VoiceXML transcoding

If you are a company who has already spent a great amount of resources building an e-commerce site or a customer self-service site, you can imagine that WTP's HTML-to-VoiceXML transcoder can be employed to voice-enable your entire site with little effort on your behalf. As we have already described, visual interfaces differ greatly from audio interfaces. If you intend to voice-enable a Web site, you will invariably need to either apply various WTP functions to simplify and mutate your content to make it easier to transcode into VoiceXML, rework the HTML and the Web site in general to make it easier to traverse using voice, or both.

WTP provides functions that can simplify an HTML page and make it more usable as a voice application, such as through the use of annotation, clipping, or custom response editors, but these only work on a page by page basis (see Figure 2). The HTML-to-VoiceXML transcoder and the rest of WTP can only help improve the page, not the entire site as a whole. For instance, your Web site's navigation methods may be too complex for a voice medium, or based largely on navigation bars and graphic hot spots which can't be associated well with the core content, or will get lost all together.

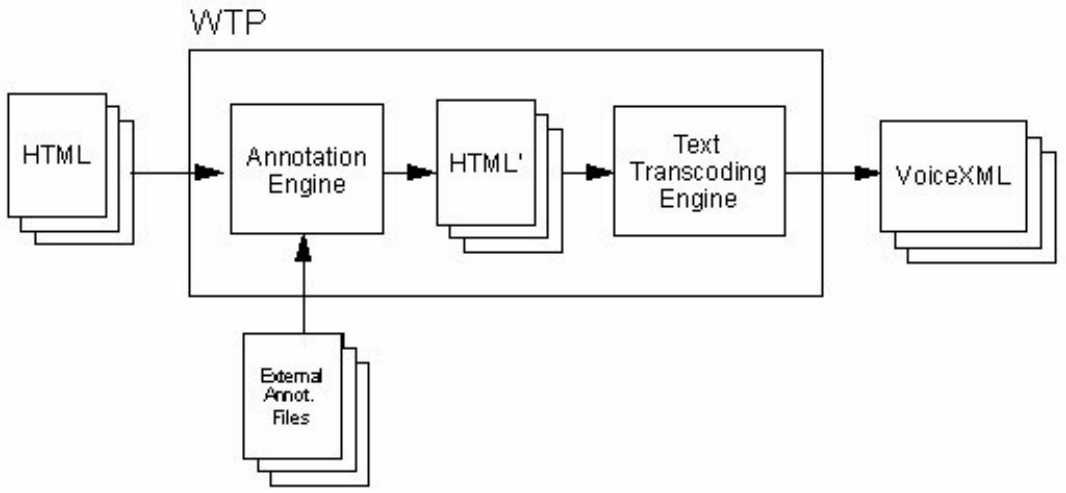


Figure 2: WTP uses annotation to remove unwanted data from an HTML page before it is transcoded into VoiceXML.

Mining content from HTML pages

Potentially the most useful application of WTP's HTML-to-VoiceXML transcoder is to convert HTML content mined from existing pages already hosting dynamic data (weather updates, stock quotes, employee addresses and phone numbers, catalog information) for use within a VoiceXML application.

A VoiceXML application is maintained separately from its visual counterpart and is tailored to the unique requirements of a listening audience. Still, it can be difficult to incorporate dynamic data within a static VoiceXML application, especially if that data already exists on a Web page somewhere. Certainly it would be easier for the VoiceXML application designer to reuse existing HTML data than to reproduce that data from some other source. WTP serves this purpose nicely, not only providing the HTML-to-VoiceXML

transcoder to render the HTML source in the appropriate markup, but also providing the tools necessary to carve out the required information from a page littered with superfluous data (see Figure 3). The HTML-to-VoiceXML transcoder can be used to derive either entire VoiceXML pages for use within the voice application, or portions of content which are to be placed within a VoiceXML page (as the figure shows).

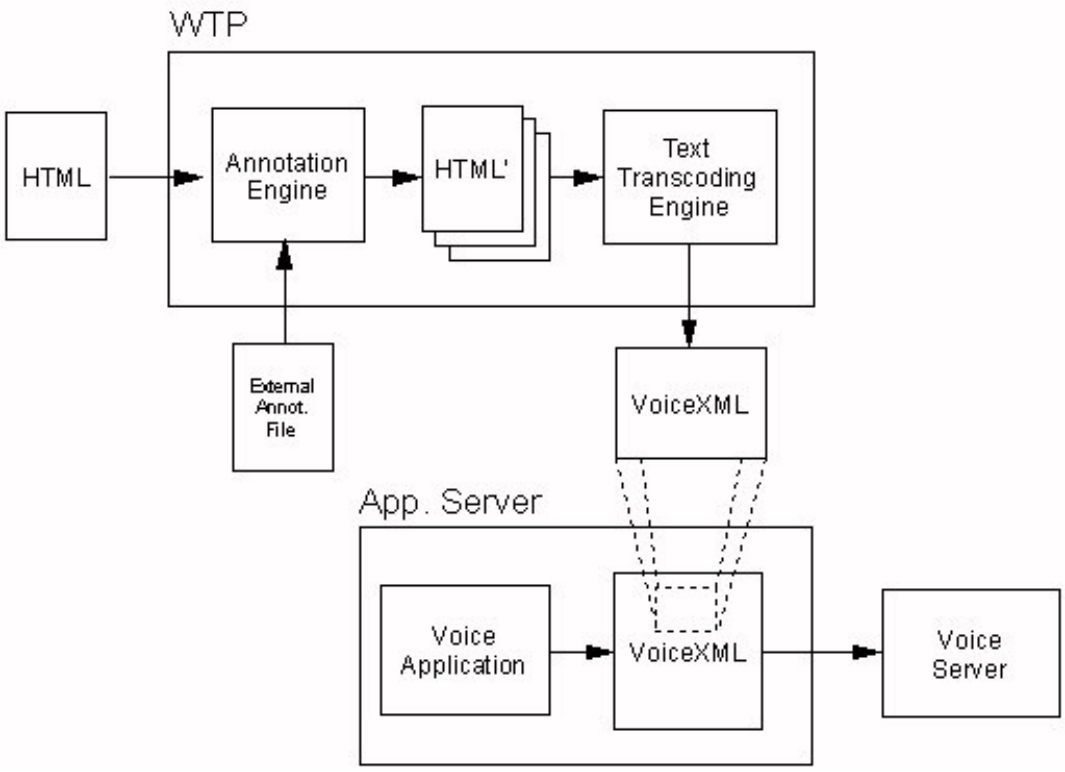


Figure 3: WTP applies annotation to a page to derive the necessary data before it is included within the VoiceXML application.

In General

The challenges of converting HTML or XML to VoiceXML can be remedied, assuming you have control over the original HTML source or over the conversion of the XML source to VoiceXML or HTML. What can't immediately be remedied is the fact that most voice synthesis technologies which will be used to read the transcoded HTML content will sound synthetic. Until voice synthesis becomes more natural, which it is promised to become, most companies will prefer voice applications written totally in VoiceXML, incorporating prerecorded human voice cues. Only dynamically generated content needs to be read synthetically, but it can still make use of prerecorded voice snippets.

Once you fully understand the differences between visual and audio interfaces, you might prefer to re-author the voice interface entirely in VoiceXML. Besides the complexity of deriving a usable vocal interface from a visual one through transcoding, maintenance could also be cumbersome as visual interfaces tend to change quite often, requiring changes to how they are transcoded as a result. In any case, extensive usability testing should be conducted on the solution to determine the best mix of transcoded content versus static or native VoiceXML content and how to best blend that mix.

HTML to VoiceXML transcoding capabilities

The HTML-to-VoiceXML transcoder does more than simply convert HTML tags to VoiceXML tags. It also analyzes the HTML document and splits it into three sections. One section contains the body of the document, divided into subsections based on the use of heading tags (<h1>, <h2>, etc.). Another section contains all of the links on the original page, removed from the source and offered here in a list. The last section contains a generated series of links which prompt the user to go to the main content of the page (first section), listen to a list of links on the page (second section), or exit the application.

Note that these main navigational links are inserted by the HTML-to-VoiceXML transcoder in the language of the requester. If the client requests a language WTP does not support, then English links are generated.

Isolating main content

The main content of the page is determined by the use of heading tags, which are assumed to be indicators of the primary subject matter for the page. If the document does not contain any heading tags, then the entire document is assumed to be the main content. All

imbedded URL links are removed and added to a list of links on the page which can be read separately (if that choice is made from the main prompt mentioned above). Imbedded images and other objects are also removed, leaving only the text to be spoken.

The **first three words** of the text between the heading tags are turned into links to the subject matter under that heading and these links are collected and listed at the top of this section to give the user a choice of what content to have read, as opposed to listening to the entire document. The end of each headed section contains a link back to the content list. The last link of the content list is a link back to the main prompt section. Refer to the appendix for examples of how an HTML document can be converted to VoiceXML.

If the data you are transcoding into VoiceXML isn't necessarily structured into paragraphs and topics, such as tabular data (stock quotes, weather forecasts, part numbers), then it is likely that this data will not contain any header tags and the main content will not be prefixed with any navigational links.

List of links found on the page

All HREF links found on the page are collected and added to a list which is voiced separately than the main content on the page, based on a selection made from the main menu. If the links were to remain embedded within the main content, then the voicing of the content would be interrupted with prompts to take that link to another page.

It is worth noting that the list of links on a page will contain both links embedded within text as well as those part of navigation areas. It may be difficult to understand the relevance of the link if taken out of context of the text around it. HTML design guidelines section later in this document will detail some techniques to ensure clarity in this area.

List grammars

The easiest input for a voice application to process is a selection list, where the choices are limited to a finite set of items. The user selects an item by simply speaking it. When processing an input form, the HTML-to-VoiceXML transcoder generates a grammar out of the options under the <SELECT> tag and inserts that grammar into the result.

Tables

Tables are a two-dimensional organization of data. Since voice applications are one dimensional, there is no concept of a table representation in voice. The HTML-to-VoiceXML transcoder converts tables to lists before being converted to VoiceXML. This function flattens a table vertically listing the table cells' contents row by row, from left to right. So, an original table which looks like this:

	Breakfast	**Lunch**	**Dinner**
Monday	Cereal	Sandwich	Pizza
Tuesday	Eggs and bacon	Soup	Meatloaf
Wednesday	Waffles	Salad	Chicken

Would be read like this:

Breakfast Lunch Dinner, Monday cereal sandwich pizza,
Tuesday eggs and bacon soup meatloaf,
...and so on.

Consider using annotation to simplify a table before it gets converted into the list mentioned above. Removing entire rows or columns will help limit the amount of information the end user has to process, focussing instead on the most important data.

Common command support

When using IBM WebSphere Voice Server, the built-in commands **Help**, **Quiet\Cancel**, and **Repeat** are always active and operate independent of the content being read. Two other common commands, Backup and Start Over, will execute depending on how they are used in the source. The points which you can back up to, or where you go when you start over, is specific to the document and cannot be automatically implemented.

Using WTP, you can insert grammars and links for these common commands using a Java clipper like the one in the Appendix of this document. Refer to the WTP Developer's Guide for more information on how to develop text clippers.

Barge-in

Barge-in describes the ability of the user to interrupt prompts being read and provide a command (such as Repeat or Exit) or make a selection from a menu. While the behavior of barge-in is defined within the Voice Server, you can enable and disable barge-in on a

per-prompt basis. You may want to disable barge-in if you are reading important information, such as legal notices or advertisements, and enable it for long sections of text.

Using the same Java clipper logic as described in the previous section, you can insert **<prompt bargein="true/false">** wherever applicable in the transcoded document.

HTML to VoiceXML transcoding limitations

The HTML-to-VoiceXML transcoder cannot predict the intention of all elements within an HTML page. There are potential issues that are specific to voice applications which may need to be resolved by the developer of this system, possibly through the use of facilities provided by WTP.

Grammars for text input in forms

The HTML-to-VoiceXML transcoder has no way of knowing what type of input is expected for a text box control and can't therefore generate a grammar for it. Text input using voice is not impossible, such as for numeric entries (as described earlier), so the HTML-to-VoiceXML transcoder preserves these text prompts in the VoiceXML, leaving it up to the developer to tag the prompts with the appropriate grammar, replace them with a selection list, or remove them all together. Since the Voice Server requires a grammar, you must specify one for each input control, using either annotation or Java clippers.

Multi-selection controls in forms

Some input controls in HTML allow multiple selections, such as <SELECT> tags with the MULTIPLE attribute set, or convey state, such as checkboxes. Since VoiceXML applications don't allow for more than one choice to be made from a list at a time or for an item to be checked, the MULTIPLE attribute is removed from select controls and checkboxes are omitted. In general, though, all controls of type **text**, **input**, **select**, and **option** are preserved.

Elements which are difficult to speak

It is inevitable that the voice browser will find words that it does not understand how to pronounce and will resort to spelling it. The most common occurrences include:

- **Acronyms -** If you are accustomed to hearing an acronym spoken as a word instead of spelled (for example, WYSIWYG stands for "what you see is what you get", but is normally pronounced "wizzi-wig") or spelled instead of spoken, then you might not recognize the acronym when the reverse is done by the voice browser. If you have access to the HTML source, then the number of such problems can be reduced by substituting the actual words for the acronym. You can also use annotation or Java clippers to insert <sayas> tags to instruct the Voice Server on how to pronounce certain words or acronyms.
- **Spelled URL links -** It is not uncommon for the text of an anchor tag to match its associated HREF URL link, so that the user of the page actually clicks on the address to be taken there. Besides being hard to understand the spoken address of a URL link, it isn't clear at all what is the significance of the link, especially since this link will be part of a list of links optionally read to you for selection. Annotation can be used to replace the anchor tag text with a meaningful description, or, if you have access to the source yourself, you can specify your own description.

Unsupported embedded types

Since the voice interface cannot support visual elements, all embedded graphical source is removed, such as images, video, and other forms of multimedia. There is also no means of specifying recorded audio files as embedded source within VoiceXML, so audio source is also removed, such as WAV and MID files.

HTML design guidelines

If you own, or have the ability to modify, the content of the HTML pages that will be transcoded into VoiceXML, then here are some HTML design guidelines which you can employ to improve the transcoding process.

Consistent page layout and structure

Whether transcoding your HTML pages into VoiceXML or not, it is good design practice to maintain a consistent structure for your HTML pages. It's easier to maintain such pages since you always know what areas of the page to update. The consistent layout provides consistent organization of content and look-and-feel across all of your pages. This will help create a consistent sound-and-feel in your transcoded pages as well.

Additionally, it is assumed that portions of your visual layout will have to be clipped to simplify the content and transcoding process. Any clipping you perform, whether it be through deploying custom Java clippers or using annotation, is based on a predictable document structure. If the structure changes, then your clipping instructions or logic will have to be updated. A consistent page structure helps guard against changes of content affecting the transcoding process.

Examples of consistent layout include placement of banners, frames, and navigation bars. These typically need to be removed or reworked by logic that you provide in WTP through clipping or annotation. If they always reside in the same location of the HTML document, then it makes it easier to isolate them from the rest of the content that you want to keep.

Consistent use of headers

As mentioned before, the HTML-to-VoiceXML transcoder identifies content topics based on the use of header tags (<h1>, <h2>, etc.). If your content is made up of paragraphs of text organized around topics, such as news headlines, magazine articles, and instruction manuals, then the consistent user of header tags will greatly improve the navigation of their VoiceXML equivalent. If a large amount of content exists under a heading, then all of this content will be read to the user if not reduced using annotation or clipping. Be sure to consider the appropriate use of the built-in Repeat, Backup, and Start Over commands.

The heading text should be clear, concise, and unique from other headings. When the HTML-to-VoiceXML transcoder converts these headings to links at the top of the main content document, it will be easier for the listener to keep up with his or her options. Remember, only the first three words of the heading are used in the link. If the words are not unique, then the user will not know the difference between the options. Also, since the attention span of a phone user is short, the number of headings, and thus topics, in a document should be limited to an average of seven, plus or minus two.

When the HTML-to-VoiceXML transcoder converts heading text into a link to that heading, it does so assuming the text is all part of the same text "node", or continuous block of text. If any text within the heading changes state, such as through the use of the tag to adjust font size, then internally the text will be split between different text nodes. After removing the formatting tags, the HTML-to-VoiceXML transcoder will not know how to reconstitute the original phrase and you will end up with an incomplete heading. Take the following heading for example:

```
<h1><span style="font: 14pt">O</span>verview</h1>
```

This would be converted to a link using the text "verview" instead of "Overview". Headings need to use a single style.

URL links

As mentioned before, it is common to find HTML documents where the address of a URL link is actually spelled out as the text of the anchor tag. For example:

```
<A HREF="http://www.ibm.com">http://www.ibm.com</A>
```

It is also common to find instructions to the user for where to click to go somewhere in particular. The most typical example is the use of the word *here* to denote where to click. For example:

```
Click <A HREF="http://www.ibm.com">here</A> to go to the IBM home page.
```

In either of these examples, the HTML-to-VoiceXML transcoder will convert the anchor tag text into the link, which will be read to the listener. It is cumbersome to listen to the voice browser spell the URL address and the listener will not know what *here* refers to. The solution is to provide a clear, concise, and unique description of the link:

```
<A HREF="http://www.ibm.com">IBM's home page</A> provides more information...
```

Consistent form layout

The HTML-to-VoiceXML transcoder converts form input controls into a series of fields. Each field has a prompt and possibly a series of options (for lists and radio choices). HTML forms do not enforce the use of a label, or prompt, to describe each input control. The HTML-to-VoiceXML transcoder makes certain layout assumptions on what text to use for input prompts and which to use for describing the options available. If the source HTML is not consistent with this layout, the resulting field prompts and options in VoiceXML may not be properly aligned with the original source.

For example, here is a simple HTML form that adheres to the transcoding layout assumptions:

```
<form action=".....">
   <p>The preamble or question for the form should be here.
   <input type="radio" value="somevalue1" > Input description for value1.
   <input type="radio" value="somevalue2" > Input description for value2.
```

```
   .
   .
   .
</form>
```

After transcoding to VoiceXML, the resulting field would look like this:

```
<field name="...">
   <prompt>The preamble or question for the form should be here.</prompt>
   <option value="somevalue1">Input description for value1</option>
   <option value="somevalue2">Input description for value2</option>
   .
   .
   .
</field>
```

Now if the preamble were missing, then the first input description would be used as the prompt, and the option description for value2 would be missing, like so:

```
<field name="...">
   <prompt>Input description for value1</prompt>
   <option value="somevalue1">Input description for value2</option>
   <option value="somevalue2"></option>
   .
   .
   .
</field>
```

Likewise, if the preamble is split due to extra text formatting, then the prompt will include only the text in the first text "node" of the source, before it is reformatted. The remainder of the text will be used as option descriptions. So, if the original HTML form source looks like this:

```
<form action=".....">
   <p>The preamble or question for the form should be here.<p>Don't put any extra formatting tags in the
preamble.
   <input type="radio" value="somevalue1" > Input description for value1.
   <input type="radio" value="somevalue2" > Input description for value2.
   .
   .
   .
</form>
```

The resulting VoiceXML would look like this:

```
<field name="...">
   <prompt>The preamble or question for the form should be here.</prompt>
   <option value="somevalue1">Don't put any extra formatting tags in the preamble.</option>
   <option value="somevalue2">Input description for value1</option>
   .
   .
   .
</field>
```

Note that the description for value2 has been lost and none of the descriptions line up with the input controls.

Data simplification

HTML pages offer a large amount of real estate for displaying data. You can probably enhance a voice experience with this data if you can somehow categorize some of the data under a link for "more details". This way, less information is initially presented to the listener, and the listener can get more information by following a link. If the page in Figure 5 below were to be transcoded, you would probably want to only read the movie titles and show times and move the details, such as rating, length, and additional details to another page which can be linked to from the main list. Remember, the link should be qualified somehow to ensure its uniqueness from other links, such as through including the movie title in the link.

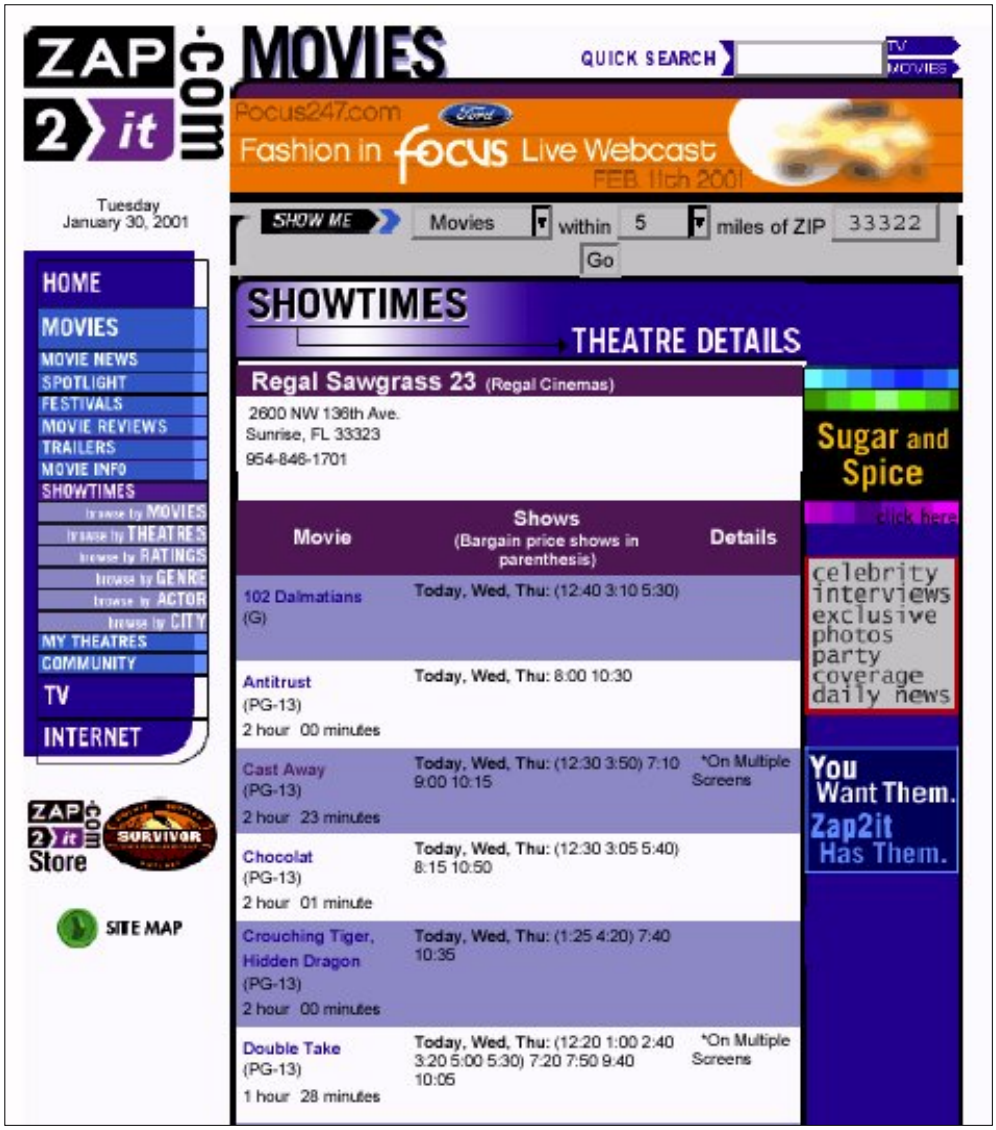


Figure 5: Example of an HTML page where the information presentation could be simplified with links to more information.

HTML pages typically make wide use of tables for page layout and data presentation. Focus on the most important information, eliminating rows and columns when possible.

Using annotation to simplify the page

Annotation refers to an XML-based language which provides a series of instructions for modifying an HTML document. The instructions are typically for clipping regions of the page, but could also involve replacing parts of the document with new content, or inserting new content.

Again, it is assumed that the typical HTML page will not transcode to VoiceXML well enough to produce a usable result. Annotation is the easiest way to simplify the HTML page before it is transcoded without affecting the original HTML content.

Difference between internal and external annotation

The annotation instructions can reside within the HTML source itself (internal annotation) or in a separate file (external annotation) which gets applied to the HTML file at runtime.

Internal annotation allows the instructions to be maintained along with the rest of the document. The instructions are imbedded as HTML comments which are interpreted only by WTP's annotation engine. If the source or structure of the document were to change, the internal annotations could be synchronized with the changes. Internal annotation does presume you have control over the HTML content and can add the annotations. If you do not have such control, then you will have to use external annotation.

Note: IBM WebSphere Studio's HTML editor allows internal annotations to be built visually based on clipping regions that you mark in the graphic view of the page.

Annotation instructions in an external annotation file are the same as internal annotations; they simply reside in a separate file. Using WTP administration, the external annotation file is associated with the HTML document's URL against which its instructions are to be applied. The disadvantage is that this file must be maintained separately from the HTML source. Extra effort may be required to keep abreast of changes to the HTML source in order to keep the annotations current. The benefits are that you don't have to own the HTML source to support annotation and you can write annotations which can be applied to multiple HTML files.

Content clipping

The most common use of annotation is for clipping regions of the original HTML document, leaving only that which you want to convert to VoiceXML. The annotation instructions identify the tags, or nodes, which mark the start and end of the clipping regions. (These tag locations are specified in terms of their XPATH. Before the annotations are processed, the HTML document is parsed into a document object model (DOM) where it is represented as a tree where each node in the tree is a tag in the original HTML. The XPATH of a node is the unique path to that node in the DOM tree.) A common use of annotation is to clip table regions, specifying rows and/or columns to remove, which is an easy method of simplifying data presentation, or even page layout.

Here is an example of some simple annotation statements for clipping regions of a document:

Table annotation

These instructions keep all but the first column and second row of the first table in the document.

```
<description take-effect="before" target="/descendant::TABLE[1]">
    <keep />
    <table majoraxis="row">
        <column index="1" clipping="remove" />
        <column index="*" clipping="keep" />
        <row index="*" clipping="keep" />
        <row index="2" clipping="remove" />
    </table>
</description>
```

Text clipping

These instructions remove all content after the first table in the document. Content will continue to be clipped until a <description> with a <keep /> instruction is encountered.

```
<description take-effect="after" target="/descendant::TABLE[1]">
    <remove />
</description>

<description take-effect="before" target="/descendant::TABLE[2]">
    <keep />
</description>
```

Form simplification

Most forms on Web sites contain several optional fields which could actually be clipped out of the content to reduce the amount of input required from the listener. For other input controls, you might be able to identify what typically is provided as input the majority of the time. You could default the value of that input and eliminate the control all together. The less information you require of the listener to complete the task, the better their experience with the interface will be. You could also replace a text input control with a select and limit the possible values to a list. See the next section for details.

Content replacement

Using annotation to clip out unnecessary regions of the document before transcoding to VoiceXML may not be enough to produce usable results. It could be the case that certain elements of the original content need to be replaced all together with new content designed specifically for the listener.

A good example of this is converting a text input control to a select list to limit the choices the user has as input and eliminate the problems of having to recognized spelled input. The following annotation example replaces the CITY text input control with a SELECT control with three options:

```
<field name="CITY" type="SELECT">
    <option value="Raleigh" label="Raleigh" />
    <option value="Durham" label="Durham" />
```

```
    <option value="Chapel Hill" label="Chapel Hill" />
</field>
```

In the original HTML input form, the user could type in any city. After applying this annotation, the choices for the city are limited to picking from a list of three: Raleigh, Durham, and Chapel Hill.

Alternatively, you can use annotation to replace an entries form with a new one:

```
<description take-effect="before" target="/descendant::FORM[1]">
   <replace>
      <form>
         <text>NEW prompt for INPUTTEXT3(now a select)</text>
         <field name="INPUTTEXT3" type="SELECT">
            <option value="No" label="No." />
            <option value="Yes" label="Yes." />
            <option value="Maybe" label="Don't Care" />
         </field>
         <field type="SUBMIT" />
      </form>
   </replace>
</description>
```

You can also replace text between tags in the document with alternative text. The following example changes the title text of the document:

```
<description take-effect="after" target="/descendant::TITLE[1]">
   <replace>
      <text>IBM Stock Quote</text>
   </replace>
</description>
```

Refer to the appendix for an example of how annotation can be used to simplify an HTML page before it is converted to VoiceXML.

Transcoding Publisher deployment considerations

There are several ways to deploy WTP into production: as a network proxy, as a servlet filter within a WebSphere Application Server Web application, and as a set of JavaBeans. The consideration that goes into deciding which way to deploy WTP needs to include not only performance and capability, but also the impact to a voice application.

Let's explore each of the three options in some more detail. For a complete description of these deployment models, refer to the WebSphere Transcoding Publisher Administrator's Guide.

Network Proxy

When deployed as a network proxy, WTP sits between the client issuing a request and the server returning a response. All requests and responses flow through the proxy, exposing WTP to all types of clients making requests and allowing the content of the responses to be tailored to the client devices' capabilities. Clients simply have to specify WTP as their network proxy. The back-end Web servers don't change. This type of network proxy is often termed a "forward proxy," where requests are forwarded to servers on behalf of the client.

WTP can also be deployed as a "reverse proxy" where the proxy appears as the server to the client. Reverse proxies forward responses back to the client on behalf of the actual server. Reverse proxies are useful for devices which don't support specifying a network proxy or for proxy-capable devices when you don't want to require the device's proxy setting to change. Since WebSphere Voice Server supports the configuration of a proxy, and the WTP reverse proxy has the same implications to the voice application as the forward proxy, it is assumed that the network proxy would be the preferred proxy implementation for use with the Voice Server.

A variation on this model allows for an external cache to be defined, such as WebSphere Edge Server's caching proxy. WTP makes use of Edge Server to store transcoded results such that if another request is made for a page which has already been transcoded and stored in the cache, the cached version is returned as the response instead of requesting and transcoding a new instance of the target page.

Deploying WTP as a network proxy is quick and easy, not to mention necessary if you do not own the Web servers serving the content. The only major restriction with the proxy model is that there is no visibility to encrypted data. Therefore, if a secure connection is established between the Voice Server and the document server through WTP, then WTP cannot transcode that data as a proxy.

WTP configuration

After installation of WTP is complete, you use the Server Setup wizard to specify how WTP will be run. After selecting Network Proxy,

with or without an external cache, simply start the WTP server and the proxy is now ready to accept requests.

The runtime preferences and resource registrations can be altered using either the Administration Console or the Import/Export facility.

Voice Server configuration

WebSphere Voice Server is the client issuing the request through WTP, so it must configure WTP as its proxy using the HTTP_PROXY parameter in the sysmgmt.properties file.

Performance considerations

As mentioned before, WTP sees every request from every client which has configured WTP as its proxy. WTP interjects at least some amount of processing overhead on every request and response to determine if any action must be taken, based on rules derived by the runtime preferences you have defined for WTP. Even if the rules determine that no action needs to be taken, that processing distracts from the over all response time for the request.

The performance impact will be on the order of a second or less, but consider using the VoiceXML <**fetchaudio**> tag to fill dead air time with prerecorded audio so listeners will know that the call is still active.

Voice application implications

If the voice application is derived from applying XSL stylesheets to XML data, then WTP can apply those stylesheets by sitting between the Voice Server and the XML data source. This assumes that the XML data is requestable from a Web server.

If the voice application is derived from transcoding an entire Web site, page by page, into VoiceXML, then WTP as a network proxy can be placed between the Voice Server and the Web server, transcoding the HTML as it is requested by the Voice Server. This means the Voice Server requests HTML pages, not VoiceXML pages.

If the voice application is already based in VoiceXML, and WTP is to be used to convert Web content into VoiceXML for use within the existing application, then the transcoding must occur before the resulting VoiceXML from the voice application is rendered, not after. Therefore, WTP as a network proxy stands between the application server hosting the voice application and the Web server hosting the dynamic content, as opposed to between the Voice Server and the application server. The voice application must request the additional content from the external Web source and specify that WTP be its proxy to that source. For example, if the voice application is servlet based, then that servlet may request additional content through WTP as a proxy by issuing a GET request to the WTP proxy for a URI specifying another Web server. That way, WTP will transcode the content on the way back to the voice application.

Servlet Filter

WTP is able to be configured as a MIME filter servlet within a Web application defined in WebSphere Application Server. MIME filter servlets have the opportunity to process the response data generated by one or more servlets in a Web application before it leaves the server. Because of this, WTP is able to transcode data before it is encrypted by the hosting Web server, something WTP as a network proxy is unable to do.

The MIME types the filter servlet processes are equivalent to the supported input types: text/html, text/xml, image/gif, and image/jpeg.

Deploying WTP as a servlet filter presumes that you own the data source since application servers are responsible for generating that source. A good example would be where the Web application that defines WTP as its servlet filter employs JavaServer pages (JSPs) to generate content dynamically. JSPs are very much like HTML pages that contain embedded Java statements for producing content for the page. WTP as a servlet filter could be used to transcode the HTML resulting from JSP processing into a presentation suitable for the device originating the request.

WTP configuration

WTP must be installed on the same system as WebSphere Application Server. When configuring WTP as a MIME filter servlet using the Server Setup wizard, you will be presented with a list of Web applications from which you must select the ones to which you want the filter servlet added. Once setup is complete, WebSphere Application Server is reinitialized to pick up the changes.

Voice Server configuration

In this case, WTP is running as part of the data source instead of the proxy, so Voice Server doesn't configure anything in particular as its proxy. Instead, it makes its content requests directly to the WebSphere Application Server.

Performance considerations

Since WTP is not running as a proxy, you don't have to worry about WTP processing every request and response passing through it. Also, since WTP is running as a servlet within a Web application, it only transcodes content generated by that Web application, so WTP can be specialized for specific data sources.

Voice application implications

If the voice application is derived from applying XSL stylesheets to XML data, then WTP can apply those stylesheets as a filter servlet to the XML data being produced by other servlets in the same Web application.

If the voice application is derived from transcoding an entire Web site, page by page, into VoiceXML, then WTP as a servlet filter can convert HTML being generated by other servlets in the same Web application to VoiceXML for Voice Server requests while leaving the HTML alone for normal browser requests.

If the voice application is already based in VoiceXML and is hosted by WebSphere Application Server, then adding WTP as a servlet filter to the same Web application is not going to accomplish anything. The data is already in VoiceXML. However, if this voice application requires data which is in HTML or XML, then the voice application servlets could make an HTTP request to another web application for that data and have it transcoded by WTP. WTP could also be used as a proxy between the voice application and the external data source, as described in the previous section. The difference is that a proxy doesn't have to be specified on the request and the data can now be encrypted between the voice application server and the external data source application server.

Conclusion

WebSphere Transcoding Publisher offers several different ways to convert traditional Web-based data into VoiceXML. The developer of the voice application must keep in mind, though, that going from a two-dimensional presentation, or a raw data source, to a one-dimensional presentation carries with it many challenges. The end-user's experience with this voice application should be the focus, and how to best incorporate Web data into a voice application and make it usable should be the challenge.

Use the guidelines offered in this paper to decide the best approach to solving the problem, whether it be re-authoring the content to make for a better voice experience, or applying transcoding tools such as annotation to first simplify the data sufficiently before converting it to VoiceXML, or go straight to the data source and build sets of XSL stylesheets to provide a voice presentation to raw data. It is not likely that you will use WebSphere Transcoding Publisher to transcode your entire Web site to VoiceXML, but you can use its flexibility to make Web data in general accessible to someone listening on the other end of the phone.

Appendix: Examples of VoiceXML transcoded from HTML

Straight HTML to VoiceXML transcoding

The following is a simple HTML page which uses header tags and hyperlinks:

```
<HTML>
<HEAD>
<TITLE>Voice Interface Considerations</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" VLINK="#800000">
<H2>Voice interface considerations</H2>
<P>A voice interface requires special considerations for ensuring usability. The first thing to understand is that a voice interface implies a completely different set of rules and expectations for the end user than traditional visual interfaces. You can get more information on authoring a voice interface using VoiceXML from the <A HREF="http://www.voicexml.org/specs_1.html" TARGET="_top">VoiceXML 1.0 specification</A>.
<H3>Mental load</H3>
<P>Voice application designers need to limit the amount of information presented to the user, avoiding overpowering short term memory. Remember, there's no document to view and no back button. Thus when transcoding HTML to VoiceXML using <A HREF="http://www.ibm.com/software/websphere/transcoding/" TARGET="_top">WTP</A>, extra care needs to be taken in reducing the complexity of the data.
<H3>Navigation and error recovery</H3>
<P>For voice applications, it is not obvious to the user where he is in the application. Where navigation is concerned, you need to provide a consistent approach for traversing menus and pages of data. You need to be restrained in how much data is presented to the user and ensure that all of the data is closely related in subject.
<H3>Wording and input processing</H3>
```

<P>When a voice application requires text input, that input must be limited to a well-understood and bounded grammar, or specification of valid input. When the grammar can be limited to a (short) list of words from which to choose, such as through a navigation menu or selection list, then the voice browser needs only to "concentrate" on understanding those set inputs. That grammar must either be generated as part of the page, or dynamically generated as part of the application logic, such as from a database of grammars.
</BODY>
</HTML>

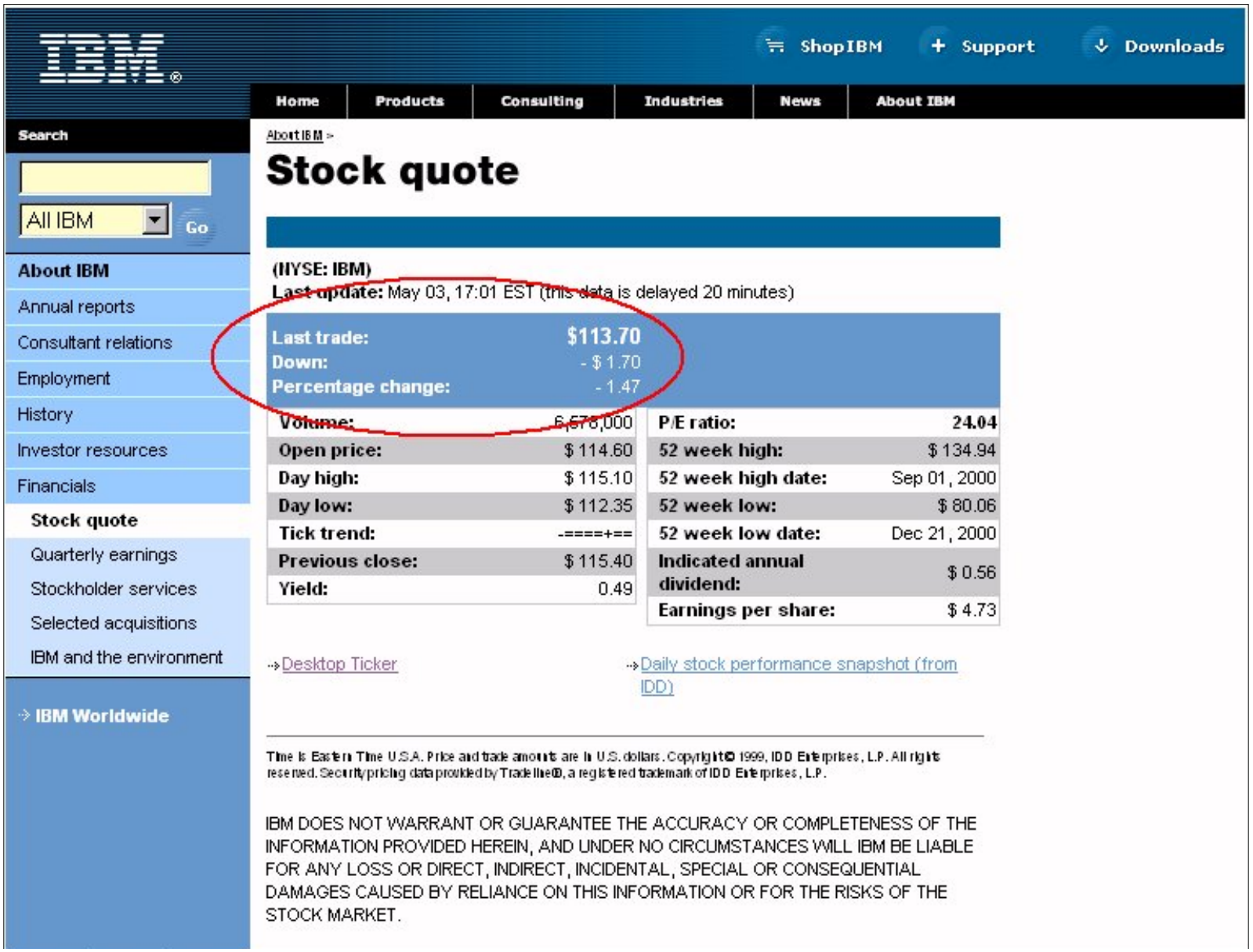
Here is the resulting VoiceXML after it passes through the HTML-to-VoiceXML transcoder. Note the major sections in bold:

<?xml version="1.0" encoding="UTF-8"?>
<vxml version="1.0"><form id="start"><field name="answer_one"><prompt>Please choose one of the following sections to go to.
<break size="large"></break>
<enumerate/></prompt>
<option value="**main**">Main Content </option>
<option value="**links**">Links </option>
<option value="exit">Exit </option>
<filled><if cond="answer_one == 'main'"><goto nextitem="main"></goto>
<elseif cond="answer_one == 'links'"></elseif>
<goto next="#links"></goto>
<elseif cond="answer_one == 'exit'"></elseif>
<exit/></if>
</filled>
</field>
<field name="**main**"><prompt>To exit the browser, say exit. Otherwise, please choose one of the following topics. <break size="large"></break>
<enumerate/></prompt>
<option value="**Voice_interface_considerations**">Voice interface considerations </option>
<option value="**Mental_load**">Mental load </option>
<option value="**Navigation_and_error**">Navigation and error </option>
<option value="**Wording_and_input**">Wording and input </option>
<option value="exit">Exit </option>
<filled><if cond="main == 'exit'"><exit/><elseif cond="main == 'Voice_interface_considerations'"></elseif>
<goto nextitem="Voice_interface_considerations"></goto>
<elseif cond="main == 'Mental_load'"></elseif>
<goto nextitem="Mental_load"></goto>
<elseif cond="main == 'Navigation_and_error'"></elseif>
<goto nextitem="Navigation_and_error"></goto>
<elseif cond="main == 'Wording_and_input'"></elseif>
<goto nextitem="Wording_and_input"></goto>
</if>
</filled>
</field>
<block><goto next="#start"></goto>
</block>
<block name="**Voice_interface_considerations**">Voice interface considerations <goto nextitem="block4"></goto>
</block>
<block cond="true" name="block4">A voice interface requires special considerations for ensuring usability. The first thing to understand is that a voice interface implies a completely different set of rules and expectations for the end user than traditional visual interfaces. You can get more information on authoring a voice interface using VoiceXML from the VoiceXML 1.0 specification . <goto next="#start"></goto>
</block>
<block name="**Mental_load**">Mental load <goto nextitem="block6"></goto>
</block>
<block cond="true" name="block6">Voice application designers need to limit the amount of information presented to the user, avoiding overpowering short term memory. Remember, there is no document to view and no back button. Thus when transcoding HTML to VoiceXML using WTP , extra care needs to be taken in reducing the complexity of the data. <goto next="#start"></goto>
</block>
<block name="**Navigation_and_error**">Navigation and error recovery <goto nextitem="block8"></goto>
</block>
<block cond="true" name="block8">For voice applications, it is not obvious to the user where he is in the application. Where

navigation is concerned, you need to provide a consistent approach for traversing menus and pages of data. You need to be restrained in how much data is presented to the user and ensure that all of the data is closely related in subject. <goto next="#start"></goto>
</block>
<block name="**Wording_and_input**">Wording and input processing <goto nextitem="block10"></goto>
</block>
<block cond="true" name="block10">When a voice application requires text input, that input must be limited to a well-understood and bounded grammar, or specification of valid input. When the grammar can be limited to a (short) list of words from which to choose, such as through a navigation menu or selection list, then the voice browser needs only to quote concentrate quote on understanding those set inputs. That grammar must either be generated as part of the page, or dynamically generated as part of the application logic, such as from a database of grammars. <goto next="#start"></goto>
</block>
</form>
<menu id="**links**"><prompt>The following is a list of links on this page. To follow a link, please say the link name into the microphone. <enumerate/></prompt>
<choice next="http://www.voicexml.org/specs_1.html"> VoiceXML 1.0 specification </choice>
<choice next="http://www.ibm.com/software/websphere/transcoding/"> WTP </choice>
</menu>
</vxml>

Using annotation along with transcoding HTML to VoiceXML

This is a snapshot of the IBM stock page at www.ibm.com/ibm/stock.



Obviously, this is too much information for presenting in a voice interface. We need to focus on the pertinent data and clip out the unnecessary information, such as the navigation bar along the top, bottom, and left side of the page, as well as much of the statistical data. For a voice interface, all we want to present is the last trade, the amount it changed from the last quote, and what percentage change that is. In other words, clip away everything but the data in the blue box in the middle of the page, circled in red.

To do this, we apply the following annotation:

```
<?xml version='1.0' ?>
<annot version="1.0">
<description take-effect="before" target="/HTML[1]/BODY[1]/*[1]">
   <remove />
</description>
<description take-effect="before" target="/descendant::IMG[1]">
   <keep />
   <replace>
     <text>IBM Stock Quote</text>
   </replace>
</description>
<description take-effect="after" target="/descendant::IMG[1]">
   <remove />
</description>
<description take-effect="before" target="/descendant::TABLE[8]">
   <keep />
   <table>
      <column index="*" clipping="remove" />
      <column index="2" clipping="keep" />
      <column index="3" clipping="keep" />
      <row index="*" clipping="remove" />
      <row index="6" clipping="keep" />
      <row index="7" clipping="keep" />
      <row index="8" clipping="keep" />
   </table>
</description>
<description take-effect="after" target="/descendant::TABLE[8]">
   <remove />
</description>
</annot>
```

The result of applying this annotation to the stock page looks like this:

The resulting VoiceXML after passing this simplified HTML through the HTML-to-VoiceXML transcoder looks like this:

<?xml version="1.0"?>
<vxml version="1.0"><form><field name="answer_one"><prompt>Please choose one of the following sections to go to. <break size="large"></break>
<enumerate/></prompt>
<option value="**main**">Main Content </option>
<option value="exit">Exit </option>
<filled><if cond="answer_one == 'main'"><goto nextitem="main"></goto>
<elseif cond="answer_one == 'exit'"></elseif>
<exit/></if>
</filled>
</field>
<block name="**main**">IBM Stock Quote Last trade: $113. 70 Down: - $ 1. 70 Percentage change: - 1. 47 <goto nextitem="answer_one"></goto>
</block>
</form>
</vxml>

Trademarks

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.