

AN EXTENSIBLE TRANSCODER FOR HTML TO VOICEXML CONVERSION

APPROVED BY SUPERVISORY COMMITTEE:

Supervisor: _____

AN EXTENSIBLE TRANSCODER FOR HTML TO VOICEXML CONVERSION

by

Narayanan Annamalai, B.E. in CSE

THESIS

Presented to the Faculty of
The University of Texas at Dallas
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Copyright © 2002
Narayanan Annamalai
All Rights Reserved

Dedicated to my Parents

ACKNOWLEDGEMENTS

My sincere gratitude goes to my advisors Dr. Gopal Gupta and Dr. B Prabhakaran for being instrumental in shaping my ideas and helping me achieve my ambitions. But for their guidance, it would not have been possible for me to complete my thesis.

I would also like to thank Dr. Latifur Khan for serving in my thesis committee. I would like to extend my sincere gratitude to all my family, friends who have been supportive and encouraging all through my career. Finally, I would like to thank my fellow graduate students Srikanth Kuppa and Vinod Vokkarane for reading my thesis draft and providing me valuable suggestions and helping me with the format of the draft.

AN EXTENSIBLE TRANSCODER FOR HTML TO VOICEXML CONVERSION

Publication No. _____

Narayanan Annamalai, M.S
The University of Texas at Dallas, 2002

Supervising Professors: Dr. Gopal Gupta and Dr. B Prabhakaran

‘Anytime anywhere Internet access’ has become the goal for current technology vendors. Sudden increase in the number of mobile users has necessitated the need for ‘Internet access through mobile phones’. The existing web infrastructure was designed for traditional desktop browsers and not for hand-held devices. The data in the web is stored in HTML (Hyper Text Markup Language) format which cannot be delivered to mobile devices. The only acceptable way to present data to devices like cellular phones, is audio. Certain voice browsers are capable enough to process VoiceXML content and produce the output in the form of audio. Thus it is imperative that a transcoder which converts HTML data to VoiceXML data is developed. In this thesis, we propose and implement a system which converts HTML (4.0 or lower version) file to corresponding VoiceXML file. The transcoder is divided into two phases: The parsing phase where the input HTML file is converted to HTML node tree, and the translating phase where each node in the HTML tree is mapped to its equivalent VoiceXML node. Our transcoder is extensible in the sense it can be upgraded easily by users to accommodate new HTML tags in the future.

TABLE OF CONTENTS

Acknowledgements	v
Abstract	vi
List of Figures	ix
List of Tables	x
Chapter 1. INTRODUCTION	1
1.1 Voice Enabled Applications	1
1.2 Related Work	2
1.3 The Thesis	4
1.4 Research Objectives	4
1.5 Contributions	5
Chapter 2. INTRODUCTION TO VOICEXML	6
2.1 What is VoiceXML?	6
2.2 VoiceXML Features	6
2.3 Architecture	7
2.4 Framework	7
2.4.1 Dialogs	8
2.5 Grammar	8
2.5.1 Scope of Grammars	9
2.6 Elements of VXML files	9
2.6.1 Event Handling	11
2.6.2 Links	12
2.6.3 Forms	12
2.6.4 Field Items	12
2.6.5 Control Items	13
2.6.6 Form Interpretation	13
2.6.7 Menus	13
2.6.8 Prompts	14
2.7 HTML 4.0	14
2.7.1 Table	14
2.7.2 Forms	15
2.8 Comparison of HTML versus VoiceXML	15

Chapter 3. SYSTEM MODEL AND ASSUMPTIONS	17
3.1 Architecture	17
3.2 Assumptions.....	19
Chapter 4. TRANSCODER LOGIC	21
4.1 Parsing Phase	21
4.2 Translation Phase	23
4.3 Structure	24
4.4 Framework of the Overall Process	25
4.5 Translation Issues	25
4.6 Translator Logic	28
4.6.1 Form Tags	28
4.6.2 Link Tags	32
4.6.3 Image Tags	32
4.6.4 List Tags.....	33
4.6.5 Frame Tags	34
4.6.6 Spacing/Layout Tags	35
4.6.7 Rule Tags	35
4.6.8 Tables	35
4.6.9 Text Tags	37
4.7 Pseudo-code of the Algorithm	37
4.8 Implementation Details.....	38
Chapter 5. CONCLUSION AND FUTURE WORK	40
5.1 Applications	41
5.2 Future Work	41
Bibliography	44
Vita	47

LIST OF FIGURES

2.1	System Architecture	7
2.2	Scope of VXML Grammars.....	10
3.1	System Model	18
3.2	Structure of Interface Sheet	19
4.1	Example Html Node Tree.....	22
4.2	HTML Parser - Example conversion	23
4.3	Overall Process	26
4.4	Form node - Example of radio type.....	30
4.5	Example conversion of radio Tags.....	30
4.6	Example Conversion - Input Tag <i>text</i>	31
4.7	HTML Frame	34
4.8	Example Conversion - HTML table	36
4.9	High Level Implementation Diagram	39
5.1	Future Work - Proxy Server.....	42

LIST OF TABLES

2.1	Overview of HTML Tags	15
4.1	Elements of Interface Sheet	24
4.2	Table Tags	35

CHAPTER 1

INTRODUCTION

The World Wide Web (WWW) has seen tremendous growth in the recent years and has become the primary source of information all over the world. The WWW accounts for the major proportion of entire Internet traffic. The other area of remarkable growth is the wireless access medium. Surveys report that mobile phone users in the United States alone are increasing by 12 million every year. Mobile phones are designed to provide anytime anywhere access to users. The challenge that is presented to the present Internet world, is to make the enormous web content accessible to mobile phone users as well as visually impaired users.

The major drawback of the existing web infrastructure is that, the present web content was originally designed for traditional desktop browsers. The data is basically stored in the HTML (Hypertext Markup language) format and so they are not suited for devices that has less processing power, limited screen size, constrained memory, and limited input capabilities. For example, consider accessing the CNN's weather page through mobile phone, it is very difficult to display world maps and weather forecasts in the mobile phone's small screen. It is even difficult and time consuming to enter text through mobile phone's key pad.

1.1 Voice Enabled Applications

Voice has always been an accepted medium of user interaction which greatly simplifies the input process. The development of interactive voice browsers which uses an improved means of voice recognition and efficient Text-to-speech (TTS) engines has made it possible for the mobile users to access the Internet. VoiceXML (VXML) is a standard markup

language founded by AT&T, IBM, Lucent and Motorola to make Internet content and information accessible via voice and phone. Just as a web browser renders HTML documents visually, a VoiceXML interpreter renders VoiceXML documents aurally. Since the documents are rendered aurally they can be heard over the phone. Since most of the content in the web consists of ‘HTML pages’, they cannot be easily accessed through voice interfaces.

One solution to this problem, is to maintain different versions of the same web site that provides content, formatted for specific devices. From the web server’s perspective, maintaining multiple versions of the same web site is labor intensive and expensive(in terms of memory required). However, this approach still would not solve the problem completely as it is impractical to maintain versions of web site in all possible formats.

The only solution to the problem seems to be, developing an application that is capable of converting HTML content in the web to VoiceXML content, since VoiceXML is used for specifying the dialogs of interactive voice response applications. Voice browsers featuring speech synthesis and voice recognition technologies are capable of processing VoiceXML content and provide them in the form of audio which can be heard over phone.

In this thesis, we propose an extensible Transcoder that can convert HTML data to corresponding VoiceXML data on the fly. The novelty of our approach is that the functioning of the transcoder, (or) the outcome of the application can be customized according to the user’s instruction/idea. The user is enabled to instruct the system via a user interface sheet.

1.2 Related Work

Until recently, to access web related information the visually impaired people had to rely upon Braille output devices and software applications such as *Screen readers*. The *Screen reader* [3] would capture the text present in the web page and employ speech synthesis technologies to speak this information to the user. This application captures only the plain text from HTML pages ignoring the structural information provided by the tags. Thus the produced speech output will not help in understanding the structure of the web page. Moreover such systems cannot provide the interactive features present in the current HTML

pages.

Frankie James [15] proposed a framework for developing a set of guidelines for designing audio interfaces to HTML called as the *Auditory HTML access system* or the 'AHA'. The system is basically an improved version of the Screen reader. The system understands the structure of the HTML page by analyzing the tags. The authors developed an interface for marking structures with a distinct sound or tone. For example, a link in the web page is spoken out with a preceding distinct sound. The visually impaired can make-out the structure of the document with the help of tones. The clear disadvantage of the system is that it fails to address the interactive features provided by the HTML pages.

Stuart Goose et al. [8] proposed an approach for converting HTML to VoXML. Very similar to VoiceXML, VoXML also requires an audio browser that is capable of processing VoXML tags and reading out the information contained in VoXML marked-up web page. Since VoXML is a primitive language, current application vendors do not develop audio browsers that support VoXML tags. Hence, this approach is seldom used by business clients.

Gopal Gupta et al. [6] have proposed an approach for translating HTML to VoiceXML. Their approach is based on denotational semantics and logic programming. Even though, the methodology is efficient, it handles only a subset of HTML tags. Our work can be considered as an extension of the approach outlined in [6].

IBM has devised a novel application called *WebSphere Transcoding Publisher: HTML-to-VoiceXML Transcoder* [10], which is capable of converting HTML web pages to VoiceXML. The transcoder requires the user to insert *hn* and *link* tags to identify header sections and links. The disadvantage of this approach is that a naive user who does not know much about HTML would find it difficult to re-author the HTML source page. Moreover the high cost involved in purchasing the product prevents ordinary users from using the product.

1.3 The Thesis

In this thesis ‘An Extensible Transcoder for HTML to VoiceXML Conversion’, we propose an efficient scheme to convert HTML files to corresponding VoiceXML files on the fly. The scheme is a two step process. In the first step we convert the HTML file in to a HTML node tree and in the next step we traverse the HTML tree in a recursive fashion and convert each HTML node into a corresponding VoiceXML node. The conversion from HTML node to VoiceXML node is always not trivial since there is no equivalent VXML node for every HTML node and vice versa. The most prominent feature of the transcoder is that the user can specify his own phrases for particular HTML tags. This would be dealt in more detail in the following chapters.

1.4 Research Objectives

In our thesis, we strive to achieve the objectives mentioned below:

- The sole purpose of the developed transcoder is to provide the visually impaired/mobile users a means to access the Internet anytime.
- It is evident that the structure of a HTML page contributes a lot to the understanding of the page by the user. Proper organization of the web page and the layout of information helps the user in assimilating the required information quickly. Since the structure of the page is a key component to the comprehension of visual document, it is of immense importance to convey this to the user of voice browser. So it is easy to conclude that a plain text-to-speech conversion is not enough to present a web page through voice. Our system strives to reflect the HTML document’s structure.
- Certain information provided by a web page, like the background image, animated cartoons, banners etc contributes to the understanding only when viewed visually. These elements can be skipped by the audio browsers since trying to convert these as voice could end up distracting the listener from understanding the core content of the web page. Thus the application developed for HTML conversion should provide means to ignore such extra information.

1.5 Contributions

The transcoder developed by our methodology converts any HTML (version 4.0 or lower) file to the corresponding well formed VoiceXML file. Our transcoder provides means to tailor the transcoding logic of the application through an user interface sheet.

Chapter 2 in our thesis introduces the basic concepts of the VoiceXML language and draws a comparison between the structure of HTML and VoiceXML documents. Chapter 3 presents an overview of the system model and enlists the assumptions made by the application. Chapter 4 discusses in detail, the various phases of the transcoder and provides an idea of the transcoding logic. Chapter 5 concludes the thesis and provides some insight to the future work that is possible in this area.

CHAPTER 2

INTRODUCTION TO VOICEXML

2.1 What is VoiceXML?

As the name indicates VoiceXML is a derivative of World Wide Web consortium's (W3C) XML- 'The Extensible Markup Language'. While XML has become the default standard for representing data and structures on the web, VoiceXML has become a standard for describing voice applications. VoiceXML is a language for creating Human - Computer interfaces through telephone. VoiceXML can be thought of as a markup language for voice, like HTML is for text. VoiceXML is used extensively for speech recognition and application development. The main goal of VoiceXML is to deliver the web content to interactive mobile clients through voice.

Voice XML was developed as a standard by the VoiceXML forum which consists of organizations like AT&T, lucent, Motorola, IBM etc. Since VoiceXML had the technology backing from top organizations, it was able to implement voice enabled web applications in reality.

2.2 VoiceXML Features

Just as a user can interact with a HTML page, he/she can also interact with a VXML page. The notable features in VXML are :

1. Recognition of spoken/DTMF (Dual Tone Multiple Frequency) input. Here DTMF refers to pressing Telephone keys.
2. Assigning spoken input to variables in the document and making decisions based on the assigned values to the variables.

3. Playing synthesized speech, audio files with the help of Text-to-Speech (TTS) converter.
4. Linking to other documents/other areas of the same document as an HTML file would do.

2.3 Architecture

The User first contacts the web server requesting for VXML pages. This request is directed to the VoiceXML interpreter context for initial interaction, like recognizing the call etc. Later it is passed to the VXML interpreter which takes care of the dialog to be played which may involve getting inputs from the user. At this point it may involve getting inputs from the user. Here certain grammars (rules to recognize input, discussed later), may be active to validate the input and to switch to another sub-dialog based on the input. The VoiceXML interpreter context also has certain active grammars which may be looking for phrases from the user, which would take the user to a different level, like exiting from the web-page.

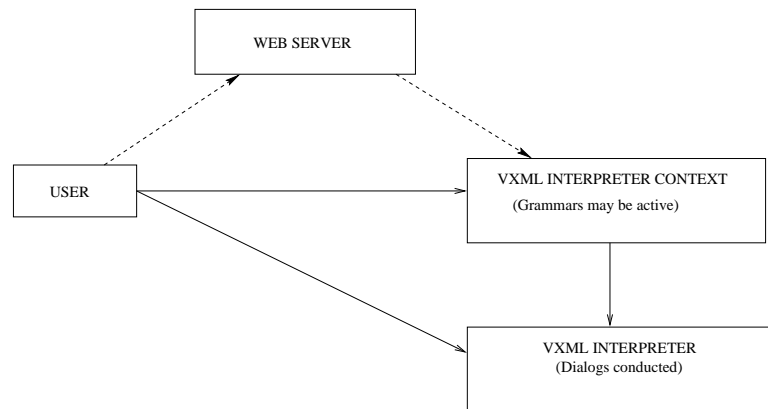


Figure 2.1: System Architecture

2.4 Framework

Any VXML document forms a conversational finite state machine, i.e., the user is in one conversational state at a given time. The entire document is divided into dialogs. In naive terms a dialog can be thought of as a paragraph. Each dialog has to specify its successor, i.e., the flow of execution or control should be specified by each dialog. The

flow is specified by providing URIs. The URI can lead to a external document or to an internal document. If the URI is not specified then the dialog ends there.

2.4.1 Dialogs

Dialogs are basically a set of executable commands in VoiceXML. The entire form is divided into dialogs. The two types of dialog are forms and menus. Forms are used to obtain inputs from the user through voice or DTMF. Menu provides the user with a list of options, and it transits to that URI based on the choice. A sub-dialog is just like a function call in any programming language. A sub-dialog is called from a dialog and then the control is shifted to the sub-dialog and once the sub-dialog is over, the control is returned to the dialog which invoked the sub-dialog.

2.5 Grammar

The most important aspect when obtaining inputs from users is checking the validity of input. This is done by what are called Grammars. Grammars are used for specifying the speech recognition software, the combination of words and DTMF tones that it should be listening for. It is clear from this fact that each field item in a form needs to have a grammar of its own. For example if the question is “say the state of residence in US,” the valid answer to this would be the name of any one of the fifty states in the united states. Hence a grammar for this application would check the user’s input with the names of fifty states. The input is determined valid if it matches any one of the words in the grammar.

Writing a valid grammar is the most important part in developing a VoiceXML application because, an improperly written grammar would make the user repeat inputs several times. Grammars can be of two types, internal (in-line) and external. In-line grammars can be directly embedded in the VoiceXML code with the `<grammar>..</grammar>` tag. These grammars would contain a set of words with which the user’s input would be matched. These type of grammars can be used for inputs with small sample space. External grammars are specified in the ‘SRC’ attribute of the grammar element :

```
< grammar src = “state.gsc” >
```

These grammars are basically used for inputs with very large sample space. The main advantage of having an external grammar is that it can be reused easily by specifying the URI. External grammars are usually written with tools like Nuance Grammar Specification Language (GSL) or Speech Grammar Markup Language (GRXML).

2.5.1 Scope of Grammars

The beauty of VXML is that more than one grammar can be active at a time. Since any VoiceXML page is organized in a hierarchical fashion, starting with a root (<vxml>) element and descending down to <form> and <field> elements, grammars can be associated with any of the subtrees. A grammar which is defined at the root level (immediately within the <vxml> element) is active throughout the document. Similarly a grammar defined within the <field> element is active only within the field. Hence for getting specific user inputs like ‘name of the city’, grammars can be defined within the field. For instances like “exit” (when the user wants to exit from the web-page) and “operator” (when the user seeks human intervention) which may need to be valid anytime has to be defined at the root level. So the grammars have to be used judiciously and accurately.

2.6 Elements of VXML files

A VoiceXML document is highly structured and the entire document can be realized as a hierarchical tree. The most basic element of any VXML application is the document (<vxml>) element. Every VXML application starts with <vxml> and ends with </vxml> element. The <vxml> encapsulates one or more dialog within it. Dialogs are basically <form> or <menu> elements. The document can also contain some other elements like :

<meta> - Information about document/author

<var> - Declaring certain variables

<script> - containing ECMA script fragments

<catch> - Managing events

<link> - Global grammar declaration , transition to another document

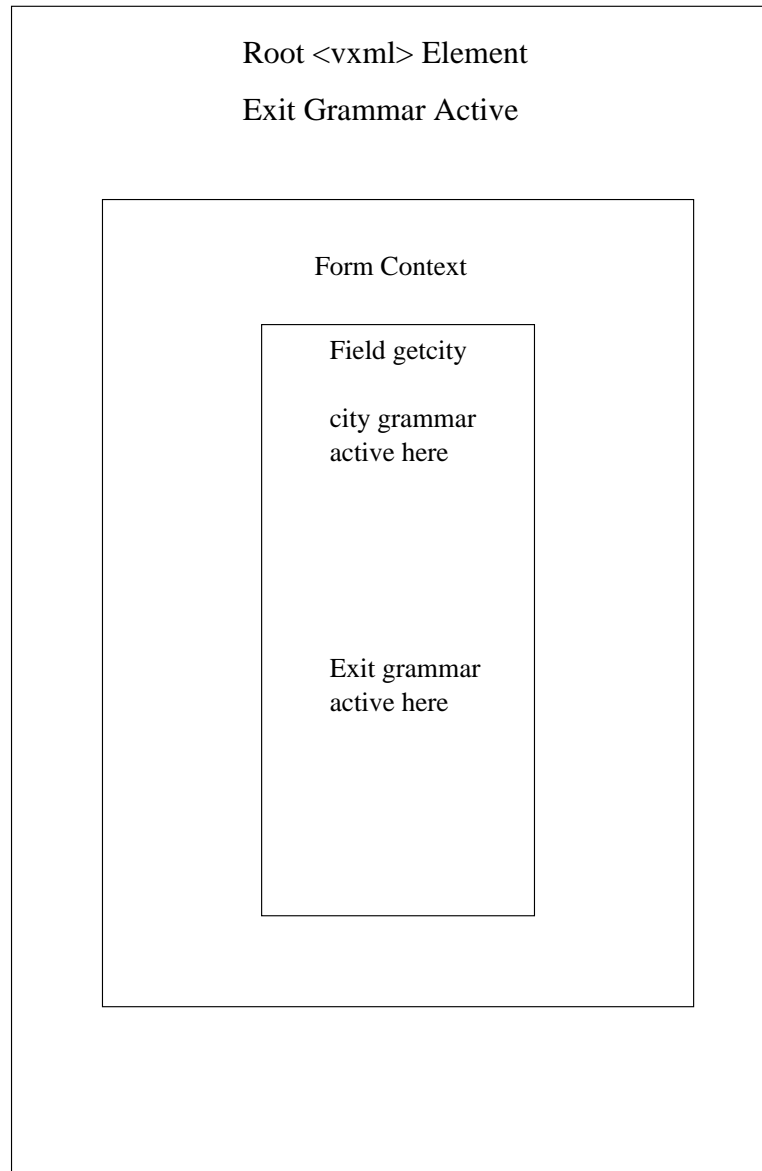


Figure 2.2: Scope of VXML Grammars

`<grammar>`- Specifying in-line Grammar.

Each of these elements can internally hold only certain specified elements. For instance a `<prompt> —text —</prompt>` element is used to read out enclosed text, or prompt the user for providing inputs. Thus element can occur only within the `<field>` element, i.e., the `<field>` element is used to describe a input variable and assign value to this variable by getting input from the user. The correct way to use `<prompt>` is :

```
<field name='`city`'>

    <prompt> Enter city name </prompt>

    <grammar>

        ..

    </grammar>

</field>
```

2.6.1 Event Handling

VoiceXML defines a mechanism for handling events which may be thrown as a result of improper user behavior. For instance if the user is prompted for providing an input and if he/she keeps silent without interacting, the application could get blocked for ever. Such situations has to be handled appropriately. A methodology employed for such scenario is the event handlers. The system throws events when users do not respond, or when they don't respond properly, or needs some extra help, etc. These events are caught by the *catch* elements. The default event handlers provided by the system are: `<catch>`, `<error>`, `<noinput>` and `<nomatch>`. The event handlers execute the set of commands enclosed within these elements.

2.6.2 Links

Links are basically used for providing transfer to other parts of document. They specify a grammar and if any of the user input matches that grammar then the control is transferred to the link's destination URI.

2.6.3 Forms

Forms are the most important component of any VXML document. Form is the platform by which the user provides input and obtains output from the system. A form basically contains 'form items'. 'Form items' can either be field items or control items. In general a form contains the following:

1. Field and control items.
2. Set of procedural logic that needs to be executed when a <filled> action takes place.
3. Event Handlers.
4. Description of non field items.

Every form needs to have a name. A form is given a name through its 'id' attribute.

2.6.4 Field Items

Field items are used for collecting specific field inputs from the user. For example, collecting name of a state from the user will look like

```
<field name='state'>
    <prompt>.....text.....</prompt>
    ..
</field>
```

Field items have the following:

1. Prompts to tell the user about what they want ?
2. Event handlers to handle situations like no input from the user, etc.
3. Set of statements to execute after the field element gets a value.

2.6.5 Control Items

A set of statements within the `<block>...</block>` tags would be executed in a sequential manner without any interruption. Thus the `<block>` tags can be termed as control tags.

2.6.6 Form Interpretation

VXML forms are interpreted in a specific manner. A Form Interpretation Algorithm or the FIA process the form in the following fashion. The FIA has a main loop which repeats continuously by selecting one field item at a time.

1. Select a field item whose value is not set or assigned. (i.e., the field item variable does not have a value)
2. Play the corresponding prompts and collect input from the user.
3. Execute any `<filled>` action statement or event handling if present.
4. Go to the sub-dialog or URI specified by the `<goto>` element.
5. If no direction is provided, go to the main loop, select the next available field item whose value is not set and execute the prompts associated with the field.
6. If no such field item exists, exit the algorithm and stop execution.

2.6.7 Menus

As the name indicates, `<menu>` tag is used for providing users with a choice of inputs. The `<enumerate>` tag is used to read out the choices in an ordered manner. The `<field>` element is used to capture the selection of choice. Then the transition is made using the

<goto next> element based on the user's selection. The transition can be either to an external document or to a sub-dialog within the same document.

2.6.8 Prompts

The prominent advantage of VoiceXML is that it is able to deliver web content in audio form. Hence, in most cases it needs to read out the text from web pages. The <prompt> </prompt> and <block> </block> tags in VXML reads out any text enclosed within it. The <prompt> has many children to make the audio more realistic. For example any text enclosed within the <emp> ... </emp> tags will be read out with a stress and any text which is enclosed within <sayas type=""> will be read out in a manner corresponding to the type. For example a phone number say 972-235-6377 will be spelled out as "nine seven two..."etc if we specify the type as 'phon'. The prompt tag has numerous other features which makes the interaction with the user more realistic.

The above mentioned aspects are among the few prominent ones, which a primary Voice XML user should be aware of. Since our system converts an HTML file to its corresponding voice XML counterpart , we next give a brief introduction to HTML.

2.7 HTML 4.0

Our system handles any input file built with HTML 4.0. the HTML tags can be categorized based on many factors. Based on their functionality they can be categorized into the following groups.

2.7.1 Table

Tags which are categorized under the type text, formatting, graphical elements are used for enhancing the display of the text in the web-pages. Tags under the link type are used to provide links (or) paths to other parts of the same document or to an external document. Table tags are used to represent the information in a tabular format. The basic tags are used to maintain the syntax and format of HTML. The form tags are the most important of all for the reason that they are used for interactive communication with the user. Since VoiceXML

Table 2.1: Overview of HTML Tags

| Category | Example Tags |
|-------------------|--|
| Basic Tags | <html>,<head>,<body>,etc |
| Header Tags | <meta>,<title>,etc |
| Text Tags | <pre>,<h1>.....<h6>, |
| Links | <a href ...> |
| Formatting | <p>,
,,,,etc |
| Graphical Element | ,<hr> |
| Tables | <table>,<tr>,<th>,<td> |
| Frames | <frameset>,<frame>,<noframes> |
| Forms | <form>,<option>,<input>,<textarea>,etc |

is a language used for representing web-content through audio, it involves a great amount of interaction with the user. Therefore, in this section we shall discuss briefly about the HTML forms.

2.7.2 Forms

An HTML form can be termed as a section of the document which contains normal content (text), markup elements, controls (radio buttons, check-boxes, menus, etc) and labels (which provide name to the controls). Users generally interact with the forms by entering text or selecting some of the options presented by the controls. When the user finish filling the inputs he/she can submit the contents by activating some controls. For example, the “submit” button which is of type input control is used to send a set of inputs to a URL or CGI specified by the form. Form also contains reset buttons which are used for clearing the input fields which may have been filled by inputs provided by the user earlier.

2.8 Comparison of HTML versus VoiceXML

Since VoiceXML documents contain only forms and blocks, it is sensible to compare HTML and VoiceXML in terms of forms. Basically, forms are used in HTML for obtaining user input. In VoiceXML also the user interaction is provided through forms, but this form is not as powerful and effective as the HTML form, for the reasons mentioned below. Any HTML document is a single unit which is fetched from an URL and presented to the users as it is, with its full efficiency. But VoiceXML documents contain a number

of forms and dialogs and each has to be delivered to the user through audio in a sequential manner. This is due to visual medium's capability to display numerous items in parallel and inherently sequential nature of the voice medium. Hence even though it is claimed that the HTML form can be converted to its equivalent VoiceXML counterpart, it is structurally different and has some shortcomings.

CHAPTER 3

SYSTEM MODEL AND ASSUMPTIONS

In today's environment, most of the companies provide customer service through web and telephone. These companies have to provide information anytime in order to retain their customers. And most important of all, these services have to be provided at a very low cost possible. The perfect solution for this problem is to have an application, that can tailor content to pervasive devices, by translating markup languages and image format in one form to equivalent content in other form. Such applications can be called as transcoders. If the companies were to provide *anytime customer service* without a transcoder, they would have to store information in numerous formats, which may prove costly.

In order to provide web-content over telephone, we need an interactive voice response (IVR) platform which has audio input, audio output, service logic and transaction service interface. The IVR needs a domain specific language for voice dialogs. The IVR can interpret this dialog language and act like a client to the web-server. The IVR platform can then translate an incoming request to a URL, fetch the document, interpret it and return the output to the mobile client. The dialog language used for this purpose is VoiceXML.

For all this to work, it becomes imperative that a transcoder capable of converting HTML to VoiceXML is needed. Our methodology proposed here is one of the easiest way to convert a HTML page to well formed VoiceXML page. This VoiceXML content can then be used by the IVR to interact with the mobile client.

3.1 Architecture

The components of the transcoder are the following:

- Input provider

- Parser
- Translator
- Interface sheet

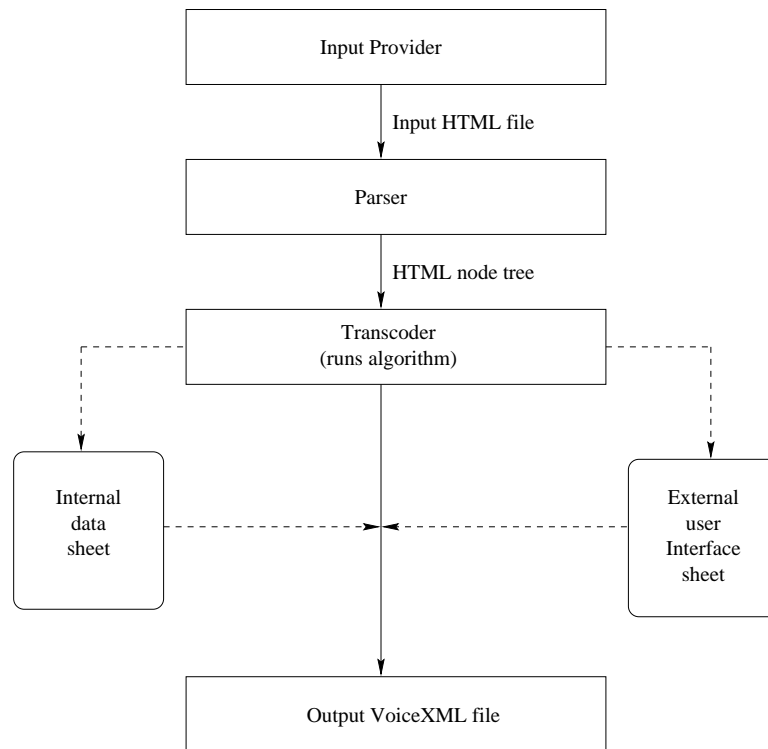


Figure 3.1: System Model

The transcoder developed by our approach has two phases. The first is the parser phase and the next is the translation phase. In the first phase, the input html file is parsed and a html node tree is obtained. In the second phase, this node tree is passed as input to the transcoder. The transcoder runs an algorithm and produces the corresponding VoiceXML file as output.

The input provider takes in any HTML file as input and passes it to the parser. The prominent component of our transcoder is the *Interface sheet*. It is used as an instruction medium between the system and the user. The model of the Interface sheet is shown in Fig. 3.2

The Interface sheet is basically divided into two blocks. One is the component block and the other is the tag block. By providing instructions in the attributes and output text

| Input Attributes | |
|--|--|
| Input duration in seconds for Text-box :
Input duration in seconds for Text-Area :
..... | |
| HTML Tags | Output Text |
| <blockquote> | Starting of the text quoted from elsewhere |
| </blockquote> | Ignore |
| | |

Figure 3.2: Structure of Interface Sheet

section, the user can instruct the system as to how to treat the corresponding components. For example, consider an input HTML file containing a ‘text box’ field. Just as an user would provide a text input to the HTML browser, he/she would have to provide a voice input to the IVR. On recognizing the text box field, the IVR would tell the user, “please provide voice input for this field”, then the user would speak and the input voice would be recorded. Now the attributes for this dialog, like providing a beep after the question, restricting the size of the spoken input etc. could be provided by the user, by filling the attribute field in the Interface sheet. Similarly, the user can provide the text that he wants to be spoken out, as a replacement for some of the html tags. If he does not alter the “output text” section, then the default text would be spoken out.

3.2 Assumptions

The transcoder developed by our methodology is capable of converting any HTML file to corresponding VoiceXML file, however it has some limitations and restrictions.

- The input HTML file should be a well formed document, in the sense, every open tag should have a corresponding close tag.

- The input file should to be error free (i.e., a file which can be viewed through a desktop browser without errors).
- The input file should not have any tags that are not defined in HTML 4.0. When HTML files are viewed through certain browsers, the browser inserts many special tags which may not be found in the HTML specification. However, these tags can be removed by performing some pre-processor operations in the input file.

CHAPTER 4

TRANSCODER LOGIC

As mentioned in chapter 3, the transcoder developed by our system is organized into four components, viz. the input provider, the parser, the translator and the interface sheet. The working of the transcoder can be defined in two phases, the parsing phase and the translation phase. We shall discuss each phase in detail.

4.1 Parsing Phase

In order to convert any file from one format to another, its syntactic structure has to be understood. An HTML file cannot be converted into VoiceXML file on a sentence by sentence or tag by tag basis. The HTML file's structure should be captured by a node tree and then it has to be transferred by identifying the subtrees. This can be illustrated by a small example,

HTML file:

```
<html>
  ..
  <h1> form example </h1>          -----> I
  ..
  <form>
  <label> Firstname </label>       -----> II
    <INPUT type="text" id="firstname">
  </form>
  ..
</html>
```

The output of this html file would display *form example* in bold and *Firstname* in ordinary font. If the above html file is treated on a tag by tag basis, then *form example* in line I and *Firstname* in line II would be treated in a similar manner. But when we translate this page into VoiceXML, we would be hearing this page in audio. Hence we may have to provide more stress when speaking out *form example* since it may be a heading of a paragraph, (since it is enclosed between `<h1>` and `</h1>`). The other aspect is that since ‘Firstname’ is enclosed within `<label> ..</label>` tag which is treated as the label for input field *firstname*, it cannot be spoken out plainly. It has to be spoken out in the following manner, “Please provide input for Firstname”. Otherwise, the user may not be aware that the system is expecting input from him/her. In brief, the HTML page’s layout or structure should be understood precisely before translation. The only way to achieve this is to parse the HTML file and obtain the node tree in a structured form.

The structure of an HTML file can be obtained by parsing the HTML file and obtaining the parse tree in an hierarchical node form. In our application, the HTML parsing is done with the help of wise-system’s web-wise package. Web-wise is an HTML template parser which takes in an HTML file and outputs a HTML node tree. An node tree is a tree type data structure of ‘HTML nodes’. Each node abstracts an HTML tag. Each of the nodes can have sub-tags and this goes on recursively. The basic structure of a HTML file’s node tree is shown in the Fig. 4.1

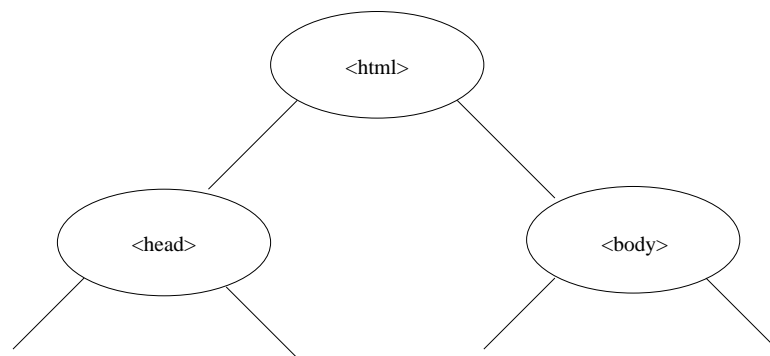


Figure 4.1: Example Html Node Tree

Here, the `<head>` and `<body>` nodes are children of the `<html>` node. Each node can have its own children and one parent. The parse tree throws away very little information contained in the source file, therefore by traversing the parse tree, an almost identical copy

of the input document can be obtained. The only source information discarded by the parser is whitespace in between tags (i.e., spaces or newline between the attributes of a tag). The parser generates trees based on HTML document type definition (DTDs). Let's analyze a sample parsing done by the webwise package.

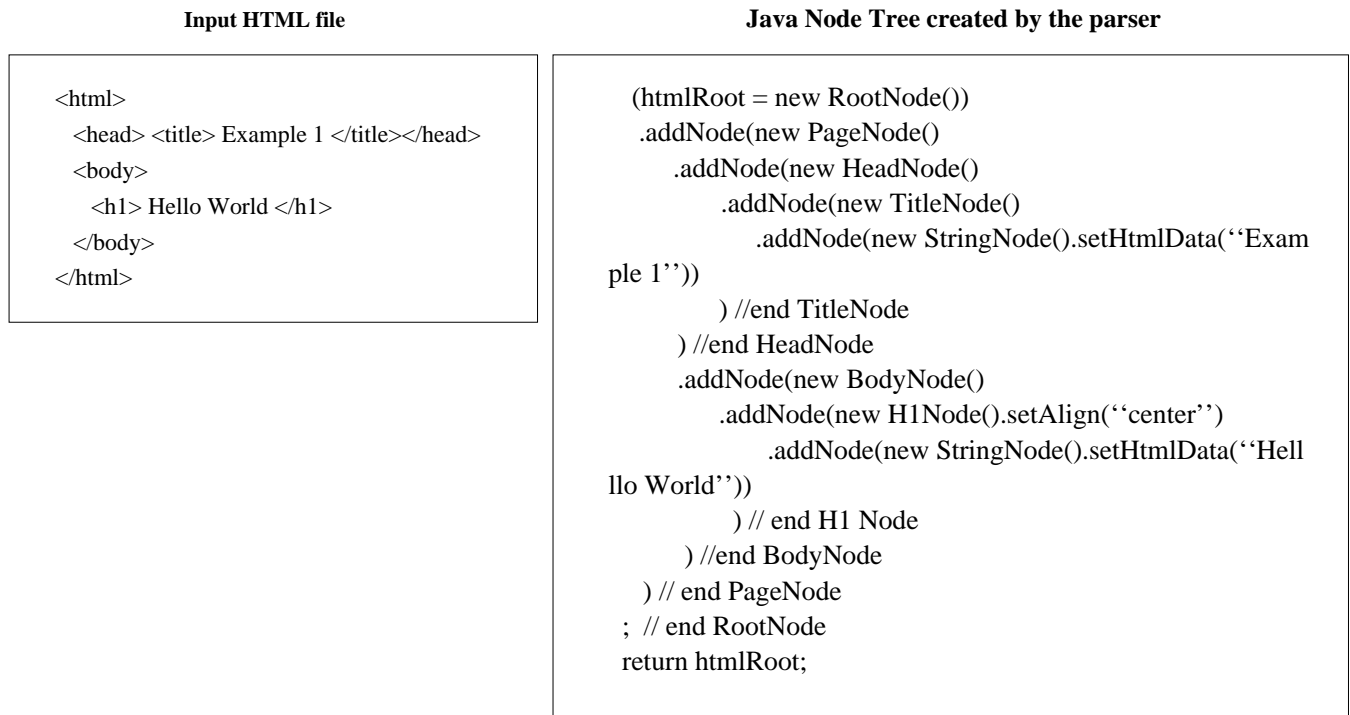


Figure 4.2: HTML Parser - Example conversion

4.2 Translation Phase

The second phase of the application is the translation phase. In this phase the output from the parser (previous phase), i.e., the input HTML file's node tree is traversed in a "left to right depth-first manner" and the corresponding VoiceXML is produced for each and every visited node. The translation of HTML node to VoiceXML node is not straightforward or simple, for the reason that the structure of HTML and VoiceXML are entirely different. Browsing a web-page visually differs remarkably when compared to browsing in an audio-only environment. Thus even though a rich multimedia presentation of a web-page cannot be given, our method tries the best possible ways to make the user comfortable with the environment.

4.3 Structure

The main component of the transcoder is the *HTML to VoiceXML core converter*. This contains the transformation logic that needs to be applied to each and every HTML tag. The core converter uses two sheets for data storage. One is the internal data sheet and other is the external user interface sheet. The internal data sheet contains a list of HTML tags and the corresponding VoiceXML tags which involves complex transformation logic to be applied before obtaining the VoiceXML output. The data contained in this sheet is not provided for the interaction of users, thus these set of HTML tags gets transferred in a standard manner. The external user interface sheet basically contains HTML tags which are used for text display or appearance of the web-page, etc. These tags adds more value to the web-page only when they are viewed visually. These tags cannot be transferred to audio with much sense. For example consider a web-site of a movie, it would have numerous pictures, flashy colors, blinking cursors, animated images, etc. These look good only when they are viewed visually, but if the same page is converted to audio, it will be something like “This page has an orange color background with red spots scattered and a doll dancing all the way etc”. This information is totally unimportant or useless. So the users can be given the opportunity to tailor the conversion of such HTML tags. So these HTML tags, along with the default conversions, are provided in the external user interface sheet. Some of the entries in the sheet are shown in Table. 4.1

Table 4.1: Elements of Interface Sheet

| HTML Tag | Text to be Spoken |
|---------------|--|
| <blockquote> | Beginning of the phrase which is quoted from elsewhere |
| </blockquote> | Ending of the phrase which is quoted from elsewhere |

Table. 4.1 implies that the appearance of the <blockquote> in the input HTML file will be substituted with the audio comment “Starting of the phrase that is quoted from elsewhere”. This can be changed by the user by providing an alternate comment.

The core converter produces VoiceXML statement for every HTML node by referring either to the internal data sheet or to the external user interface sheet.

4.4 Framework of the Overall Process

The root node of the input HTML node tree is passed to the translating processor. The translator performs the initial actions that need to be done for the particular HTML node. The initial action may involve producing a VXML tag or setting/resetting of some flags etc. Then the first child of the node is obtained, and is passed to the transcoding processor recursively. If there are no more children nodes then this node is processed, and the final actions are performed. The final actions may be a set of similar actions as the initial action. Then the recursive invoking ends and the control is returned to the parent caller. If the root node does not have any children to process, then the final actions for the root node is performed and the final VoiceXML output file is obtained. Then this file is passed to a VoiceXML syntax checker. This may be any VoiceXML compiler like tellme, heyanitha etc(VoiceXML compilers available in the market). The file coming out of the syntax checker would be the input HTML's syntax-free VoiceXML counterpart. The process is illustrated in Fig. 4.3

4.5 Translation Issues

There are a few basic issues that needs to be addressed while converting a HTML file into a VoiceXML file. They are:

1. Our model strongly believes that every open tag (e.g., <form>) would have a corresponding close tag (e.g., </form>). This assumption is very important since we would set some flags on the occurrence of some of the open tags and similarly reset the same flags while encountering the close tag. For example, for detecting the various field items inside a form, we set the form flag when '<form>' element is encountered. So the system definitely expects a '</form>' tag to reset the flag.
2. As we are aware that, interacting with a voice browser is not as trivial as interacting with a visual browser, we need to address an important issue at this point. Let's assume a user is filling a form which contains many input fields. In the case of a visual browser, he/she can see all the fields at the same time. Thus he/she can check his inputs manually before submitting it to the server. But in the case of a voice

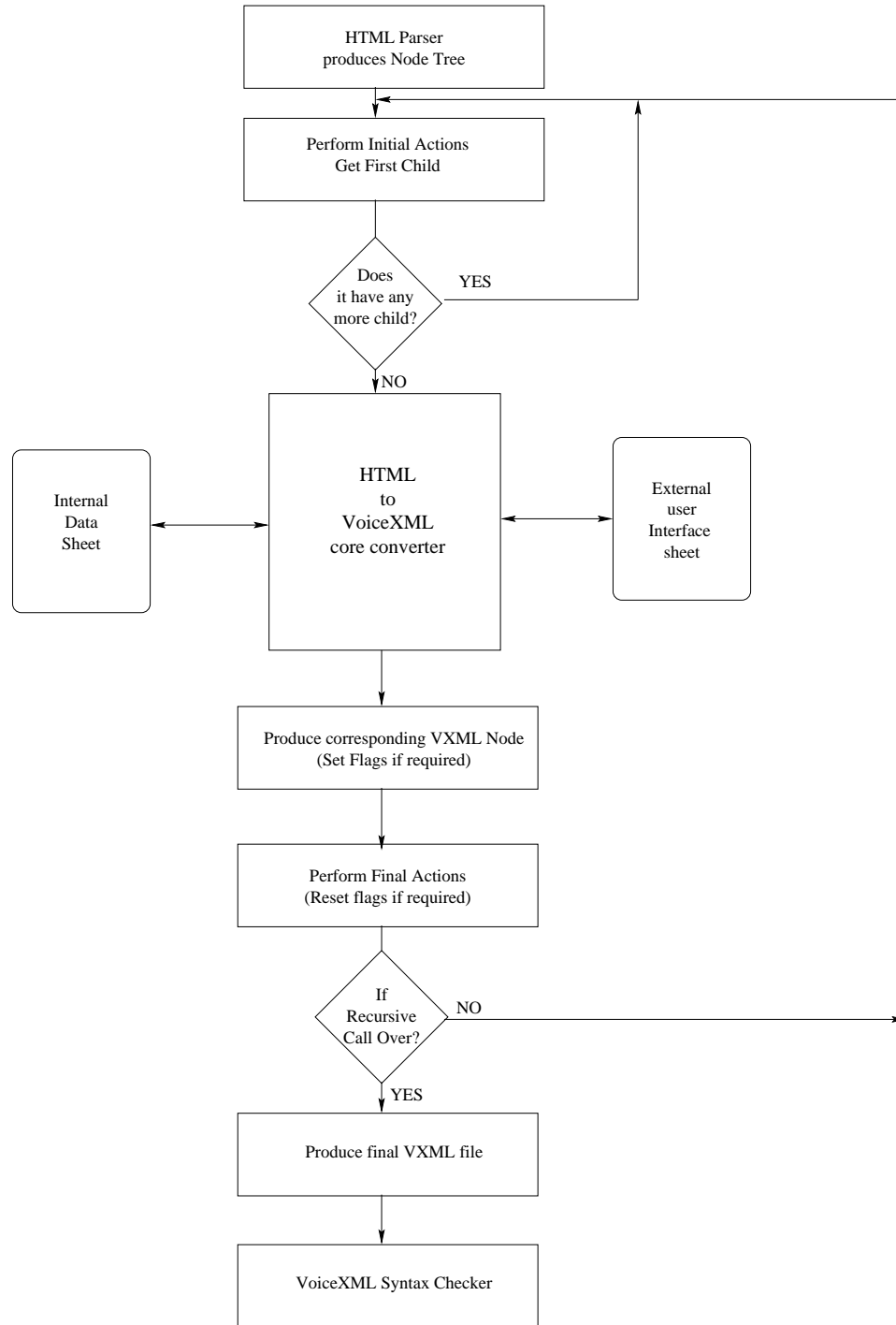


Figure 4.3: Overall Process

browser, since the commands are spoken sequentially, it is very difficult to keep track or remember the already filled-in inputs. Thus it becomes necessary that the user's consent has to be obtained before submitting the input items. For this purpose, all the input item values have to be spoken out to the user, his consent obtained and only then it can be submitted to the server.

3. Since VoiceXML is a derivative of XML, it inherits the nature of being highly structured. Thus, the position of every node or tag in VXML is restricted by a set of possible parents and a set of possible children. So even though a VXML node can be formed with correct syntax, the compiler may throw an error if the node's location (parent or child) is wrong. For example the VoiceXML statement

```
<prompt> It is a beautiful city </prompt>
```

is syntactically correct but the compiler will throw an error if it is used in the following manner

```
<form>
```

```
    <prompt> It is a beautiful city </prompt>
```

```
    ..
```

```
</form>
```

The `<prompt>...</prompt>` can appear only within a field or a block, since it is not a child of the `<form>` element.

4. While translating HTML to VoiceXML, it may so happen that most of the times an one-to-one conversion is not possible. In most cases, either one HTML tag has to be replaced by two VoiceXML tags or, two HTML tags will map to the same VoiceXML tag. Hence, the translator logic has to be designed in such a way that these features are realized.

e.g., `<h1>` has to be translated to `< prompt> <emp>`, since `<prompt>` is for speaking and `<emp>` is for stressing the words.

5. One very important issue that has to be resolved while transcoding HTML to VoiceXML is that every VoiceXML input needs a valid grammar to cross-check and validate the input provided by the user. In HTML, a textbox input can be defined to accept any text within a limited size, but that cannot be done using VoiceXML. Every input has to have a valid grammar (refer Sec2.5).

4.6 Translator Logic

As said earlier, the HTML to VoiceXML core converter is the most prominent component of the entire application. So the *translation logic* which drives the converter is the most important of all. It contains the methodology for converting all HTML tags into their VoiceXML counterparts. In this section, we shall discuss about the manner in which some complex (or) important HTML tags are handled. Due to the diverse functionalities of the HTML tags, they are organized into categories. Let us deal with each category sequentially.

4.6.1 Form Tags

Form tags are the most important and most complex to deal with. The nature of its complexity is due to the interacting feature it provides to the user. Thus, it has to behave with respect to the user inputs.

Form

The HTML form tag contains the URL (Universal Resource Locator) of the CGI (Common Gateway Interface) or the web-page to which the form elements have to be submitted and the ‘method’ in which (GET or POST) they should be sent. The method basically specifies the type of encoding that needs to be used while sending the inputs. The VoiceXML form does not have these two attributes. Instead, it needs to have the *name* attribute since, every form has to be identified with a unique name (the form name is used for transition between forms). In VXML, the URL and ‘method’ for sending the inputs has to be used only with the ‘submit’ tag. So the following actions are taken when the <form> tag is encountered:

1. The ‘URL’ and the ‘method string’ is stored in a separate global variable to be used

later along with the ‘submit’ tag.

2. An automatic name is generated for the form.

Input types

radio button

In HTML, the radio button is used for providing the user with a list of options but allowing him to select only one among that. In HTML, the radio tags does not have a closing tag. This feature of the radio tag causes problems when it is translated to VoiceXML. Take the following example.

```
<INPUT type = radio name = "sex" value = "male"> Male <br>
```

```
<INPUT type = radio name = "sex" value = "female"> Female <br>
```

This creates two check-boxes with the value field besides them. When the user selects one of the check-boxes, the other becomes inactive. Thus only one value of the ‘sex’ item will be submitted finally. To obtain all these features in VoiceXML, the following logic is employed. When the first radio tag is encountered, the field tag is declared. The name of this field and the value are stored in a separate structure. The user is also prompted about the presence of options and instructed to select only one among the options. The options are read out one after another. Now, the challenge is to recognize the last radio tag of this type and close the field item. For this purpose, the next sibling (nodes in the same level in a tree and having the same parent) is obtained and checked whether it is a radio input type and if so, its checked whether its of the same type. When its not the same, the field item is closed. An example conversion is shown in Fig. 4.5

text box

In HTML, the input type ‘text’ is used to get a text input from the user. The maximum allowable length of the input is specified by the user (number of characters) and he is allowed to input any set of characters within this limit. This type of input is used for getting a name of a person/the color he likes etc. Hence if we have to implement a similar feature in VXML, we should not have any grammar to evaluate the input. Thus we make use of

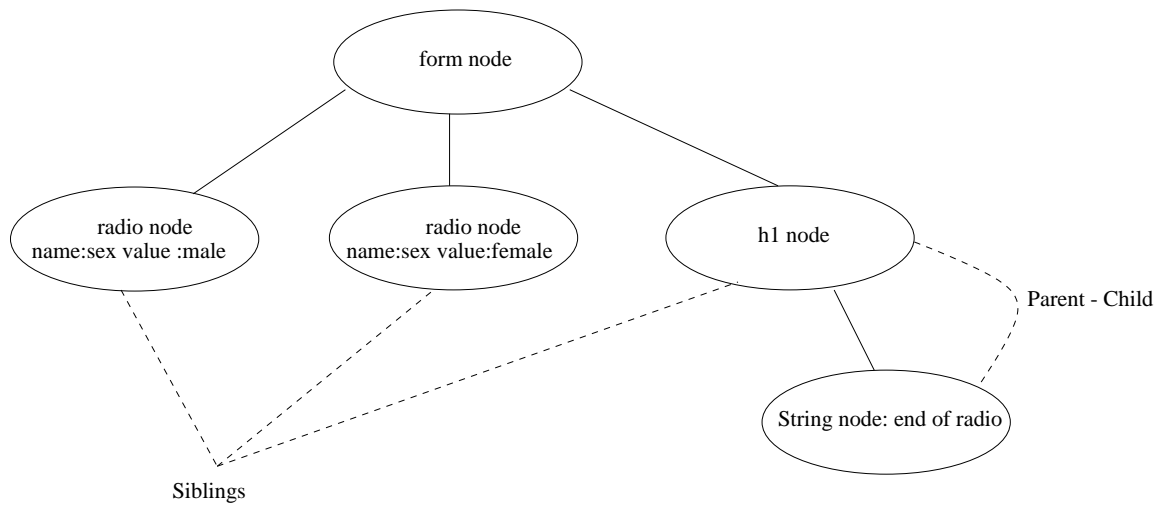


Figure 4.4: Form node - Example of radio type

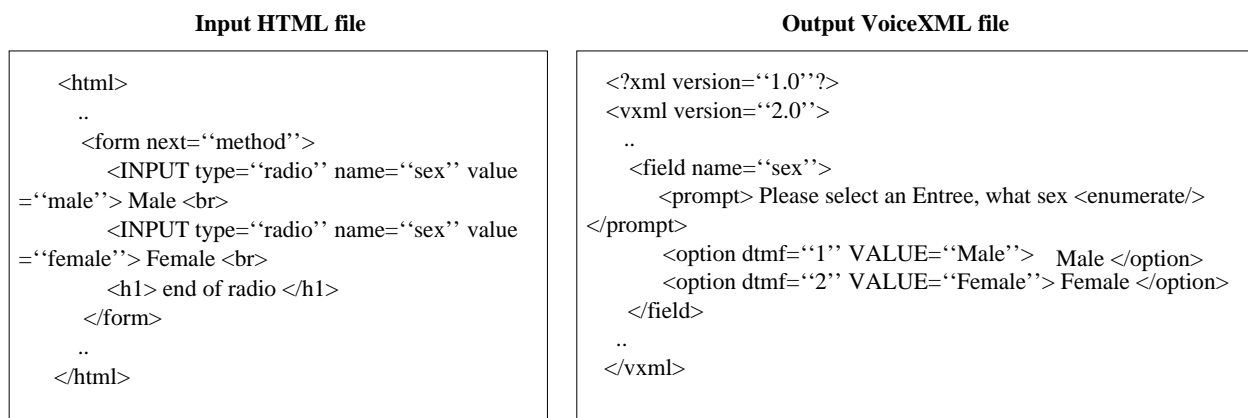


Figure 4.5: Example conversion of radio Tags

the <record> element in VXML. The system allows the user to specify the maximum allowable length of the input by specifying the ‘seconds’ (Time for recording) in the external user interface sheet. An example is shown in Fig. 4.6

| HTML Input file | VoiceXML Output file |
|--|---|
| <pre data-bbox="207 470 831 995"> <html> .. <form action="html://www.utdallas.edu/cgi-bin/cgiwrap/narayan/ multiForm.cgi" method="get"> <LABEL for="firstname"> Firstname: </LABEL> <INPUT type="text" id="firstname">
 <LABEL for="lastname"> Lastname: </LABEL> <INPUT type="text" id="lastname">
 </form> .. </html> </pre> | <pre data-bbox="889 470 1502 1327"> <?xml version="1.0"?> <vxml version="2.0"> .. <form id="f2"> <record name="firstname" beep="true" maxtime="10s" finalsilence="4000ms" dtmfterm="true" type="audio/wav"> <prompt> At tone, speak First name: </prompt> <noinput> I did not hear anything. Please try again. </noinput> <filled> <prompt> Your input is <audio expr="firstname"/> </prompt> </filled> </record> <record name="lastname" beep="true" maxtime="10s" finalsilence="4000ms" dtmfterm="true" type="audio/wav"> <prompt> At tone, speak last name: </prompt> <noinput> I did not hear anything. Please try again. </noinput> <filled> <prompt> Your input is <audio expr="lastname"/> </prompt> </filled> </record> </form> .. </vxml> </pre> |

Figure 4.6: Example Conversion - Input Tag *text*

The text area is also treated in the same way, except that the default value for recording is more than that for textbox.

submit

The <submit> tag in HTML is used to send the set of collected form inputs to the URL using the method specified in the form element. In our system, since the URL and method is stored in the global variable, it is used for submitting the inputs. In VoiceXML, the field elements which are submitted along with the submission should also be listed out. This can be achieved in the following way: whenever the <form> flag is set, the inputs are stored in a separate array. This global array is listed out at the time of submission. The

global array is re-initialized to null when the <form> flag is reset.

reset

Before every submission, the user is prompted about all the field values that he/she supplied previously and asked whether the elements can be submitted. If the answer is 'yes', then submission takes place, else the global array is re-initialized and all the field elements are set to null. Due to this the FIA (refer Sec2.6.6) starts evaluating each and every field in sequential order.

4.6.2 Link Tags

In HTML, transition from one part of a document to a different part of the same document or to an external document is done using the '<a href ..>' tag. Our system provides this feature in the output by using the following technique. The '<a href ..>' tag can provide links in two ways:

1. To a different part of the same document.
2. To a different document altogether.

In the first case, our system uses the <goto next = "URL"> tag in VXML. The URL here is the VoiceXML counterpart of the HTML URL present in the input HTML file. The target URL is converted to VXML by our system, by storing the URL address in a global array and processing each URL in the array at the end.

Second case is also accomplished using the <goto next = "sub-dialog name">, but here the 'sub-dialog name' is assigned the name of the dialog which contains the target statements.

4.6.3 Image Tags

In HTML, the image tag is used to insert an image in the web-page by specifying the height and width of the image. Since image cannot be converted into voice, VoiceXML can only support the image by speaking out the attributes of the image. Our system reads out the

source of the image, height and width of the image and name of the image. The system obtains all these information from the attributes of the image tag.

4.6.4 List Tags

The list tags (``, ``, ``, `<dl>`, `<dt>`, `<dir>`, `<menu>`) in HTML are used to present the text in a sequenced ordering. For example the `<dl>` and `<dt>` tags are used to define a set of definitions in a sequential order. The `` tag is used to define the text as a numbered list and the `` tag is used to order the list in a non-numerical fashion (using just the bullet marks and not numbers). In VXML, all these text can be read out only sequentially. But our system recognizes the importance of these tags and supplies some extra information when these tags are encountered. This is done to make the dialog more meaningful.

For example, lets take the following HTML

```
<ol>
<li> Text 1 </li>
<li> Text 2 </li>
<li> Text 3 </li>
<ol>
```

Our system speaks out the following:

“Beginning of an ordered list

Item 1 Text 1

Item 2 Text 2

Item 3 Text 3

Ending of an ordered list”.

The `<menu>` tag in HTML is also used for listing purpose, hence our system treats the `<menu>` tag in the same way as the list tags.

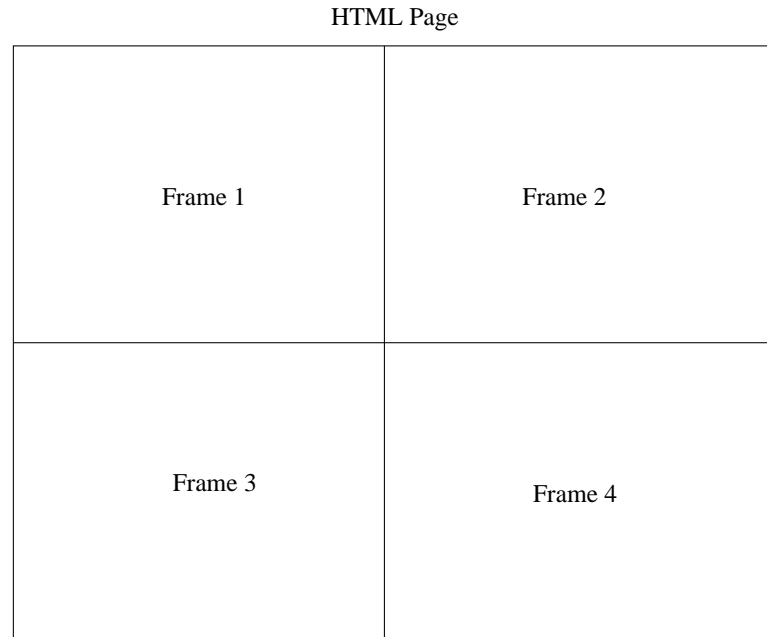


Figure 4.7: HTML Frame

4.6.5 Frame Tags

Frames are a special division of HTML web-page development. The source HTML page, containing frames would just have the declaration of frames and the links to the corresponding URLs. For example, a web page containing four frame declarations, would appear as in Fig. 4.7:

All the web-pages, whose four URLs are declared in the main page would be shown in a single page with demarcation and borders. But notice that the same method cannot be employed using a voice browser, since, VoiceXML pages can be handled only sequentially. Hence strictly speaking, a HTML page containing frames cannot be converted into an equivalent VoiceXML page. Our system employs a technique similar to the one described in section 4.6.2. The URLs of different frames are stored in a global queue and later all the HTML pages (the corresponding URLs) are converted to VoiceXML pages in a sequential manner. Our system uses the `<goto next = "VoiceXML URL">` to visit the first web-page. From that point onwards, the control of execution is left to the voice browser.

4.6.6 Spacing/Layout Tags

The typical spacing/layout tags used in HTML are the `<break>` and `<p>` tags. The break provides a line break and `<p>` starts a new paragraph. These two features are supported using a single VXML tag `<break>`. The `<break>` tag in VXML provides a pause for few seconds before reading out the next line.

4.6.7 Rule Tags

In HTML, the rule tags are used to draw horizontal or vertical lines in the web-page. Basically this can be used as a lower version of the frames. Using these tags, the web-pages can be divided into logical sections. Our system supports this feature by indicating if a horizontal or vertical line is drawn, i.e., when `<hr>` tags are encountered, the following action is taken: the user is prompted that: “ A horizontal line is drawn”

4.6.8 Tables

Table forms an integral part of the HTML pages. They are used to present information in a tabular form. Each table possesses a name (caption), few rows and few columns. The total number of elements present in the table is (number of rows * number of columns). Some of the elements may also be present in the form of a table, so a nested table is possible.

Our system handles the table in a very efficient manner. When our system encounters the following table tags, it takes the corresponding actions as listed in the Table. 4.2

Table 4.2: Table Tags

Table Tags	Action Performed
<code><table></code>	Note down table count, prompt: Starting of table(number)
<code><td></code>	Note down column count, prompt: Column(number)
<code><tr></code>	Note down Row count, prompt: Starting of Row(number)
<code><th></code>	prompt: Heading of Row(number)
<code></tr></code>	prompt: Ending of Row(number)
<code></td></code>	Ignore
<code></table></code>	prompt: Ending of Table(number)

Lets take the example shown in Fig. 4.8:

Text enclosed inside the `<block>` .. `</block>` is spoken out plainly.

Input HTML file	Output VoiceXML file
<pre> <html> .. <TABLE> <CAPTION> This is a caption </CAPTION> <TR> <TD> <TABLE> <TR> <TD> New Table </TD> <TD> Let's see works </TD> </TR> </TABLE> </TD> <TD> Element 12 </TD> </TR> <TR> <TD> Element 21 </TD> <TD> Element 22 </TD> </TR> </TABLE> .. </html> </pre>	<pre> <?xml version='1.0'?> <vxml version='2.0'?> .. <block> Starting of Table 1 </block> <block> Following Sentence is a caption </block> <block> This is a caption </block> <block> Row 1 </block> <block> Column 1 </block> <block> Starting of table 2 </block> <block> Row 1 </block> <block> Column 1 </block> <block> New table element </block> <block> Column 2 </block> <block> Let's see works </block> <block> Row Ending 1 </block> <block> Table Ending 2 </block> <block> Column 2 </block> <block> Element 12 </block> <block> Row Ending 1 </block> <block> Row 2 </block> <block> Column 1 </block> <block> Element 21 </block> <block> Column 2 </block> <block> Element 22 </block> <block> Row Ending 2 </block> <block> Table Ending 1 </block> .. </vxml> </pre>

Figure 4.8: Example Conversion - HTML table

4.6.9 Text Tags

All other tags which are not discussed in the previous sections are categorized mostly as <text> display tags. They are used to decorate the appearance of text in HTML pages. The phrases that needs to be spoken when these tags are encountered can be tailored by the user as per his/her wish. Hence, these tags are listed in the external user interface sheet, along with the default phrases that would be spoken as a substitute for these tags. These phrases could be altered by the user as per his/her will. The user may also choose to ignore some of the HTML tags.

4.7 Pseudo-code of the Algorithm

Having discussed about the specific actions that would be taken for each and every tag category. Let us outline the working of the translator algorithm. The algorithm is designed to execute in a recursive fashion.

1. The algorithm is invoked by processing the root node.
2. Initial actions for the root node are performed.(This may involve setting some flags or printing some VoiceXML tags)
3. The node is sent to the translation logic section.
4. All the children to this node are obtained and stored in a vector.
5. The algorithm is invoked recursively by passing the first child as the argument.
6. If there are no more child for this node, then
 - (a) The final actions for this node may be performed.
 - (b) The control is returned to the parent.

Translation logic section

1. The node is received in the HTML tag form.

2. The 'HTML tag form' is compared with the HTML tags in the internal data sheet. If a match is found, then the new VoiceXML statement is produced using the tag present in the data sheet.
3. If there is no match with the internal data sheet, the node is compared with the elements in the external user interface sheet. If a match is found, then the matched phrase is spoken out to the user.
4. Else, if the node is a special tag then it is treated separately.

The algorithm terminates after all the nodes have been processed. The algorithm is bound to terminate as the match for any HTML node will be found in any one of the three sections (internal data sheet, external user interface sheet, special tags section).

4.8 Implementation Details

The transcoder is implemented using Java with the help of Oracle JDeveloper (an editor for Java). Since the application is developed in Java it is platform independent. The transcoder kit comprises of four files: data.java used for storing the HTML-to-VXML map, interface.txt to provide the user interface sheet, reader.java to read the contents of the interface sheet and transcoder.java which comprises the transcoding logic. The entire application is realized in four classes and ten modules. The class diagram of the application is shown in Fig. 4.9. In an average it takes '5000' milliseconds to transcode a '10 KB' HTML input file to corresponding VoiceXML file.

Some of the limitations of the current implementation are listed here:

1. The input HTML file has to be a well-formed file which uses only the HTML tags defined by the specification.
2. The system does not handle the *check-box input type* provided by HTML. In HTML, more than one value can be assigned to a *check-box field*, but this is not allowed in VoiceXML. The only way to accommodate this feature in VoiceXML is to define a complex grammar which takes a concatenated string as input. However, processing

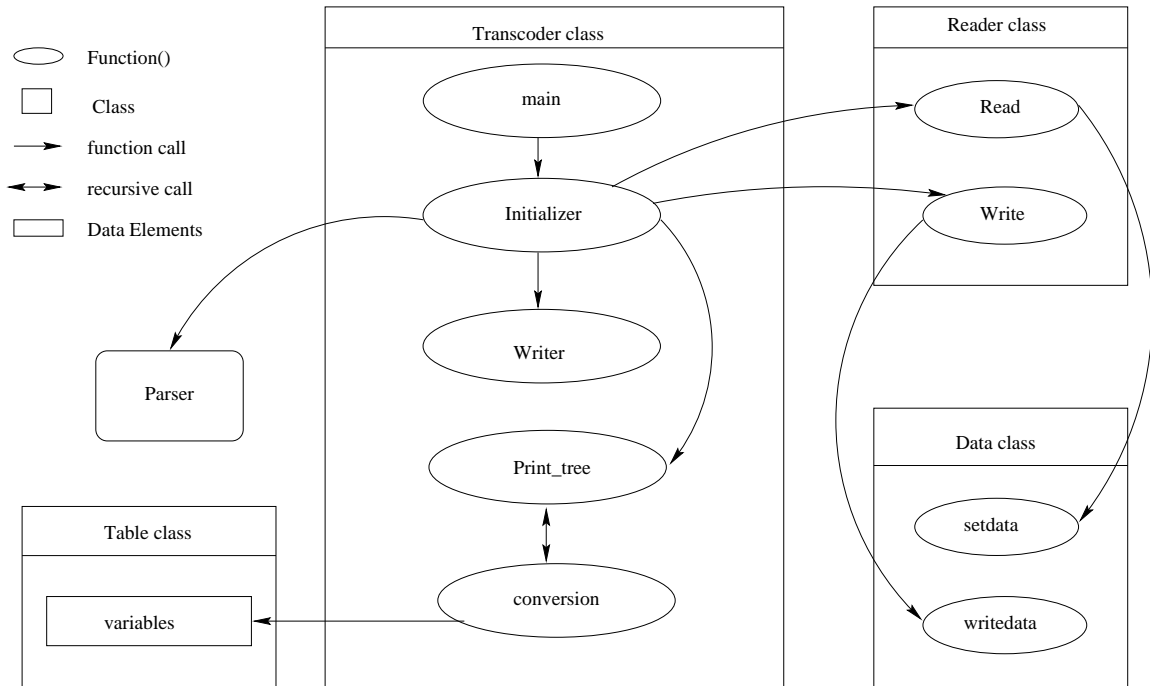


Figure 4.9: High Level Implementation Diagram

such a grammar would be more complex. Thus, check-boxes are not supported in VoiceXML.

3. Some HTML files may contain scripts embedded within them. However, scripts (Javascript and Jscript) are not supported by our system.

CHAPTER 5

CONCLUSION AND FUTURE WORK

An application that converts specific information from one file format to another is called a transcoder. In this thesis we have proposed and implemented one such transcoder that is capable of converting HTML web-pages to corresponding VoiceXML web-pages. The transcoder accomplishes the task by organizing the work in two phases. First phase is the HTML parsing phase, which takes in the input HTML file and produces an HTML node object which is organized in a tree structure. The parser that is utilized in the thesis is the one provide by the Webwise System, Inc. This Webwise free parser is capable of parsing any HTML 4.0 file. The second phase is the most important phase where in the HTML node tree is converted to a well formed VoiceXML file. In this phase, the HTML node tree is traversed in a left to right recursive manner and transcoding logic is applied to each and every node to convert it to a well formed VoiceXML node.

The distinct feature of the transcoder is that it is extensible. As the translation logic is designed with the help of internal data sheet and external user interface sheet, new HTML nodes (in future) can be added to any of the data sheets, and the corresponding logic for that node can be given by the user. But the HTML node included should not be a complex node which may require changes in the transcoding logic of other nodes. A trivial text display tag can be added easily to the external user interface sheet. For example in future if a tag “<scatter>” (e.g.,<scatter> text <scatter>) is added to the HTML specification, which displays the enclosed text in a scattered manner, then this tag can be included in our translation logic by adding it to an external interface sheet.

5.1 Applications

The main aim of developing such a transcoder is to make it function in web-servers, so that the content that is now available only to the desktop browsers can be made available to the mobile clients also. The following applications are listed based on the fact that the developed transcoder will function as an integral part of a web-server:

1. Internet access which is now available to normal human beings with proper sight can be extended to visually impaired people. By utilizing our application, Internet can be accessed via telephones. Thus, the reasonable cost involved in buying a PC and setting up an Internet connection can be wiped out.
2. If the mobile clients were to be supported without utilizing an application like ours, then the web-content has to be stored in multiple formats. This would consume huge memory space and would waste resources. But, if our application resides on the web-server then the web-content need not be maintained in different formats, instead the transcoder would convert HTML pages on the fly and the request of the mobile client would be satisfied.
3. The technology would be of immense use to drivers and workers as they could access the Internet by indulging in a hands free mode of browsing. They can access certain Internet sites which supports voice browsing by dialing a '1-800' number.
4. Above all , the HTML to VoiceXML transcoder paves the way for *anytime Internet access* for mobile users irrespective of location.

5.2 Future Work

1. Our system works on the assumption that input HTML files are always well formed. It means that, every tag that is opened must also be closed promptly. Most of the desktop browsers do not implement this restriction . Thus, even if a HTML page is not well formed, they are processed correctly by the browsers. Hence users tend to create web pages that are not well formed. In future, our system has to be modified

such that it is able to process non-well formed HTML pages also. The modification can be done by creating a software module which inserts missing HTML tags as per the syntax.

2. Our system does not support applets and Java scripts section at this stage. The applet section of the HTML pages are ignored. In future, the system has to be upgraded to a level where it can process these sections
3. Naturally a voice browser cannot match the traditional desktop browser in developing rich multimedia presentation. Thus, even though we claim that user interaction in audio browsers is as good as in desktop browser, there are still some grey areas which can always be improved. For instance 'form' is the most common way by which user interaction is carried out in HTML. Even if a HTML form contains more than one hundred inputs, all of them would be shown in the same page thus providing the overall status of the form to the user persistently. In VoiceXML, each of the field would be read out sequentially, and the next field will be processed only when the previous field's variable is assigned a value. Thus, there is a possibility that a user might lose track of his/her answers during the conversation. In the future, the system can be made to re-prompt the entries made by the user at regular intervals.

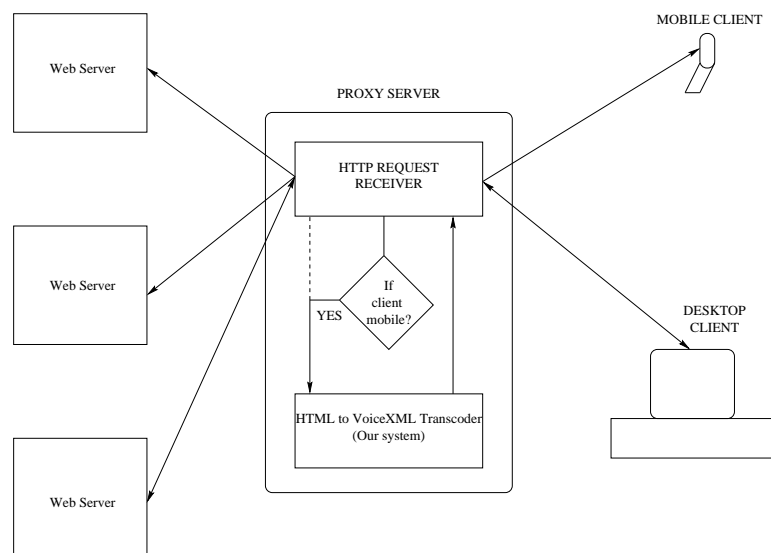


Figure 5.1: Future Work - Proxy Server

4. Our system can best fit into a proxy server. Our transcoder can be made to function as a separate block in a proxy server. When the proxy server receives request from a mobile client requesting for a VoiceXML page, the proxy server can invoke our transcoder to accomplish the task of converting the HTML file present in the web server to VoiceXML file that can be supplied to the client

Instead of making the transcoder reside in the individual web-servers, they can be made to reside in the proxy servers. This modification could cut down the number of transcoders being employed.

BIBLIOGRAPHY

- [1] Peter J. Danielsen, "The promise of a Voice-Enabled Web", *IEEE Computer*, vol. 33, issue. 8, Aug 2000.
- [2] Bruce Lucas, "VoiceXML For Web-based Distributed Conversational Applications", *Communications of the ACM*, vol. 43, no. 9, pp. 53-57, Sep. 2000.
- [3] W. Keith Edwards and Elizabeth D. Mynatt, "An architecture for transforming graphical interfaces", In *proceedings of the ACM: USIT, 1994*, pp. 39-47. ACM Press.
- [4] Janet D. Hartman and Joaquin A. Vila, "VoiceXML Builder: A Workbench for Investigating Voice-based Applications", *Frontiers in Education Conference, 2001. 31st Annual*, vol. 3, pp. S2C - 6-9, Oct. 2001.
- [5] Albers M and Bergman M, "The Audible Web: Auditory Enhancements for WWW Browsers", *Proceedings of the 3rd International World Wide Web Conference*, April 1995.
- [6] G. Gupta, O. El Khatib, M. F. Noamany, H. Guo, "Building the Tower of Babel: Converting XML Documents to VoiceXML for Accessibility", *Proceedings of the 7th International Conference on Computers helping people with special needs*, pp. 267 - 272, OCG Press (Austria).
- [7] Juliana Freire, Bharat Kumar and Daniel Lieuwen, "WebViews: Accessing Personalized Web Content and Services", *The Tenth International World Wide Web conference on World Wide Web*, pp. 576-586, May 2001.
- [8] Goose S, Newman M, Schmidt C and Hue L, "Enhancing web accessibility via the Vox Portal and a web-hosted dynamic HTML to VoxML converter", *WWW9/Computer Networks*, 33(1-6):583-592, 2000.

- [9] Home page for the World Wide Web Consortium for Voice applications.
<http://www.w3.org/Voice>
- [10] IBM WebSphere Transcoding Publisher: HTML-to-VoiceXML Transcoder.
<http://www7b.boulder.ibm.com/wsdd/library/techarticles/0201-hopson/0201-hopson.html>
- [11] VoiceXML Forum, 'VoiceXML: The Voice eXtensible Mark-up-Language',
<http://www.voicexml.org>.
- [12] Mynatt, E, "Auditory Presentation of Graphical User Interfaces", in *Auditory Interfaces*, G. Kramer Ed., 1994 pp. 533-555.
- [13] Hakkinen M, "Issues in Non-Visual Web Browser Design: pwWebSpeak", *Proceedings of the 6th International World Wide Web Conference*, pp. 287-288, June 2001.
- [14] Home page for conversay systems: provider of Voice-driven solutions,
<http://www.conversay.com/Solutions/WebApps.asp>
- [15] James F, "Presenting HTML Structure in Audio: User Satisfaction with Audio Hypertext", *Proceedings of the International Conference on Auditory Display(ICAD)*, pp. 97-103, November 1997.
- [16] Wynblatt M and Benson D, "Web Page Caricatures: Multimedia Summaries for WWW Documents", *Proceedings of the IEEE International Conference on Multimedia and Computing Systems*, November 1997.
- [17] VoiceXML compiler: <http://www.tellme.com>
- [18] T. V. Raman, "Emacspeak-direct speech access", *ASSETS '96: Second Annual ACM Conference on Assistive Technologies*, pp. 32-36, New York, April 1996. ACM SIG-CAPH, ACM.
- [19] Wynblatt M, Benson D and Hsu A, "Browsing the World Wide Web in a Non-Visual Environment", *Proceedings of the International Conference on Auditory Display(ICAD)*, pp. 135-138, November 1997.

- [20] Wynblatt and Goose S, "Towards Improving Audio Web Browsing", WWW Consortium(W3C) Workshop on Voice Browsers, Boston, USA, October, 1998
- [21] 'HTML Template Parser Productivity', <http://www.wisesystems.com>
- [22] A complete list of VoiceXML tags and their uses, <http://www.voicexmlreview.org>
- [23] 'A Compact list of HTML Tags', <http://www.willcam.com/cmat/html>

VITA

Narayanan Annamalai was born in Madurai, India, on April 30, 1979, the son of N Annamalai and Meenakshi Annamalai. He finished his high school at Vikaasa Matriculation school, Madurai, India in 1996. In the same year he was admitted to the bachelors programs in Computer Science Department, Annamalai University, Annamalai Nagar, India. He was awarded the Bachelor of Engineering in Computer Science and Engineering in June, 2000. In August 2000, he was admitted to the Masters program at The University of Texas at Dallas, Department of Computer Science. He is right now a Teaching Assistant in the Computer Science Department and in his final semester in UTD.