Advanced Vocal Web Browser

Michael Sazbon

School of Engineering
Computer Science & Networks Dept.
Jerusalem College of Technology (JCT)
michael_sazbon@yahoo.fr

Yoram Haddad

School of Engineering
Computer Science & Networks Dept.
Jerusalem College of Technology (JCT)
haddad@jct.ac.il

Abstract— Nowadays, more than 2 billion people around the world have access to the Internet regularly, and the Internet is the most important platform for information, work and entertainment with more than 150 million active websites. However, these websites are accessible only through devices equipped with a screen and a Graphical User Interface (GUI). Furthermore, these devices need a network connection to access the internet. With the development of a Vocal User Interface (VUI), Artificial Intelligence (AI) and VoIP communication, we present in this article a system which allows browsing the Internet by using a standard phone only. The system works like an intelligent call routing mechanism that accepts vocal commands as input from the user, translates those commands into HTTP requests, sends them to the web server which processes it and finally returns the HTTP response translated back to the user in a vocally manner. To reach this goal the system implements Content Extraction (CE) algorithms over web content in order to analyze, classify and return relevant parts of web pages to the user. We conducted a series of experiments to evaluate the performance of the system. Our results show that the system offers a reliable and efficient web browsing experience in more than 80% of the websites tested.

Its applications are numerous: for example helping the blind access the internet through speech and hearing, helping disabled people or young children unable to use a keyboard to "speak" their commands into the web, or, simply enabling any person to interface the web contents via oral commands, instead of a keyboard.

I. Introduction

The Web has become a critical source of information and is used more and more in daily activities. The primary mode of interaction with the Web is via graphical browsers, which are designed for visual interaction. Vocal access to the network is provided by screen-readers software which process web pages sequentially and read through the page contents. Web browsing can become tiring and arduous, time-consuming and finally inefficient.

In 2000, the The World Wide Web Consortium (W3C) released a new markup language, VoiceXML. According to the W3C, VoiceXML (Figure 1) is designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and dual-tone, multi-frequency DTMF key input, recording of spoken input, telephony, and mixed initiative conversations[1]. Its major goal is to bring the advantages of Web-based development and content delivery to interactive voice response applications. Similar to the HTML language, it provides a vocal equivalent of HTML pages and allows interaction through VoiceXML form, dialog, links and menus between the user and web applications hosted on a web server. Moreover, this technology offers access to the internet network through the Public Switched Telephony Network (PSTN) and does not require an additional network connection.

Our system is based on the actual VoiceXML platform to benefit from the support of Text To Speech (TTS), Automatic Speech Recognition (ASR) and VoIP communication. It adds some important functionality to this platform, from one side, a vocal transcriptor of websites' content by using Content Extraction algorithms, and from the other side, a vocal command interpreter using Natural Language Processing (NLP), and Machine Learning algorithms (MLA).

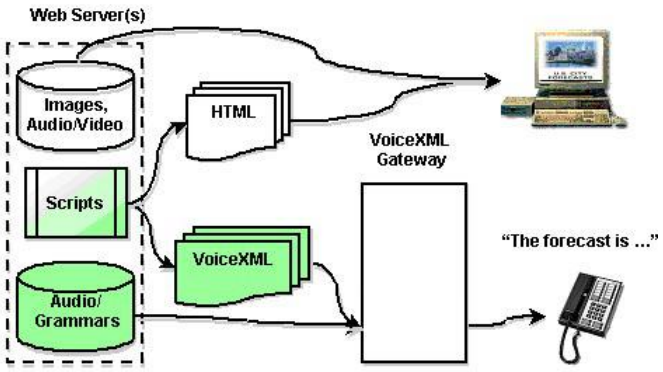

Figure 1: VoiceXML Architecture

II. System Architecture

The system works in a Client/Server Architecture, meaning that all the application resides on the web server and the client uses the vocal interface to interact with it. The system is composed of the following modules (Figure 2): VoiceXML client, main controller, VoiceXML template builder, HTTP client, Content extractor, database server and the virtual browser.

The VoiceXML client is usually a regular phone (land phone or cell phone) which is able to send and receive speech data through the Telephone network (PSTN).

The main controller is the central processing unit of the system. Its task is to receive HTTP requests from the client, to

978-1-4244-8682-3/10/$26.00 ©2010 IEEE

dispatch and route all the processing tasks to the appropriate modules. Once each of the modules terminates its task, the controller encapsulates the response in a HTTP response and returns it to the client.

The VoiceXML template builder is a programming framework which is responsible of dynamically formatting the vocal interface using the VoiceXML markup. This module uses NLP and MLA in order to become more and more accurate over the time.

The HTTP client is regular HTTP client software. Once an HTTP request is received from the client, its task is to process it by visiting the URL requested, collecting the HTML page, parsing the web content and return it to the main controller.

The content extractor is the core element of the system. Based on efficient CE algorithms and powerful heuristics, its role is to parse, analyze and extract relevant parts of the HTML pages retrieved by the HTTP client. It builds a map of all the different elements contained in the web page (menu, sub-menu, categories, items, forms, banners…) and transforms them into logical objects (trees, array…) for processing.

The database server is used to hold the information collected from the websites during the overall process.

The virtual browser is the entity which simulates a web browser to the end user. It provides the basic functionalities of a web browser like browsing, refreshing, forward and backward navigation, history and bookmarks. It gives the user a web browsing experience equivalent to the traditional visual browsing on a computer.

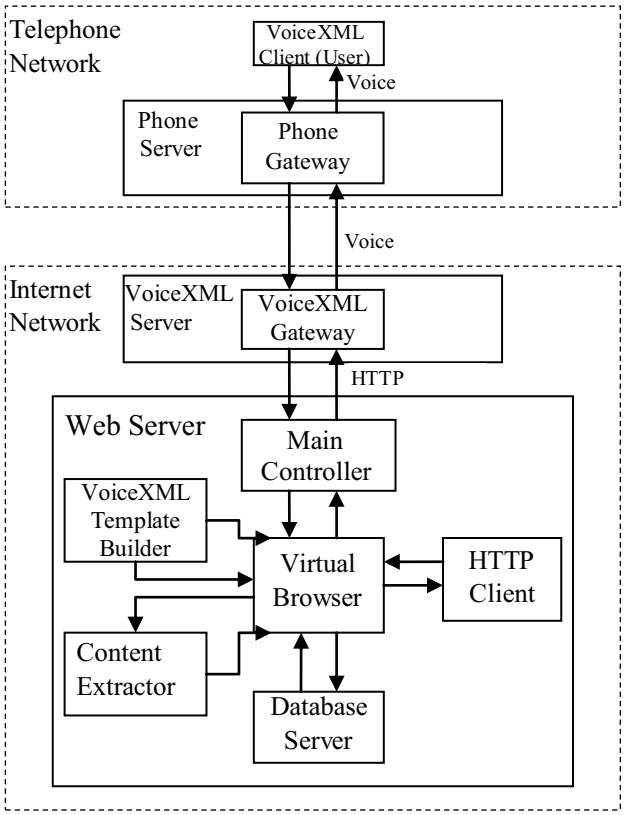

Figure 2 : System Architecture

III. System Functionality

Websites can be divided into categories like News websites, or Entertainment, Social, Sports, etc.

Each category has a common structure, for example news websites will always have some traditional categories like: World, Sport, Weather, Finance….

A web page can be broken down into logical parts like main menu, sub menus, main content and secondary contents. Some parts of the page, like navigation modules, are common to all the pages, others, like articles, are specific to a single page.

As already mentioned, the core functionality of the system is to parse, analyze and extract these contents. In order to reach this goal and after a long research, we decided to use a Document Object Model (DOM) approach. The DOM defines a standard for accessing documents like HTML and XML. It provides an Application Programming Interface (API) which defines the objects and properties of all document elements, and the methods (interface) to access them. The DOM is a tree representation of a HTML document; it gives information about its visual output, not on its semantic meaning. That is to say, HTML code is a markup language designed to structure and format content for visual representation, it doesn't include information about the type of data it contains or how it is encapsulated inside the page. If in a visual environment, this data is clearly identifiable , however through the DOM there is no simple way to recognize those elements inside the pages. For instance, on a News website via a visual browser, it will be easy to recognize the main menu generally positioned on the top of the page and the headlines displayed on the center of the page.

Moreover, HTML documents, as a W3C standard, have a strict hierarchical structure. However, in this standard there are several ways to present information on the web page. There is not a unique structure or template to format information inside a web page, making it difficult to scan the page and extract relevant parts. For example, a possible HTML representation of the main menu of a website could be formatted inside a list of elements (Figure 3) with the <ul> (unordered list) and <li> (list item) elements. But it could be also displayed inside a table (Figure 4) using the <table> element or even without any additional markup as a simple list of link elements <a>.

Brute force method consisting to try all the possible templates could be used but is time consuming and not appropriate. Many heavy CE Algorithms exist but we wanted to use heuristic algorithms which give a faster response time.

IV. Content Extraction Methodology

One of The main CE algorithms we developed uses the XPath language. XPath is a query language for selecting nodes from an XML or HTML document. It is based on the DOM, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria. To keep a homogenous visual aspect, a website needs to maintain its global HTML structure over all the pages, only the data inside the page can change. When we look carefully at the structure of an HTML document, we can see repetitive patterns used to format specific kinds of information. For example, in Figures 3 and 4

```
<div id="nav">   // menu
  <ul>
    <li>
      <a href="news.php">News</a>
    </li>  // menu item 1
    <li>
      <a href="weather.php">Weather</a>
    </li>  // menu item 2
    <li>
      <a href="sport.php">Sport</a>
    </li>  // menu item 3
  ……..
  ……..
  </ul>
</div>
```

Figure 3 : Unordered list representation of a menu

```
<table id="nav"> //menu
  <tr>
    <td>
      <a href="news.php">News</a>
    </td>    //menu item 1
    <td>
      <a href="weather.php">Weather</a>
    </td>  //menu item 2
    <td>
      <a href="sport.php">Sport</a>
    </td>  //menu item 3
  ……..
  ……..
  </tr>
</table>
```

Figure 4 : Table representation of a menu

we can see that the menu doesn't have a unique structure common to all websites, but in a specific website the structure is the same on all the pages. Logically a menu could be represented as a list of link items displayed in an iterative manner. For example, as we can see in the Figure 3 each menu item is wrapped inside a <li> tag called an **iterator**. All the menu items are inside a <ul> tag called a **container,** itself inside a tag and so on until we reach the <body> tag defining the root element of the page. Supposing this div element is inside the body element, it gives us the following XPath expression:

body[0]/div[0]/ul[0]/li[i]/a[0]

With **i** the index of the menu item in the menu. This expression identifies each menu item uniquely because the DOM is a tree, not a graph. By removing the [i] of this XPath expression, we get :

body[0]/div[0]/ul[0]/li/a[0]

This expression selects all the link elements <a> inside an **iterator** contained inside a **container** which is exactly the menu items we were looking for.

With this method, once a menu item is found in the page (by a keyword search on common category name like news, sport, or finance for a News website for example) we could easily compute a more general XPath expression to retrieve all the other menu items. A possible implementation for this algorithm can also be seen in the Algorithm1.

V. Vocal User Interface

A vocal interface differs from a visual interface in that spoken prompts are transient and don't have any consistency. Furthermore, speech recognition is not 100% accurate. As a result, voice interfaces are much more dialog oriented, with emphasis on verbal presentation and response. Visual interface can present to the user more information at once, whereas a vocal interface can read to the user only few amounts of data at a time. Therefore we had to design an efficient vocal interface pattern. Reading to the user the all webpage from top to bottom, like a screen reader, is not a good interaction model. Once all logical parts of a webpage are extracted we have to present them to the user in an orderly manner. The first time a user visits a website, the system analyzes all the pages, extracts the relevant contents (main menu, secondary menus, main content…) and gives the user a brief overview of its content.

Then the system reads a navigation menu which allows the user to interact with the website like a visual standard interaction .This menu is composed of all the functionalities provided by this website (static navigation pages, search form, articles…). From this point on, the system reacts like an event driven application, once a user makes a request the system responds by executing the desired action on the website.

A MLA mechanism has also been implemented in order to learn by itself with the collaboration of the user which elements are relevant to the user and how to recognize them inside the webpage. After each response the system asks the user if the data retrieved was accurate or not.

1- Find a menu item M[x] in the page
2- Compute the XPath expression $XP_{M[x]}$ for M[x]
3- Compute the general XPath expression XP_M for all the M[i] menu items
4- Use XP_M to find all the menu items

Algorithm 1 : A possible implementation for the XPath extraction algorithm (XEA)

VI. Evaluation And Experimentation

For our tests we built an experimental implementation of the system. We developed it using the PHP programming language running on an APACHE web server on the UNIX platform. The system was able to connect to a VoiceXML services provider through the network and accessible by a Skype™ phone number. For simplicity we conducted the tests only on websites adapted to mobile phones. The difference between mobile and regular websites is that mobile websites have a flatter HTML structure, which makes the separation between the different logical parts of a web page more visible. The visual display is simplified but the functionality remains unchanged. For the tests we ran our CE algorithms and especially the XEA on different News websites for mobiles, and tried to extract relevant parts of the page like the main

menu, item list (headlines), main content (a news article) and forms (search or login form). The results indicated the percentage of content the algorithm managed to extract, a result of 100% means that the algorithm managed to extract the totality of the desired content. As we can see in the Figure 5 we obtained an average success rate of **81.67%**, showing the high effectiveness of our algorithm.

Website	Main Menu	Item List	Main Content	Forms	Average
BBC NEWS	100	50	100	50	75
Boston.com Mobile	30	100	100	50	70
CBC News (Canada)	50	50	80	80	65
Cincinnati.com	50	0	100	80	57.5
CNN Mobile	100	100	50	80	82.5
Dallas Morning News	100	100	0	0	50
Denver Post	100	100	100	100	100
Houston Chronicle	100	100	100	100	100
KOMO-TV Seattle	100	100	100	100	100
Mobile.Newsvine	80	100	100	100	95
NBC 5 West Palm Beach	100	100	100	-	100
The Hill Mobile	100	100	100	-	100
USA Today	50	50	100	-	66.7
				Total	**81.67**

Figure 5 : Results of CE Algorithms test in percent

VII. CONCLUSION

In this paper we proposed a system able to translate visual websites into vocal applications giving the user a realistic browsing experience. Our system is based on the collaboration of distributed modules communicating over the network. A key feature is that it doesn't require any additional components to work like a TTS engine or a speech recognition engine since it is built on the actual VoiceXML platform. We developed our CE algorithm, the XEA, which is based on the XPath language a W3C standard, and tested it with an experimental implementation. Furthermore, this system is more accessible to everyone, given that it doesn't need any Internet network connection but only a regular phone. Thus, it represents a significant progress in the world of Internet access and will generate new vocal applications based on richer vocal interfaces.

ACKNOWLEDGMENT

The authors would like to thank Dr. Aryeh Teitelbaum from the Jerusalem College of Technology for his valuable input and Dr. Yossi Peretz, for his supervision of part of this research. The help of Hanan Magouri is also gratefully acknowledged.

REFERENCES

[1] W3C Recommendation 16 March 2004. "Voice Extensible Markup Language (VoiceXML) Version 2.0"

[2] A. F. R. Rahman, H. Alam and R. Hartono. “Content Extraction from HTML Documents”. In 1st Int. Workshop on Web Document Analysis (WDA2001), 2001.

[3] Suhit Gupta, Gail E. Kaiser, Peter Grimm, Michael F. Chiang, Justin Starren. Columbia University, Automating Content Extraction of HTML Documents

[4] Broadbent D.E. (1958). Perception and communication. London: Pergamon press.