I.J. Information Engineering and Electronic Business, 2017, 4, 13-20
Published Online July 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2017.04.03

IReadWeb: Towards Best Performance of WebAnyWhere

Najwa K. Bakhsh
Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
Email: Najwa.bakhsh@gmail.com

Saleh Alshomrani
Department of Information Systems Faculty of Computing and Information Technology, University of Jeddah, Jeddah, Saudi Arabia
Email: sshomrani@uj.edu.sa

Imtiaz Hussain Khan
Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
Email: ihkhan@kau.edu.sa

***Abstract*—**This article describes IReadWeb system, which is based on existing WebAnyWhere technology. The existing WebAnyWhere system uses depth-first search (DFS) to traverse the Document Object Model (DOM) during the Web surfing task. DFS uses an exhaustive search and crawls through an entire page until it identifies the target node thereby greatly increasing the response time to users. We developed a user-experienced based algorithm, which, unlike DFS, exploits pre-fetched information stored in a local cache to speed up the browsing task. The performance of IReadWeb is thoroughly evaluated and compared against WebAnyWhere by using a sizeable sample of blind native Arabic speakers. The experimental results show that IReadWeb outperformed WebAnyWhere in attaining fast response speed.

***Index Terms*—**Web application, Disability, Text-to-speech, IReadWeb.

I. Introduction

We are interested in building a WebAnywhere-based system that is available in the Arabic language and can help visually impaired Arabic speakers access the Web. WebAnywhere is a Web-based, self-voicing browser that enables blind users to access the Internet from almost any computer that can produce sound [1]. The performance of a WebAnywhere system depends (amongst other things) on the quality of a screen reader, which uses a text-to-speech (or speech synthesiser) system in reading or parsing a webpage. WebAnywhere uses depth-first search (DFS), a method of parsing the Document Object Model (DOM), in which an algorithm crawls through a page until it identifies a target node. Starting from the root node, the algorithm crawls down to the next element and repeats the process until the element of interest is found or the search space is exhausted. The speed with which an algorithm operates depends on two factors: the number of calculations it must make and the time required to access the resources necessary for the algorithm to run (usually, memory). A DFS must remember the element that it previously visited; thus, the minimum amount of memory it requires is equal to the number of elements in a tree. On the other hand, DFS has an exponential time complexity which makes it very inefficient [2]. This problem motivated us to enhance the prefetching function in WebAnywhere and improve the system's performance as highlighted in [9,10].

In the present study, we built an intelligent Arabic WebAnywhere plug-in that is intended to assist Arabic users in accessing educational websites and checking their email. The plug-in also enables reading and shopping from any location at any time. To enhance the system, we examined user experience with WebAnywhere. In the evaluation, we configured the system so that it begins reading the most frequently used element, instead of reading all the unwanted elements on a webpage. This way, the system can identify a target element in a short period. This feature enables the creation of an intelligent, highly accessible system.

In a previous study [3], we conducted three investigations to evaluate the performance of NonVisual Desktop Access (NVDA) and IBSAR, which are two speech synthesiser systems used by blind people [11,12]. The first investigation focused on the quality of the systems in terms of pronunciation, the second revolved around the intelligibility of the systems and the third investigate the meaning of sentences means the degree of received messages being understood. This article describes IReadWeb system which is based on existing WebAnyWhere technology. In building the system, we developed a new algorithm called the User Experience algorithm to enhance page reading. The algorithm uses

DFS and saves all reading-related behaviours, skips and bookmarking of favourites in local storage. This method also reduces the time consumed in reading a page. In the present study, the performance of IReadWeb is thoroughly evaluated and compared against WebAnyWhere by using blind native Arabic speakers. The experimental results show that IReadWeb outperformed WebAnyWhere in attaining fast response speed.

The rest of this article is organized as follows. Section II discusses the related work. The WebAnyWhere system is described in Section III. In Section IV, the proposed IReadWeb is described. The evaluation of the proposed system is given in Section V. Section VI discusses the results. The paper concludes in Section VII.

II. Related Work

In [4], the authors overview existing solutions for mobile web access for blind users and presents the design of the WebAnywhere system. WebAnywhere generates speech remotely and uses prefetching strategies designed to reduce perceived latency. A user evaluation of the system is presented showing that blind users can use Web-Anywhere to complete tasks representative of what users might want to complete on computers that are not their own. A survey of public computer terminals shows that WebAnywhere can run on most.

In another study [7], the authors describe the performance and security implications of the system's unique design and how it has been engineered to provide usable access anywhere. Specifically, we present prefetching and caching strategies developed to make the system responsive even on low-bandwidth connections and security considerations that replicate existing browser security policies.

In [5], the authors describe recent developments and their experiences in releasing WebAnywhere. WebAnywhere was originally designed as a web-based alternative to a traditional screen reader. It can be run on any computer without installing new software, making it ideal for use on-the-go or in libraries and schools where the appropriate access technology is unlikely to already be installed and where users do not have permission to install it. Since its initial release nearly two years ago, WebAnywhere has expanded from its original goal of supporting blind web users to become a platform for an array of technologies supporting access for people with disabilities.

In yet another study [1], the authors expanded the WebAnywhere from its main goal to become a platform for an array of technologies supporting access for people with disabilities.

In [6], the authors overview the browsing strategies that the research have observed screen-reader which users utilize when they are faced with challenges, ranging from unfamiliar Web sites and complex Web pages to dynamic and automatically-refreshing content. The author develops a new tool that makes experienced users more effective, and assist them to overcome the initial learning curve for users who have not yet acquired effective browsing strategies.

Webanywhere is designed to function on most computers that have Internet access and that can play sound. Webanywhere does not support users' behavior or browsing history also Webanywhere do not support Arabic language.

In this research, we built an intelligent WebAnyWhere to be available in Arabic languages with best text to speech by using Text to speech evaluation. In addition, enhance the performance of the WebAnyWhere[15] as well as improving the perfection time by forcing the system start reading the most frequently and favorite elements used content in stated of reading all unwanted content.

III. The Existing WebAnywhere System

The WebAnywhere [14, 15] works as follows. After a user requests a page from the mainframe domain, he/she can then browse the page locally. The client process of JavaScript parses the subDOM document and re-sends it as text to the Web to be read. The client then interactively accepts the voices that represent the transmitted text and dynamically plays it for the user. The user can interact with the system through functions such as "voice and skip," "save as favourite" and "listen". Information on user interaction should be saved locally so that the same data can be re-loaded onto a page and the system can control how the same page will be read for a user. User interaction should be continuous to enable our main algorithm to determine what components to save and in what manner in the local storage.

The three main processes in the WebAnywhere architecture[1,5] are the domain server process, the client browsing process and text-to-voice conversion in the server. The following diagram (Fig. 1) shows the overall data exchange that occurs locally in a user's client PC or between Internet servers.

A. Domain Server Process

In the domain server process, a web server that hosts a domain on the Internet accepts requests from a client and loads the corresponding webpages in the main frame of WebAnywhere. An intermediate box loads pages within the subframe for browsing in the client where the user sends requests.

B. Client Browsing Process

The client browsing process is a program that exists in a user PC. Its operation primarily involves the use of a JavaScript processor and an HTML converter that loads pages for browsing over Internet browsers, such as Internet Explorer 10 and Firefox. As described in the preceding chapters, WebAnywhere uses DFS to search through a tree. The time spent traversing the DOM via DFS depends on the number of elements on a page and the duration with which the page loads. The worst-case scenario would be that the algorithm will have to pore through every element and every page. Assuming that

each element on a page has the same number of elements—a constant branching factor— DFS incurs in exponentail time complexity. Of course, this scenario affects the duration of prefetching. To solve this problem, we developed a new algorithm called the User Experience algorithm and incorporated it into the plug-in (details follow).

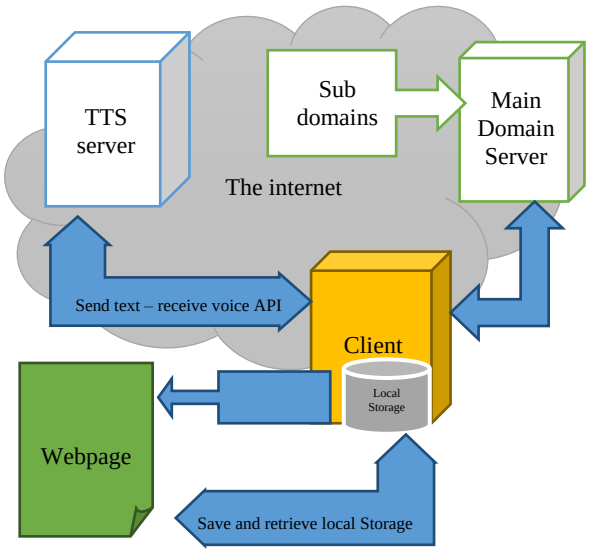

Fig.1. Text to Speech System Architecure

C. Text-to-Voice Conversion

In text-to-voice conversion, a server accepts texts from clients and responds by transmitting a voice that represents the texts. In this stage, a sophisticated speech generation system is required to present the texts, but the differences in strengths and weaknesses amongst various speech production systems impedes this process. To solve this problem, we evaluated existing TTS synthesisers to choose the best application (details follow). Another problem that we encountered is the unavailability of such applications in the Arabic language. Specifically, no application offers free reading in Arabic for communication with automated requests from clients.

D. IRead Web

Our goal in building the plug-in tool is to help blind and visually impaired individuals browse the Web without the need to download new software. We formulated the following criteria for development:

- WebAnywhere requires a sophisticated speech generation system for presented text, but the problem is that different speech production systems are characterised by various strengths and weaknesses. Therefore, the first implementation goal is to determine what components constitute a good speech production system. To achieve this goal, we experimentally studied text-t-speeach (TTS) synthesisers to choose the best amongst existing technologies.
- Prefetching in WebAnywhere should be enhanced for better performance [4,5]. This criterion is motivated by the manner of parsing in WebAnywhere. As stated above, WebAnywhere uses DFS to parse the DOM. DFS lacks in both space and time efficiency thereby compromizing the overall performance of WebAnywhere.

We enhanced the performance of DFS by taking user experience into account and incorporated it into the plug-in, henceforth IReadWeb. The user experience script is added to the client browsing process. It uses browsing history, which comprises data on the interesting interactive behaviours of a user on a given webpage. The navigation algorithm enables the client to save this data in a PC as cookies or similar elements.

The client sends two main requests. It first sends the requested page domain to the domain server and sends text to the text-to-speech server through an API. The server then plays the transmitted voice.

To save information regarding user behaviour, favourites and unimportant elements accessed on a webpage, the navigation algorithm first evaluates these elements and subsequently saves them as cookies. A problem that arises is that data on browsing history are larger than the available cookies offered by a browser. A solution is the adoption of several techniques in which external files store all the data in a PC.

The first issue that confronts us is that the JavaScript running in a browser is unauthorized to handle a computer file API. Thus far, the only solution to this problem is the representation of files in Chrome via a file system API. We expect this technique to be available in all browsers in the future because it is officially supported by the W3 organisation. Until then, however, this solution

remains impractical. The alternative that we recommend is to opt for local storage that is supported by HTML5 [8]. This type of storage presents many advantages over ordinary cookies. It enables long-term storage, unlike HTTP cookies, which expire by default. To create an application that allows long-term storage, developers must use complex syntax and constantly update expiration dates. Another advantage of HTML5-supported storage is that it does not involve the use of a plug-in.

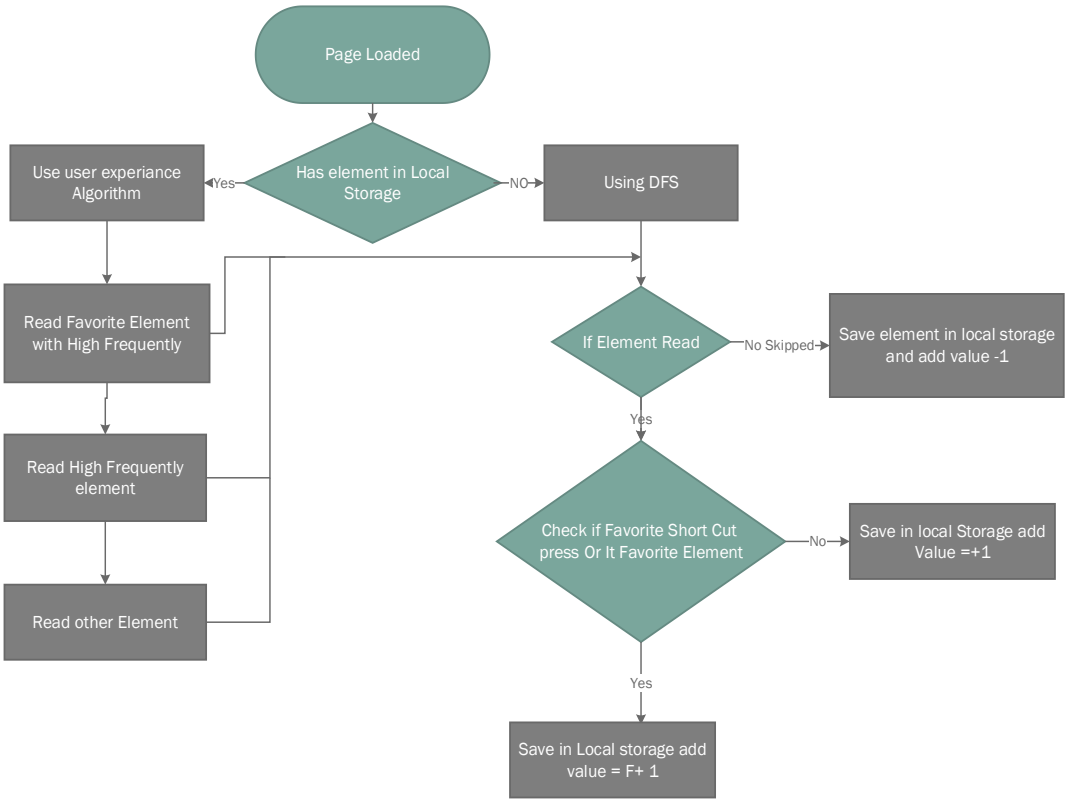

Fig.2. IreadWeb Flow Chart

The Fig. 2 comprehensively illustrates user experience. After a page is loaded by the client, script two scenario is operated first; that is, the system checks whether this is the first time an element on a page is read, after which DFS is executed as follows:

- If an element is read, 1 is added to the element value. The element ID is saved and the value is sent to the local storage.
- If the element is read and the favourites shortcut is pressed (Ctrl + M), then the element value is saved as (f1) in the local storage.
- If the element is skipped, then 1 is subtracted from the element value. The remaining value is saved to the local storage.

The second problem is that if the page has been previously loaded, then the User Experience algorithm functions as follows:

- All favourite elements with high values are read. If an element is read, 1 is added to the element and previous values (value = fv + 1, where f is the favourite and v denotes the last value added).
- If a user skips one of his/her favourite elements, then 1 is subtracted from the element value (Value = fv – 1, where f denotes the favourite and v represents the remaining value added to the element).
- If the element being read is not a favourite, then 1 is added to the element value (Value = value + 1).
- If an element is not a favourite and it is read, 1 is subtracted from the element value (Value = value – 1).
- The algorithm checks on whether an element has been read and saves this situation in the local storage as the first scenario. This step facilitates page updates.

The second issue that requires addressing is the manner by which user experience data are stored. In some cases, the DOM element that a user needs to save or discard an element—using the main algorithm—will not have an ID; it therefore lacks the unique characteristics that enable automated saving by a programme. This situation, in turn, generates bugs that may impede user experience. For example, a user hears elements that he/she did not access, or a programme discards data or assigns low priority to data that are considered important by the user.

Internet developers cannot create an ID for every element in the pages that they design. Furthermore, certain data are already generated on a page even before

loading, making this page self-generating in terms of process, function, search and database access. The text node in any DOM contains no distinguishing features; it is an actual text and no tag represents the text node itself. A text node is always a leaf node.

E. Interface

In WebAnyWhere [5], we designed the interface so that it is divided into two frames: One frame works as a browser, and one frame is for downloading the website. The browser frame contains the address bar, "إذهب" ("Go"); to make this more usable, we added two more buttons: "التالي" ("Next") and "السابق" ("Previous"), so the user can move forward or backward. Also, by using the Search Text box, the user can look for specific text or elements. The other frame is to download the requested web page after it download the WebAnyWhere (WA) script with the user experience will work.

Fig.3. IreadWeb Interface

IV. Test And Evaluation

A. Different Operating Systems/Browsers

We tested the performance of IReadWeb (the WebAnyWhere after adding the user experience) in different operating systems and browsers to make sure the pages look as intended. The following combinations as shown in Table 1 were tested.

B. Usability Test

In order to test the usability of the website, the following attributes were tested: appearance, clarity, and ease of access. Other factors measured include ease of learning, efficiency of use, and subjective satisfaction. Tasks were chosen to test each of these factors. Test participants were asked to complete typical tasks while observers watched, listened, and took notes regarding whether tasks were completed correctly and the required completion time.

Table 1. Different Operating system and Browser Test

Browser	Operating System	Comment
Internet Explorer 8	Microsoft Windows 7	All functions work as intended
Internet Explorer 7	Microsoft Windows 7	All functions work as intended
Internet Explorer 8	Microsoft Windows 7	All functions work as intended
Safari	iOS 4.2.1	All functions work as intended, but PHP code does not appear.
Mozilla Firefox	Microsoft Windows 7	All functions work as intended
Google Chrome	Microsoft Windows 7	All functions work as intended

After the Arabic language was added to the local server, the program was tested using 12 blind participants from King Abdulaziz University. The experimenter explained the guidelines of use for WebAnywhere, and outlined all the short keys. The test lasted approximately 80 minutes, and was carried out in King Abdulaziz University's specialized experimental laboratory for disabled people. Each participant completed 10 trials, where a trial means surfing a given website; a total of 10 different websites were used: 4 static and 6 dynamic. The following tasks were assigned to test usability factors:

- Open http://www.alarabiya.net/
- Use the arrow key to skip down to Saudi News

- Add Al Saudi News to favorites using CTL+M
- Go to http://www.blindtec.net/vb/forum.php using CTL+L to write to the text field.
- Use the tab key to skip to "Al-Maktaba Al-Saotia المكتبة الصوتية".

C. Performance Test

We evaluated thoroughly the performance of IReadWeb in English, as there is no Arabic version of WebAnywhere. The performance of IReadWeb was compared against the performance of WebAnyWhere using 10 websites, including both static and dynamic websites. Table 2 shows the websites used in the test. The total time elapsed was recorded for each website visited.

WebAnywhere was tested using http://webanywhere.cs.washington.edu/beta/index.php?locale=en and the user experience was uploaded to www.ireadweb.net/wa.

Table 2. The Websites Examined

No.	Websites	Type
1	http://www.co-operativebank.co.uk/	Dynamic
2	http://www.ssa.gov/	Dynamic
3	http://www.cdc.gov/	Dynamic
4	http://www.usa.gov/	Dynamic
5	http://www.canada.ca/en/index.html	Dynamic
6	https://www.couchsurfing.org/	Dynamic
7	http://www.9boninnes.co.za/	Static
8	http://www.blouberg-holiday.co.za/	Static
9	http://www.quayside906.co.za/	Static
10	http://www.fibercom.co.za/	Static

The network speed during testing was 10 mb/s. We assigned target elements, such as X in the website. We chose X as an element at the end of the webpage. Unlike the intelligibility test as reported in [16], we examined the usability of the WbAnyWhere and the IReadWeb. In the WebAnyWhere, to read the website to reach the target element X, the following three different scenarios/conditions were used to examine the websites:

1. Read the page without using any skips to reach the target element (henceforth WS-1, i.e., WebAnyWhere Scenario 1).
2. Read the page using the down arrow skip key (which skips by word) to reach the target element (henceforth WS-2).
3. Read the page using the tab skip key, which skips by element (henceforth WS-3).

Similarly, in the IReadWeb, we added the target element X to the favorites list (Local Storage) in the following three scenarios and measured the performance accordingly:

1. Add the element X to the first elements on the favourites list and give it the highest frequency (henceforth IR-1, i.e., IReadWeb Scenario 1).
2. Add the element X to the last elements on the favourites list and give it the lowest frequency, where the favourites list consists of a maximum number of elements on the webpage (henceforth IR-2).
3. Add the element X to the last elements on the favourites list and give it the lowest frequency, where the favourites list consists of half the number of elements on the webpage (henceforth IR-3).

V. RESULTS

We present the evaluation results measured in terms of reading time taken by a participant against different conditions (as described above). The reading time (speed) was recorded in seconds. The results are shown in Table 3.

Table 3.The Web surfing time of WebAnyWhere and IReadWeb

Website	Web surfing time (seconds)					
	WebAnyWhere Conditions			IReadWeb Conditions		
	WS-1	WS-2	WS-3	IR-1	IR-2	IR-3
1	262.4	182.7	159.3	238	119.3	80.5
2	502.6	251.3	189.7	260.5	139.7	109.3
3	403.7	263.2	241.2	184.2	155.2	130.2
4	342.3	339.8	291.8	115.4	234.8	141.2
5	703.4	341.2	452.3	193.8	232.4	179.5
6	738.6	325.8	320.3	238.2	267.6	190.3
7	315.2	60.8	56.6	34.21	20.34	46.3
8	282.8	151.2	73.2	56.32	43.65	64.2
9	174.6	94	59.8	46.43	34.56	56.8
10	213.4	59.51	48.32	38.76	32.13	40.65

Table 4. The mean Web surfing time of standard WebAnyWhere and IReadWeb

Website	Web surfing time (seconds)					
	WebAnyWhere Conditions			IReadWeb Conditions		
	WS-1 (σ)	WS-2 (σ)	WS-3 (σ)	US-1 (σ)	US-2 (σ)	US-3 (σ)
Static	246.5(64.13)	91.38 (43.52)	59.50 (10.26)	43.93 (9.68)	32.67 (9.60)	55.77 (8.99)
Dynamic	492.17 (194.25)	284.00 (63.03)	275.17 (104.38)	205 (52.5)	191.67 (62.42)	138.33 (41.67)

The mean web surfing time for the static and dynamic websites, averaged over static and dynamic websites separately, is reported in Table 4.

It is clear from the results (Table 3 and Table 4) that overall the IReadWeb offered best performance to the blind people. An ANOVA test was used to test the performance gain against the different conditions (three WebAnyWhere conditions and three user IReadWeb, total six conditions). The ANOVA results revealed that the performance gain differed significantly among these conditions: $F(5,54) = 11.78$, $p < 0.01$.

VI. Discussion

Building on existing WebAnyWhere technology base, we developed IReadWeb which offers rapid web surfing to blind people. To achieve this rapid web surfing goal, the IReadWeb takes user experience into account by exploiting pre-fetched information stored in a local cache. We experimentally showed that IReadWeb outperformed WebAnyWhere in attaining fast response speed. The present study reveals some interesting results which are worth discussing.

- On static websites, both WebAnyWhere and IReadWeb offered competitive performance. The response time of IReadWeb was slightly better than the response time of WebAnyWhere in all conditions though.
- On dynamic websites, IReadWeb outperformed WebAnyWhere in all conditions. This is important because if the user needs a link or an element at the end of the page, (s)he must wait until that link/element is reached. However, in the IReadWeb, a user can add such elements to the favourites list thereby retrieving the required page(s) much quickly.
- Of course the performance of IReadWeb would be affected with an increase in the number of elements on the favourites list. But comparing the number of such elements to the total number of elements skipped/read by WebAnyWhere, it seems plausible that IReadWeb would still process lesser number of elements thereby reducing the response time.
- Many different factors affected the test, including the number of elements on the webpage. These factors add adverse effects especially to WebAnyWhere because it uses depth-first search whose worst-case time complexity is O(V+E) (V as vertical, E as edge). It is important to mention here that this complexity becomes even worse in webpage processing, because the number of pages (P) read to reach the target node also contribute towards this complexity resulting in total worst-case time complexity O(P+V+E).

VII. Conclusion

This study aimed at developing an intelligent plug-in tool for blind users, specifically Arabic native speakers. WebAnywhere was used as the base technology in the development of the intelligent plug-in tool. We enhanced the performance of WebAnyWhere by developing a user-experienced based algorithm which exploits pre-fetched information stored in a local cache to speed up the browsing task. The algorithm was implemented resulting in an improved version of WebAnyWhere: IReadWeb.

The performance of IReadWeb was thoroughly evaluated and compared against WebAnyWhere by using blind native Arabic speakers. The evaluation study was carried out on both static and dynamic websites. The experimental results showed that IReadWeb outperformed WebAnyWhere in attaining fast response speed.

In future, we intend to extend the present work to mobile devices too. Most mobile devices today use browsers with limited functionality that cannot play sound from within the browser, although this is beginning to change. WebAnywhere may become a popular method to enable accessible browsing on mobile devices.

References

[1] J. P. Bigham, C. M. Prince, and R. E. Ladner, “WebAnywhere : A Screen Reader On-the-Go”, 2008.
[2] J. Erickson, “Algorithms”, University of Illinois, 2013.
[3] Najwa K. Bakhsh, Saleh Alshomrani, Imtiaz Khan, “A Comparative Study of Arabic Text-to-Speech Synthesis Systems”, IJIEEB, Vol. 6, No. 4, pp.27-31, 2014.
[4] J. P. Bigham, C. M. Prince, and R. E. Ladner, “Addressing Performance and Security in a Screen Reading Web Application that Enables Accessibility Anywhere”, 2007.
[5] J. P. Bigham, W. Chisholm, and R. E. Ladner, “WebAnywhere - Experiences with a New Delivery Model for Access Technology”, pp. 1–4, 2010.
[6] Y. Borodin, J. P. Bigham, G. Dausch, I. V Ramakrishnan, and S. Brook, “More than Meets the Eye : A Survey of Screen-Reader Browsing Strategies”, 2010.
[7] J. P. Bigham, A. C. Cavender, J. T. Brudvik, J. O. Wobbrock, and R. E. Ladner, “WebinSitu : A

Comparative Analysis of Blind and Sighted Browsing Behavior."
[8] Adobe shockwave and flash players: Adoption statistics. Adobe (June 2013).
[9] J. P. Bigham and R. E. Ladner, "Accessmonkey: A collaborative scripting framework for web users and developers", In Proceedings of the International Cross-Disciplinary Conference on Web Accessibility (W4A '07), pp. 25-34, 2007.
[10] J. P. Bigham, C. M. Prince and R. E. Ladner, "Engineering a self-voicing,web-browsing web application supporting accessibility anywhere", 2008, Available at: http://webinsight.cs.washington.edu/publications/webanywhere-engineering.pdf.
[11] "NVDA screen reader", NV Access Inc., 2007, Available at: http://www.nvda-project.org/.
[12] "Sakhr Software Co. Sakhr Building", Cairo, 2008, Available at: http://www.sakhr.com/.
[13] "Sound Manager 2", Available at http://www.schillmania.com/projects/soundmanager2/, last access May 2014.
[14] "WebAnywhere", Available at: http://webanywhere.cs.washington.edu/beta/, last accessed May 2014.
[15] "WebAnywhere Open Source Site at Google Code", Available at: .http://webanywhere.googlecode.com, last accessed January 2012.
[16] Y. Y. Chang, "Evaluation of TTS systems in intelligibility and comprehension tasks", In Proceedings of the 23rd Conference on Computational Linguistics and Speech Processing, Association for Computational Linguistics, pp. 64-78, 2011.

Authors' Profiles

Najwa K. Bakhsh was born in Jeddah, Kingdom of Saudi Arabia (KSA). She is currently a computer science postgraduate student at King Abdulaziz University, Jeddah, KSA. She obtained her bachelor's degree in computer science from King Abdulaziz University, Jeddah, KSA, in 2003. She is currently working as a programmer at King Abdulaziz University. Her research interests include Web mining, Web accessibility, cloud technology and parallel algorithm.

Dr. Saleh Alshomrani is an Associate Professor of Information Systems Department at University of Jeddah. He is also serving now as the Dean of Faculty of Computing and Information Technology, University of Jeddah. He earned his Bachelor degree in Computer Science (BSc- with Honor) from King Abdulaziz University, Saudi Arabia 1997. He received his Master degree in Computer Science from Ohio University, USA 2001. He Also earned his Ph.D. in Computer Science from Kent State University 2008, Ohio, USA, in the field of Internet and Web-based Distributed Systems and Technologies. His research areas include: Data Mining, Algorithms, Computing Education, E-Learning, and E-Government, and Web Programming and Technologies.

Dr. Imtiaz H. Khan is an Associate Professor in the Department of Computer Science at King Abdulaziz University, Jeddah, KSA. He received his master's degree in computer science from the University of Essex, UK, in 2005. He earned his Ph.D. in artificial intelligence from the University of Aberdeen, UK, in 2010. His research interests are natural language processing, particularly natural language generation and evolutionary computation.

How to cite this paper: Najwa K. Bakhsh, Saleh Alshomrani, Imtiaz Hussain Khan,"IReadWeb: Towards Best Performance of WebAnyWhere", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.9, No.4, pp.13-20, 2017. DOI: 10.5815/ijieeb.2017.04.03