Enabling Scaleable, Efficient, Non-Visual Web Browsing Services

Ashish Verma
IBM India Research Lab
4 Block C, Institutional Area, Vasant Kunj, New Delhi, 110070. India
vashish@in.ibm.com

Tyrone Grandison
IBM Almaden Research Center
650 Harry Road, San Jose,
California 95120, USA
tyroneg@us.ibm.com

Himanshu Chauhan
IBM India Research Lab
4 Block C, Institutional Area, Vasant Kunj, New Delhi, 110070. India
himchauh@in.ibm.com

ABSTRACT

Over the last few decades, the discipline of Web Accessibility has been focused on building more efficient and more effective speech generators for Web Browsers. The visual browser interface is central to the current paradigm. However, in many cases, visual interaction is not required or desired, e.g. it is not relevant for blind people. More generally, when the input and output points are WAV files, SMS messages or natural queries, it becomes very clear that going through the visual user interface is overkill. In this paper we introduce a solution to this problem - a scalable, efficient, non-visual web browser that works with a Web whose central assumption is that visual interaction is an integral part of the user experience.

1. INTRODUCTION

While all of the efforts in the contemporary Accessibility research [1-6] are quite significant in helping the traditional Web browser users access information online faster and more efficiently, they still require the users to be familiar with the Web browsing concept and have access to a computer and so are not well suited for the part of the world population where non-computer based technologies need to be applied, i.e. the developing world.

Our solution, named CORBAC (CORe Browser for ACcessibility), is required because current techniques are 1) based on incremental benefits for impaired people (whether visually or otherwise impaired), 2) not amenable to scaling to a large number of non-computer users (e.g. a large mobile phone network), and 3) still based on a visual browser which has no relevance to blind/poor/low literate people.

2. MOTIVATION

The main motivation behind CORBAC is that the information on the Web should be accessible in a pervasive manner to a much larger population who don't and or can't user computers and have access to simple devices, such as phones. CORBAC represents a paradigm shift for Accessibility research - as it removes the (currently) key assumption that the screen is critical in meeting the needs of the disabled or impaired user.

3. SYSTEM

CORBAC consists of two main components: a core browser service and a simulator. The core browser engine handles web languages, e.g. HTML, PHP, Flash, etc and detects & blocks visual-only material, such as images, videos and animations. The simulator mimics visual interactions, e.g. menu selections, radio button clicks, form field - filling, etc., and is designed to be extensible to handle current and future dynamic Web technologies, e.g. JS, Perl, Ruby on Rails, etc.

Figure 1 demonstrates the typical operation of our system. A disabled/impaired user uses an Accessibility Interaction Point (AIP) to send a request to an Accessibility Action Point (AAP), which transforms the request into a standard canonical form. After transformation, the request is sent to the CORBAC browser, which gathers the accessibility-friendly version of the requested web-page(s) and returns the result to the AAP, which then performs another transformation into the destination language and passes back to the AIP for consumption by the user.

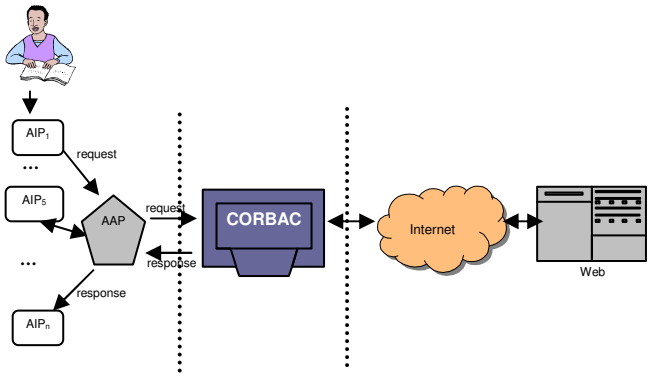

Figure 1. Insert caption to place caption below figure.

Each AIP handles a particular modality (or set of modalities), e.g. one AIP may be dedicated to receiving voice commands from a mobile phone user, whereas another AIP may be developed to accept and send SMS messages. The AAP functions as a generic translator, e.g. from speech to VXML, from text message to xml, from ?xml to some destination protocol.

Figure 2 highlights the inner workings of the CORBAC Browser. When a request enters the CORBAC browser, it is sent to Simulator, which makes a call to the Data Fetcher (DF) to retrieve the non-visual elements of the page(s). If there are any dynamic/active components in the Web page(s) returned, then the Simulator uses the Active Component Framework (ACF) to figure out how to interpret that particular content. This translates to instantiating the actions, given the information provided by the ACF. Each of these instantiations may lead to more requests to the DF. When all the instantiations have been made, the complete web page is sent to the CORBAC interface, which sends the information back to AAP (in the standard canonical form, referred to as ?xml in Figure 2).

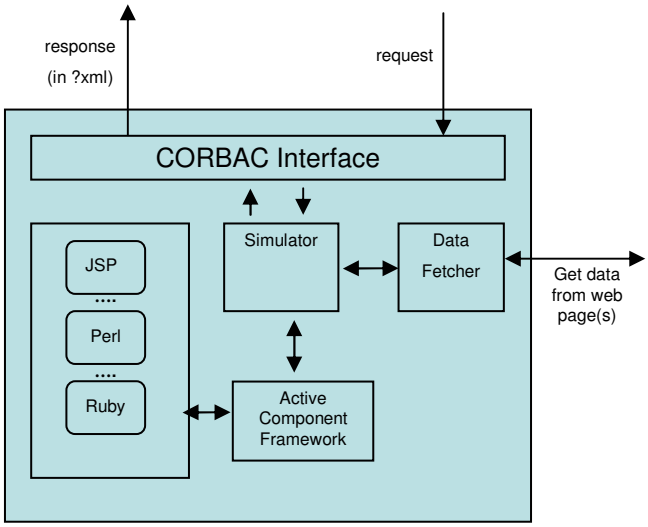

Figure 2. Inside the CORBAC Box.

The Active Models component contains a directory of various objects and the actions that can be performed on them, on a per language basis. Thus, if a new dynamic language starts being used by Web Browser, this system needs only to update the Active Models components with new definitions for that language. In a nutshell, the Active Models component is a modifiable library, which means that models can be updated, new models added and old models deleted as active technology becomes obsolete. This ensures that the system can leverage new and emerging active/dynamic web technologies.

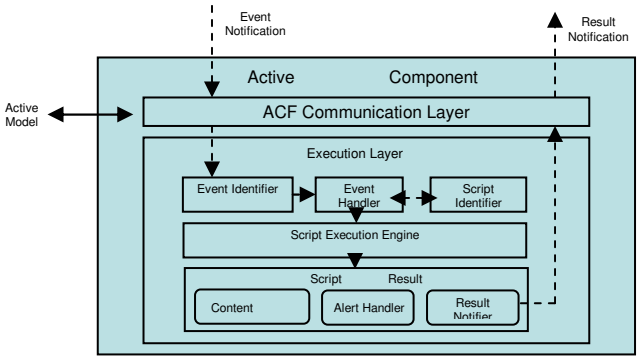

Figure 3. Active Component Framework Details.

Figure 3 shows the details of the ACF and its components. Communication between the Simulator and the Active Component Framework are facilitated by an interface implementation called ACF Communication Layer. This is a single point of communication with ACF and hence all requests for execution of client side scripts/dynamic content modifications which come to ACF are channeled though this communication layer. It also interfaces with the Active Models components, in order to determine how to handle particular objects and actions in the language in use. Result details and actions (which need to be taken by Simulator after script execution) and also communicated to the Simulator are also provided by this layer.

In an implementation scenario, a notification about execution of an event is forwarded to ACF (communication layer); for example: a user selecting an option from a listbox. Upon receiving the notification, ACF Communication layer interprets it in the context of the language (using the Active Models component) and forwards it to Event Identifier module, which identifies specific details, like id and text of option selected in the listbox, of the event and forwards them to Event Handler module. The Event Handler module interacts with the Script Identifier module to identify the script which should be executed for the event. Once Script Identifier returns the script(s) that should be executed, for example a JavaScript function making a hidden text-box visible, the Event Handler requests the Script Execution Engine to execute that script/function. The Script Execution Engine, as the name suggests, is an engine to execute client side scripts. After the execution of the script, the engine delegates the execution result to the Script Result Handler module. This module handles the result from the script execution and takes appropriate action, such as making the hidden text box visible. However, the result handler has different components to handle different results: The Content Modifier to modify the page content, The Alert Handler to handle alert notifications etc. In case there is such a need to handle the result, the corresponding handler is called to take the action. After which the Result Notifier component, of the Script Result Handler module, sends a notification to ACF Communication Layer about the result, which is further propagated to the Simulator, so that the Simulator can take appropriate action based on the result, such as registering that a new text box is visible on the page now.

4. BENEFITS

The advantages of CORBAC are that 1) it enables the information access through many interfaces (not only computers), 2) it improves the speed of web browsing for the disabled user, 3) it reduces the resources consumed when surfing the Web, and 4) it scales beyond current techniques.

5. REFERENCES

[1] Huixiang Gu, Jianming Li, Ben Walter and Eric Chang, "Spoken Query for Web Search and Navigation", Proc. of Internaltional WWW Conference, HongKong 2001.

[2] Dong Lin, Lin Bigin, Yuan Bao-Zong, "Using Chinese Spoken-Language Access to the WWW", Proc. of International Conference on WCCC-ICSP, Volume 2, pages:1321-1324, 2000.

[3] J. Mahmud, Y. Borodin and I.V. Ramakrishnan, "CSurf: A Context-Driven Non-Visual Web-Browser", International World Wide Web Conference WWW'2007, 8-12 May 2007, Banff, Canada.

[4] J. P. Bigham, T. A. Lau and J. W. Nichols, "TrailBlazer: Enabling Blind Users to Blaze Trails Through the Web", submittted to International Conference on Intelligent User Interfaces, Florida, 2009.

[5] Arun Kumar, Nitendra Rajput, Dipanjan Chakraborty, Sheetal K. Agarwal and Amit Anil Nanavati, "WWTW: The WorldWide Telecom Web", NSDR 2007 (SIGCOMM workshop), Kyoto, Japan 2007.

[6] Sheetal Agarwal, Arun Kumar, Amit Anil Nanavati, Nitendra Rajput, "The WorldWide Telecom Web Browser", Proc. of International WWW Conference, Beijing, China, 2008