International Journal of Computer Applications Technology and Research
Volume 8–Issue 08, 299-302, 2019, ISSN:-2319–8656

An Enhanced Browser Reference Model

Harun Kamau
School of Computing and Informatics
Maseno University
Maseno, Kenya

Dr.O.McOyowo
School of Computing and Informatics
Maseno University,
Maseno, Kenya

Dr.O.Okoyo
School of Computing and Informatics
Maseno University,
Maseno, Kenya

Abstract: Browsers are prime software applications in modern computing devices. They are essential in accessing internet rich content. Access to these contents pose a high memory demand on the host device thus affecting the user browsing experience and running of other programs. The architectural model adopted by the current browsers lacks a memory control mechanism that would prevent memory hogging which results to device crawl. The paper critically addresses the weaknesses of the contemporary browser reference architecture with a view to controlling memory hogging by integrating a memory analyzer into existing architecture.

Keywords: Browser reference architecture, memory hogging, web browser

1. INTRODUCTION

The Internet is gradually becoming a requisite element of modern generation. It is heavily relied upon in education sector where teaching and learning methods have gone digital. Business transactions have been digitized to reflect global reach and improve efficiency. Moreover, social and communication activities are being performed in a manner that makes the world a global village (Sagar A. et al., 2010). A web browser is prime software while seeking to realize the mentioned activities. While efficiency and multiprogramming is desired in the target computing devices, memory becomes an impending factor in realizing state of art performances. Studies have shown that browsers are memory ravenous and their consumption is dynamic contrary to generic computer programs (Doug DePerry, 2012).

The architectural model has been established to be a contributing factor to memory hogging which to leads to a computer freeze (Kamau, 2018). Modern browsers including Mozilla Firefox, Chrome, and Internet Explorer are derived from the reference architecture postulated by Allan and Michael (2006). In this architectural model, the browser continuously requests memory from the operating system to load the content it has fetched. This phenomenon leads to memory hogging and thus reduces the degree of multiprogramming. In single-processor systems, this phenomenon is undesired. To avert this problem, modification of the current model becomes a necessity. This is done with a view to providing memory control mechanism that would limit the maximum amount of memory a browser can use. This prevents memory hogging and thus increases the level of multiprogramming.

1.1 Contemporary Model

The architecture constitutes five major modules, which include User interface, Browser engine, Rendering engine, Display backend and Data persistence. This model was derived by Allan and Michael in 2006. These modules work collaboratively to interpret intricate protocols and provide a visual display of the URL fetched, (Paulina Siva et al., 2016). Modules functionality is discussed in the subsections herein. An illustration of the interaction of the mentioned modules is as shown in figure 1.

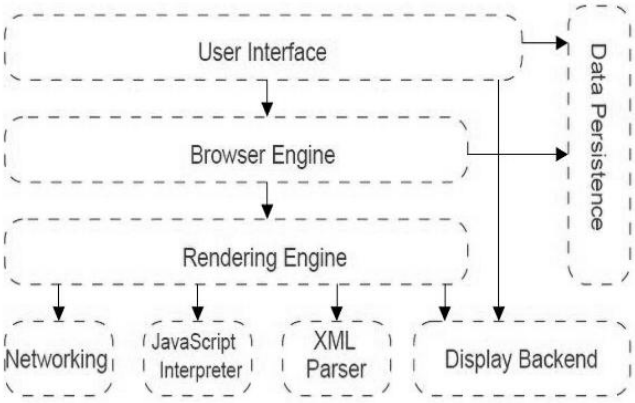

Figure 1:Browser reference architecture.

1.1.1User Interface

This module provides the methods with which a user interacts with the Browser Engine. It provides standard web browser

features including user preferences, printing functionality, downloading, opening and closing tabs etc. Browser designers have variant approaches in designing the user interface of the target browser. However, a given browser version depicts slight differences in user interface from another version of the same type. For instance, earlier versions of Mozilla Firefox had the reload button positioned to the right of the address bar while current versions have positioned to the left.

1.1.2 Browser Engine

This module provides a high-level interface to the Rendering Engine. It provides methods to initiate the loading of a URL and other high-level browsing actions like reload, back and forward. Furthermore, it provides the User interface with various messages relating to error messages and loading progress. When the browser fails to fetch the content specified by the URL, appropriate messages are conveyed to the User Interface, seeking intervention of the browser user.

1.1.3 Rendering Engine

This module provides the visual representation of the fetched URL. It comprises various subsystems that enable the browser to interpret the content of the URL. A URL contains two major parts: protocol and web resource. The protocol defines the mechanism through which resource will be fetched. Common protocols include HTTP and FTP. Web resources include text documents, images/graphics, audio and video. The multimedia content is interpreted by the appropriate parser to visually human-readable format. A prime component of the Rendering Engine is the HTML parser. The HTML parser is often tightly integrated with the rendering engine for performance reasons and can provide varying levels of support for broken or nonstandard HTML .It can display other types of data via plug-ins or extension; for example, displaying PDF documents using a PDF viewer plug-in. The rendering engine has XML parser sub system that parses XML data. The JavaScript content in the URL is interpreted by the JavaScript Interpreter. Detailed functionality of mentioned subsystems is discussed in sub sections below. Different browsers use different rendering engines: Internet Explorer uses Trident, Firefox uses Gecko, and Safari uses WebKit. Chrome and Opera (from version 15) use Blink, a fork of WebKit.

1.1.3.1 Networking Component

This component provides functionality to handle URLs retrieval using the common Internet protocols like HTTP and FTP. It handles all aspects of Internet communication and security; character set translations and Multi-Purpose Internet Mail Extensions (MIME) type resolution. This component may also implement a cache of retrieved documents to minimize network traffic

1.1.3.2 JavaScript Interpreter

This component executes the JavaScript code that is embedded in a URL. Results of the execution are passed to the Rendering Engine for display. The Rendering Engine may disable various actions based on user defined properties. Where the browser user has set JavaScript code to be disabled, the rendering engine ignores the interpreted material.

1.1.3.3 XML Parser

This is a software library or a package that provides an interface for client applications to work with XML documents. It is generic and reusable component with a standard that has well defined interface. It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers. The goal of a parser is to transform XML data into a human-readable code.

1.1.4 Display/UI Backend

This component is tightly coupled with the host operating system. It provides primitive drawing and windowing methods that are host operating system dependent. Common widgets like combo box, an input box, a check box, etc are drawn using UI properties.

1.1.5 Data Persistence

The Data Persistence component manages user's data such as bookmarks, cookies and preferences. The browser may need to save all sorts of data locally. Browsers also support storage mechanisms such as localStorage, IndexedDB, WebSQL and FileSystem (Michael Coates, 2010).

1.2 Flaws of the Current Browser Reference Architecture

The contemporary architecture has two main weaknesses which are outlined herein.

i. The rendering engine processes the requests made by the browser engine by rendering the fetched content provided there is little memory available for use by the browser. If the operating system can no longer allocate any more memory, the computer freezes hence becomes unusable.

ii. The browser process prevents other legitimate processes from being loaded in the main memory if it consumes almost all-available memory. This reduces the level of multiprogramming.

2. METHODS

The enhanced model was anchored on browser reference architecture highlighted in figure 1

2.1 Necessity to Modify Browser Reference Architecture

From the weaknesses highlighted in section 1.2, there was need to restructure the architecture to provide a control mechanism for browser memory usage. While seeking to address this problem, the researcher opted to integrate a memory analyzer to the contemporary browser reference architecture.

2.2 The Enhanced Browser Reference Architecture

The enhanced architecture incorporates the Memory Analyzer component as shown in figure 2. The memory analyzer component interacts with the operating system to track memory usage in real-time and to check browser memory consumption against the set threshold total memory. After analysis, the browser user is provided with possible actions to take to prevent memory hogging. Consequently, more applications can be loaded into the main memory awaiting execution. This guarantees that browsers do not make computer to freeze by delimiting other legitimate applications from running. As a result, it improves the level of multiprogramming and ultimately improves user-browsing experience. The analyzer is implemented as a software module included in the web browser application.

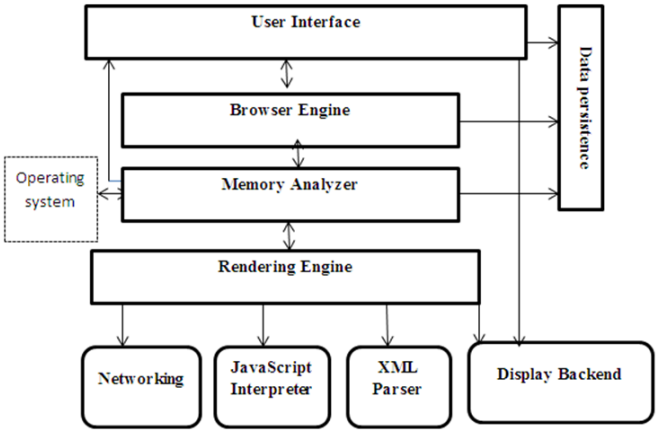

Figure 2: The enhanced browser reference architecture

2.2.1Memory Analyzer

This component checks real-time memory consumption for the browser against the threshold total memory limit set by the user and gives feedback information to the user on possible actions to take to prevent memory hogging by the browser. Memory analysis is done after the browser engine has retrieved a resource. The rendering engine interprets and gives a visual representation of the URL with the help of parsers and JavaScript interpreter if memory space is available. The integration provides memory control mechanism that hence controls memory hogging. Furthermore, the analyzer provides garbage collection mechanism to reclaim unused memory from the browser objects.

2.2.2 Flow diagram of memory analyzer

A conceptualized design of a Memory Analyzer and its interactions with other modules is as shown in figure 3. When a user enters a URL on the browser's address bar and hits the Go button, the Browser Engine takes the URL and attempts to fetch its content. The Memory Analyzer performs analysis of the memory consumed against the threshold memory as set by the user. If the memory is lower than the threshold memory, it passes the content of the URL to the rendering engine for further actions. However, if the consumed memory gets higher than the threshold memory, a notification error message is passed to the higher modules for action to be taken by the user.

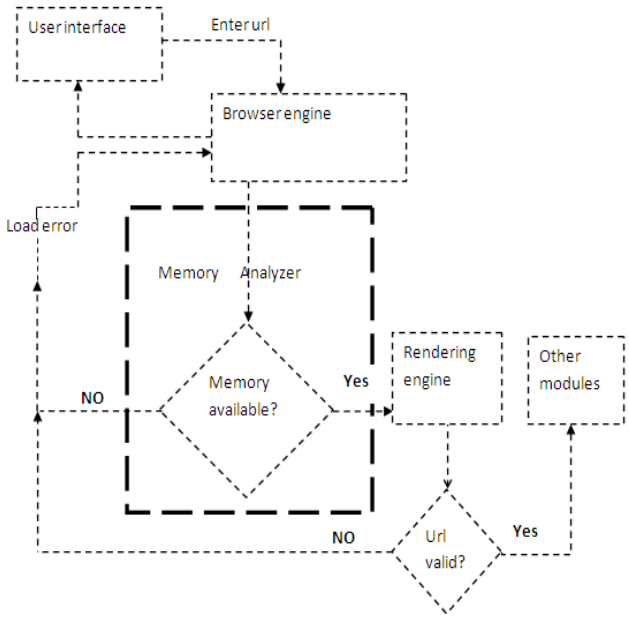

Figure 3: The Flow diagram of a memory analyzer

3. CONCLUSION

Based on the structure of the current browser architecture, it is evident enough that the model lacks a memory control mechanism and thus memory hogging becomes a habitual phenomenon in browser applications. In attempt to solve this problem, the memory analyzer was integrated to the current model with a view to providing memory control mechanism. Further, the module notifies the user when memory hogging is detected. The researcher designed a browser prototype and integrated the memory analyzer in it.

4. RECOMMENDATION

An evaluation of the enhanced browser reference should be done to unveil its performance with regard to memory consumption and its overall impact on user browsing experience.

5. REFERENCES

[1] A. E. Hassan and R. C. Holt, (2000). A reference architecture for web servers. In Proceedings of 7th the Working Conference on Reverse Engineering (WCRE ’00), pp. 150–160, 2000.

[2] A. Taivalsaari and T. Mikkonen, (2011). "The Web as an Application Platform: The Saga Continues," *Proc. 37th Euromicro Conf. Software Engineering and Advanced Applications* (SEAA 11), IEEE CS, 2011, pp. 170–174.

[3] A. Taivalsaari et al., (2008). Web Browser as an Application Platform: The Lively Kernel Experience, tech. report TR-2008-175, Sun Microsystems Labs, 2008.

[4] Accuvant Labs, 2011: Browser Security Comparison; A Quantitative Approach. Retrieved from http://files.accuvant.com/web/files/AccuvantBrowserSec C ompar_FINAL.pdf

[5] Alan Grosskurth and Michael W. Godfrey, (2005) Reference architecture for web browsers. In ICSM'05: Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05), pp 661-664, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Alan Grosskurth, Michael W. Godfrey ,(2006) Architecture and evolution of the modern web browser. Retrieved from http://grosskurth.ca/papers/browser- archevol-20060619.pdf

[7] Allan Grosskurth and Michael Godfrey, (2014). Reference architecture for web browsers. In Journal of Software Maintenance and Evolution: Research and Practice, pp 1–7, 2006

[8] Chris Anderson (2012). The Man Who Makes the Future: Wired Icon Marc Andreessen. Retrieved from http://www.wired.com/2012/04/ff_andreessen/all/

[9] Doug Deperry, (2012). HTML5 security in the modern web browser perspective**.**

[10] Kamau, H.,McOyowo, S. & Okoyo, H. (2018): *Techniques to control memory hogging by web browsers.* International Journal of Computer Applications Technology and Research. Vol. 7 issue 04,2018

[11] Matthew Braga (2011): Web Browser Showdown: Memory Management Tested. Retrieved from http://www.tested.com/tech/web/2420-web-browser-showdown-memory-management-tested/index.php

[12] Michael Coates (2010). *A journey in Security*. HTML5, Local Storage, and XSS. Retrieved from http://michaelcoates.blogspot.com/2010/07/html5-local-storage-and-xss.html

[13] Paulina S., Raúl M., & Eduardo B. (2016). *A Reference Architecture for web browsers*: Part I, A pattern for Web Browser Communication

[14] Vrbanec, T., Kiric, N. & Varga, M. (2013). “The evolution of web browser architecture”. SCIECONF 2013, pp. 472–480.

[15] W3C (2004). Architecture of the World Wide Web, Volume one. Retrieved from http://www.w3.org/TR/webarch/