International Journal of Computer Applications Technology and Research
Volume 7–Issue 04, 185-192, 2018, ISSN:-2319–8656

Techniques to Control Memory Hogging by Web Browsers: An in-Depth Review

Harun K. 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

Dr.C.Ratemo
School of Computing and Informatics
Maseno University, Maseno, Kenya

Abstract: The Web Browser is to date a popular piece of software in modern computing systems. They are the main interface for vast information access from the Internet. Browsers technologies have advanced to a stage where they do more than before. They now parse not only plaintext and Hypertext Markup Language (HTML), but also images, videos and other intricate protocols. These advancements have increased demand for memory. This increased demand poses a challenge in multiprogramming environments. The contemporary browser reference model does not have a memory control mechanism that can limit maximum memory a browser can use. This leads to hogging of memory by contemporary browsers. This paper is a review on emergent techniques that have been used to control memory hogging by browsers based on the contemporary reference architecture. We review major browsers architectures including Mozilla Firefox, Google Chrome and Internet explorer. We give an in-depth study on techniques that have been adopted with a view to solve this problem. From these reviews we derive the weaknesses of the contemporary browser architecture and inefficiency of each technique used.

Keywords: Browser reference architecture, memory hogging, web browser

1. INTRODUCTION

The Internet is progressively becoming an indispensable component of today's life. Most often than not, people largely rely on the expediency and elasticity of Internet-connected devices in learning, shopping, entertainment, communication and in broad-spectrum activities, that would otherwise necessitate their physical presence (Sagar A. et. al., 2010). Access to information or services via the Internet requires a medium; a browser operates as a medium. It is the prime component of a computer system when the Internet services are required. A browser retrieves, displays and traverses information resources on the web (World Wide Web Consortium, 2004).

Information resources comprise text, image, video, or other piece of content. These resources are accessed and indentified by a Uniform Resource Identifier (URI). The first browser known as WorldWideWeb was made in the early 1990s by Tim Berners-Lee (Tim Berners-Lee, 1999). Since then, browsers have seen tremendous advancements in their architectures and usage. The earliest browsers; Nexus, Mosaic and Netscape were less complex and used considerably low computer memory. However, they were commonly used for viewing basic HTML pages. With the birth of the Internet, browsers have gained a lot of popularity globally.

1.1 Motivation

Today, the browser is the most used computer application (Allan and Michael, 2006; Antero et. al., 2008). This phenomenon may be attributed to its various usages in everyday life. With limited computer power to process voluminous data generated from various sources, users have resorted to other technologies like the cloud computing and other online solutions where there is robust computer processing power, vast storage, scalability, reliability and on demand services. In these cases, resources are accessed as services via the Internet with thin clients especially the browsers.

Originally, Web information comprised a set of documents that in most cases contained text and hyperlinks to other related documents, having little or no client-side code. All rendered content originated from a single source. Web content has increasingly become more complex in pursuit to incorporate interactive features. Today, web programs have advanced to become highly interactive applications that execute on both the server side and client machine. With these advancements, modern web pages are no longer simple documents. They comprise highly dynamic contents that work together with each other. In other words, a Web page is now said to be a "system"– having dynamic contents as programs running in it, interacting

with users, accessing other contents both on the web page and in the hosting browser, invoking browser Application Programming Interfaces (APIs), and interacting with programs on the server side. These advancements require adequate computer memory in order to run properly from a host computer.

Consequently, these advancements in content rendering have raised memory demand browsers. In fact, memory allocation to a browser rises gradually from tens of Megabytes (Mbs), to hundreds of Mbs and eventually to Gigabytes (Doug DePerry, 2012). This fact only, categorizes browsers as today's memory "wolfs". Indeed, it leads to browser crash. The size of Random Access Memory (RAM) is an important factor in the running of software and consequently determines the level of multiprogramming. A single process consuming nearly a gigabyte of RAM in a one GB computer will lead to starvation of other processes and therefore lower multiprogramming level. This starvation may eventually lead to a crawl. However, these browsers behave differently in different platforms and with the content, the browser loads.

2. METHODS

The works reviewed were based contemporary browsers architecture and optimization techniques adopted thereon.

2. 1 Introduction

Today, browsers have advanced in terms of content rendering. This has raised memory demand for browsers. In fact, memory allocation to a browser rises gradually from tens of Megabytes (Mbs), to hundreds of Mbs and eventually to Gigabytes (Doug DePerry, 2012). This fact only categorizes browsers as today's memory wolfs. The size of RAM is an important factor in the running of software and consequently determines the level of multiprogramming; especially when it comes to browser efficiency. A single process consuming nearly a gigabyte of RAM in a one GB computer will lead to starvation of other processes and therefore lower multiprogramming level. This starvation may eventually lead to a crawl and even lead to the browser crashing. However, browsers behave differently in different platforms and with the content, they display.

2.2 Causes of Memory Hogging

Many users from the respective browser forums have regularly affirmed that memory hogging is attributed by several factors. To begin with is the length of the time the browser is used. As the browser gets used, gradually it will take more time to load during startup, the speed might decrease; and browsing eventually starts to slow down. This is a very frequent problem and occurs partially because of fragmentation in the databases browsers use. In particular, if Firefox is left running for a number of hours, consumed memory of well over a Gigabyte is observed even with only a few tabs open; a long running memory leak issue that plagues Firefox sometimes (Doug DePerry, 2012).

Secondly, when a user opens many tabs simultaneously, the browser will use more RAM. This is because each tab is designed to cache pictures, text and other active data, which keeps page data persistent while using multiple tabs. Expectedly, browsers such as Chrome and Firefox have ways to turn this behavior off, but the user may not wish it to happen. This is because, without caching, YouTube videos will not play in the background, and most real-time web apps will fail to work correctly.

Memory leakage is another factor. A memory leak happens when the browser for some reason doesn't release memory from objects which are not needed any more. This may happen because of browser bugs, browser extension problems and rarely, due to browser developer mistakes in the code architecture. Leaks may occur because of browser extensions, interacting with the page. More importantly, a leak may occur because of two extensions interaction bugs **(**Ilya Kantor, 2011). For instance, when Skype extension and the Antivirus are enabled, it leaks and when any of them is off, it doesn't.

2.3 Browser Reference Architecture

The following illustrates the convectional browser architecture adopted while building the contemporary browsers. We keenly look at how specific browsers have adopted this model.

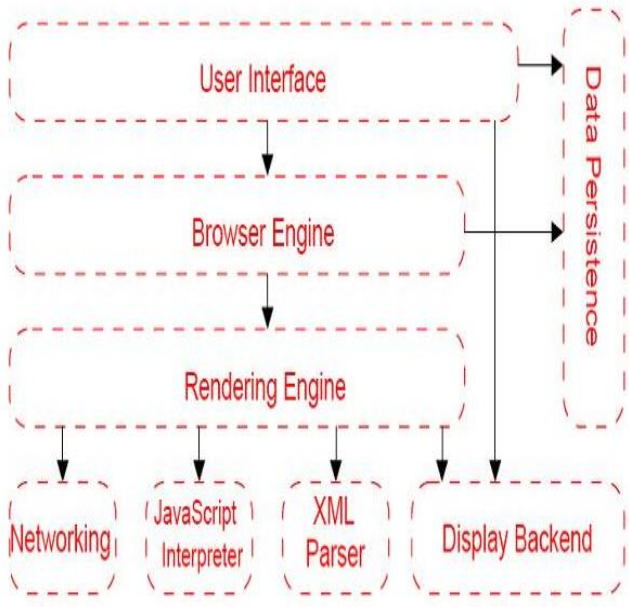


Figure 1: Browser Reference Architecture

The **User Interface** component provides the methods with which a user interacts with the Browser Engine. The User Interface provides standard features (preferences, printing, downloading, and toolbars) users expect when dealing with a desktop application.

The **Browser Engine** component provides a high-level interface to the Rendering Engine. The Browser Engine provides methods to initiate the loading of a Uniform Resource Locator (URL) and other high-level browsing actions (reload, back, forward). The Browser Engine also provides the User interface with various messages relating to error messages and loading progress.

The **Rendering Engine** component produces the visual representation of a given URL. The Rendering Engine interprets the HTML, Extensible Markup Language (XML), and JavaScript that comprises a given URL and generates the layout that is displayed in the User Interface. A key component of the Rendering Engine is the HTML parser, this HTML parser is quite complex because it allows the Rendering Engine to display poorly formed HTML pages.

The **Networking component** provides functionality to handle URLs retrieval using the common Internet protocols of Hypertext Transfer Protocol (HTTP), Hyper Text Transfer Protocol Secure (HTTPS) and File Transfer Protocol (FTP). The Networking components handle all aspects of Internet communication and security, character set translations and MIME type resolution. The Network component may implement a cache of retrieved documents to minimize network traffic.

The **JavaScript Interpreter** component executes the JavaScript code that is embedded in a website. Results of the execution are passed to the Rendering Engine for display. The Rendering Engine may disable various actions based on user defined properties.

The **XML Parser** component is used to parse XML documents.

The **Display Backend** component is tightly coupled with the host operating system. It provides primitive drawing and windowing methods that are host operating system dependent.

The **Data Persistence** component manages user data such as bookmarks and preferences.

3. BROWSER ARCHITECTURES

In a view to find how browsers have been developed, their architectures were reviewed to find out whether they are true derivations from the reference architecture.

3.1 Google Chrome

Google Chrome uses a multi-process architecture which gives it a competitive edge in performance over other browsers. Each tab has its own process which runs independently from other tabs. This allows one tab process to dedicate itself to a single web-application, thereby increasing browser performance. This protects the browser application from bugs and glitches in the rendering engine. Furthermore, it restricts access from each rendering engine process to others and to the rest of the system. This scenario offers memory protection and access control as manifested in operating systems. The multi-process architecture also increases the stability of the browser, as it provides insulation. In the case that one process encounters a bug and crashes, the browser itself and the other applications running concurrently are preserved.

Function wise, this is an improvement over other browsers, as highly valuable user information in other tabs will be preserved. Google Chrome has used the WebKit as a layout engine until version 27. Later versions have been using Blink.V8 has been used as JavaScript Interpreter in all versions. The components of Chrome are distributed under various open source licenses. Although Google developers have variant components in their architectural design, they have derived it from the reference architecture.

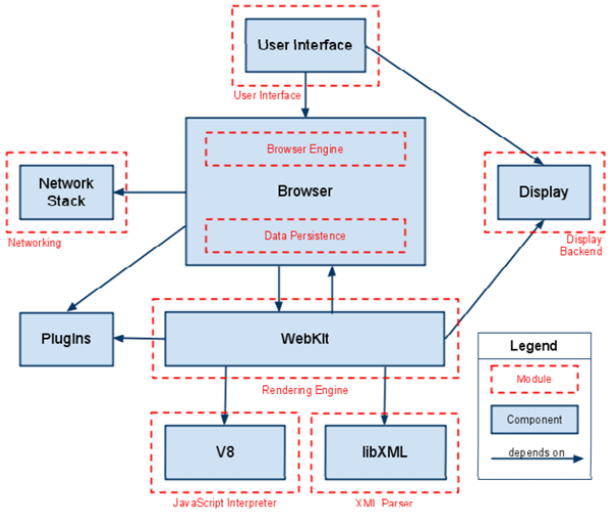


Figure 3.1: Google Chrome Architecture

3.2 Microsoft Internet Explorer

Essential to the browser's architecture is the use of the Component Object Model (COM), which governs the interaction of all of its components and enables component reuse and extensibility (MSDN, 2016). Internet Explorer uses Jscript and VBScript as JavaScript interpreter and Trident layout engine.

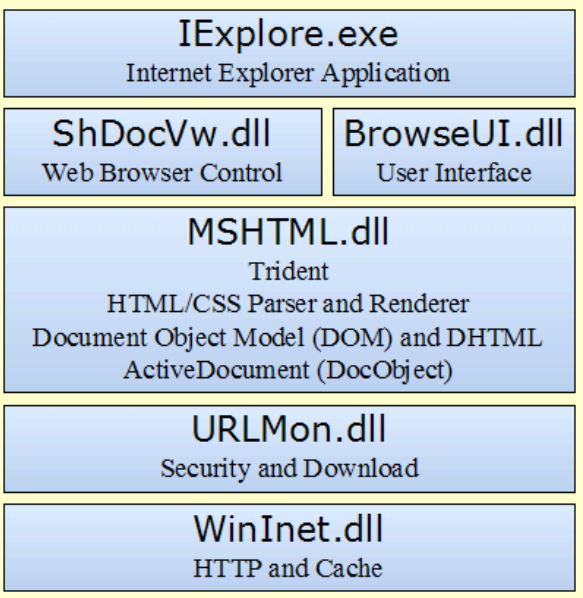


Figure 3.2: Internet Explorer Architecture

The following is a description of each of Microsoft Internet Explorer's six key framework components:

IExplore.exe is at the top level, and is the Internet Explorer executable. It is a small application that relies on the other main components of Internet Explorer to do the work of rendering, navigation, protocol implementation.

Browsui.dll provides the user interface to Internet Explorer. Often referred to as the "chrome," this Dynamic Link Library (DLL) includes the Internet Explorer address bar, status bar, menus, and so on.

Shdocvw.dll provides functionality such as navigation and history, and is commonly referred to as the WebBrowser control. This DLL exposes ActiveX Control interfaces, enabling you to easily host the DLL in a Windows application using frameworks such as Microsoft Visual Basic, Microsoft Foundation Classes (MFC), Active Template Library (ATL), or Microsoft .NET Windows Forms. When your application hosts 9999 the WebBrowser control, it obtains all the functionality of Internet Explorer except for the user interface provided by Browseui.dll. This means that you will need to provide your own implementations of toolbars and menus.

Mshtml.dll is at the heart of Internet Explorer and takes care of its HTML and Cascading Style Sheets (CSS) parsing and rendering functionality. Mshtml.dll is sometimes referred to by its code name, "Trident". Mshtml.dll exposes interfaces that enable you to host it as an active document. Other applications such as Microsoft Word, Microsoft Excel, Microsoft Visio, and many non-Microsoft applications also expose active document interfaces so they can be hosted by shdocvw.dll. For example, when a user browses from an HTML page to a Word document, mshtml.dll is swapped out for the DLL provided by Word, which then renders that document type. Mshtml.dll may be called upon to host other components depending on the HTML document's content, such as scripting engines (for example, Microsoft JScript or Microsoft Visual Basic Scripting Edition (VBScript)), ActiveX controls, XML data,

Urlmon.dll offers functionality for MIME handling and code download.

WinInet.dll is the Windows Internet Protocol handler. It implements the HTTP and File Transfer Protocol (FTP) protocols along with cache management.

Microsoft's Internet Explorer architecture utilizes the reference model components though variant in design.IExplorer.exe is a wrapper for the whole application. **Browsui.dll** serves as user interface while **Shdocvw.dll** performs functions of a browser engine**.** The **Mshtml.dll** is the core component that serves as rendering engine**.** It has HTML, CSS, XML and JavaScript parsers. **WinInet.dll** provides networking functions as provided for in the reference architecture.

3.3 Mozilla Firefox

The following model has been used in the design of Mozilla Firefox (Andre C. et al. 2007).

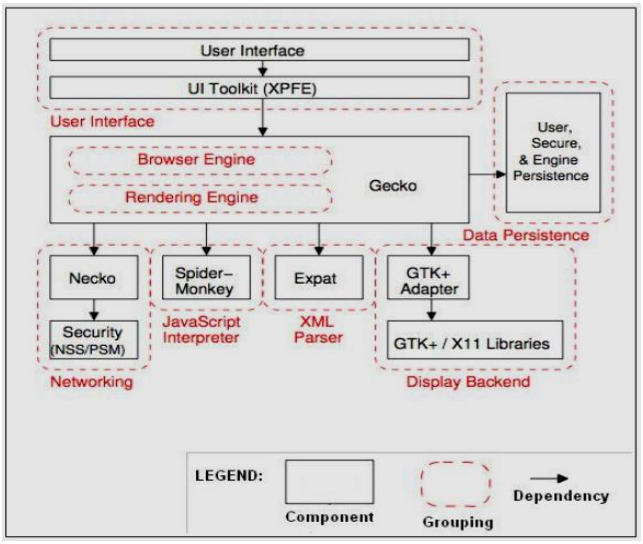


Figure 3.3: Mozilla Firefox architecture

The User Interface is split over two subsystems, allowing for parts of it to be reused in other applications in the Mozilla suite such as the mail/news client. All data persistence is provided by Mozilla's profile mechanism, which stores both high-level data such as bookmarks and low-level data such as a page cache.

Mozilla's Rendering Engine is larger and more complex than that of other browsers. One reason for this is Mozilla's excellent ability to parse and render malformed or broken HTML. Another reason is that the Rendering Engine also renders the application's cross-platform user interface. The User Interface (UI) is specified in platform-independent Extensible User Interface Language (XUL), which in turn is mapped onto platform-specific libraries using specially written adapter components. This architecture distinguishes Mozilla from other browsers in which the platform-specific display and widget libraries are used directly, and it minimizes the maintenance effort required to support multiple, diverse platforms.

Recently, the core of Mozilla has been transformed into a common runtime called XULRunner, exposing the Rendering Engine, Networking, JavaScript Interpreter, Display Backend, and Data Persistence subsystems to other applications. XULRunner allows developers to use modern web technologies to create rich client applications, as opposed to typical browser-based web applications. In fact, the Mozilla developers are working on transitioning newer Mozilla-based applications such as Firefox and Thunderbird to use XULRunner directly, rather than each using a separate copy of the core libraries. All components of this model fits exactly to those in the reference architecture.

3.4 Weaknesses of the Current Browser Architecture

a) The rendering engine processes the requests made by the browser engine by giving a visual display of the URL. This happens 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.

b) 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.

From the review of the above named architectures, memory hogging still remains a thorny issue. In attempt to reduce the its impact, third party software have been developed.

4. MEMORY OPTIMIZATION TECHNIQUES

To free memory that is unnecessary to the browser, several third party tools have been used. Memory optimization programs include but not limited to the following:

4.1 Firemin

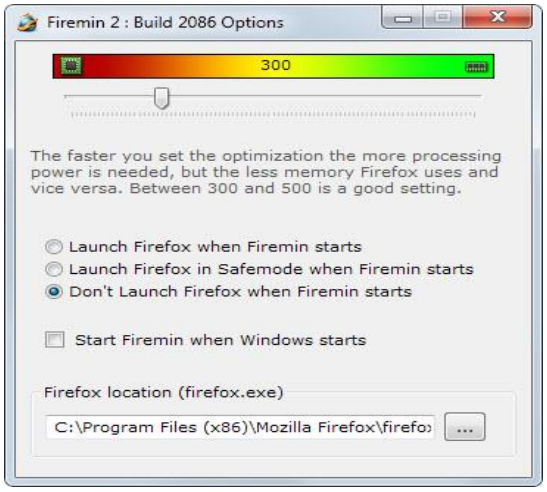
With Firemin for Firefox, you can effectively stop Firefox memory leaks automatically. As memory usage of this popular browser increases, your system slows down and you're stuck with limited system resources. In fact, Firefox can use up to 500MB of memory if you use the browser continuously (F. Ortega, 2013). Firemin forces Firefox to give back the memory it took from Windows and allows you to use Firefox in an optimized environment.

Firemin does not do anything that Windows does not do itself when the system runs out of RAM. It calls the Windows function EmptyWorkingSet over and over again in a loop to free up memory. Calling the function removes as many pages as possible from the working set of the specified process. The program ships with a slider that you can use to set the desired interval in which you want it to call the function.

However, the limitations of Firemin.exe are that, the technical security rating is 30% dangerous. This is because it records keyboard and mouse inputs, monitors applications and manipulates other programs. Moreover, some malware

camouflages itself as Firemin.exe, particularly when located in the C:\windows or C:\windows\System32 folder. Also, Firemin is only compatible with Mozilla Firefox.

Figure 4.1: Firemin



4.2 Wise Memory Optimizer

Wise Memory Optimizer helps you free up and tune up the physical memory taken up by some unknown non-beneficial applications to enhance PC performance. You can enable automatic optimization mode when the free PC memory goes below a value that you can specify, and make Wise Memory Optimizer run even when the CPU is idle, as well as adjust the amount of memory you want to free up. Then it will optimize PC memory automatically in the background.

However, this tool does not prevent the browser from hogging memory it only reclaims memory from unknown non-beneficial applications.

Figure 4.2: Wise memory optimizer

4.3 SpeedyFox

SpeedyFox is a tool designed specifically for compacting the SQLite database files which will in turn reduce the time taken to read from and write to them. In addition to Firefox which it was originally designed for, SpeedyFox can now also compact the databases for the Chrome, Epic Browser, SRWare Iron and Pale Moon browsers. It also supports the Mozilla Thunderbird and Skype tools as well.

Upon running the portable executable, SpeedyFox automatically detects and loads the default profile for each of the supported applications. As they're very popular these days, it's also possible to load custom profiles for Firefox or Chrome portable versions. Click the SpeedyFox menu bar and select "Add custom profile" or drag the profile folder and drop it onto the SpeedyFox window.

Simply tick the application profiles to optimize and click the Optimize! Button, SpeedyFox will start to compact the SQLite databases. The progress window shows what databases are optimized and also how much space is saved. You need to make sure the programs being optimized are not running at the time or they won't be processed. In a quick test it reduced 14MB of Firefox databases to 6MB and 192Mb of Chrome databases to 186MB. The author of SpeedyFox recommends running the tool every 1-2 weeks depending on your usage of the included browsers.

Though tool increases Mozilla Firefox launch speed, it does not prevent memory hogging. It just clears cache over some time.

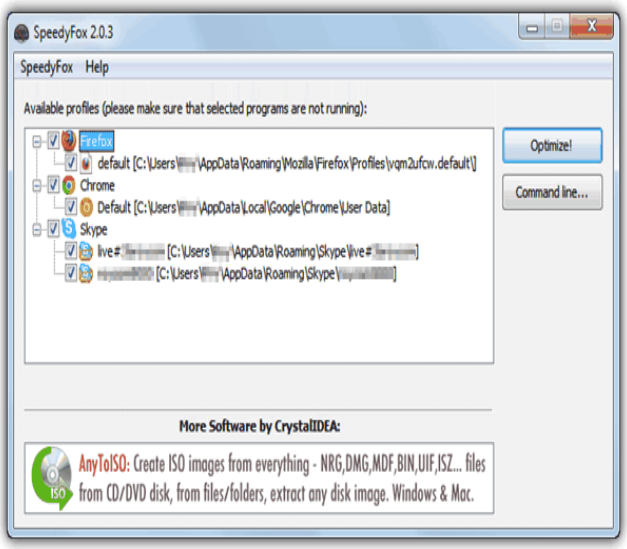


Figure 4.3: SpeedyFox

4.4 All Browsers Memory Zip

All Browsers Memory Zip has no database compacting functions but is a dedicated memory-optimizing tool for a large number of popular web browsers. It works very much like another memory optimizer called CleanMem, but this tool only handles browsers. In addition to Chrome and Firefox, it also works with other popular browsers like Opera, Internet

Explorer and Maxthon etc. The program is portable but has separate 32-bit and 64-bit versions, and when you run it there will be a small tooltip and then All Browsers Memory Zip will sit in the system tray optimizing the memory of any running supported browsers. If you open Task Manager (Ctrl+Shift+Esc) before you launch the tool, you will see the used memory for the browser process suddenly decreases by a massive amount. It is not uncommon to see 1GB+ in memory usage drop to fewer than 10MB in a few seconds.

Right click on the tray icon to pause the program from optimizing and pressing Usage Controller will popup the window above that will allow you to set the maximum amount or RAM for each browser and edit the shortcut keys. Just select the browser form the dropdown, enter the max amount in Megabytes and click Set.

This tool must execute all times a browser process is running. It requires a significant amount of memory. Consequently, it impacts negatively when streaming content over the Internet.

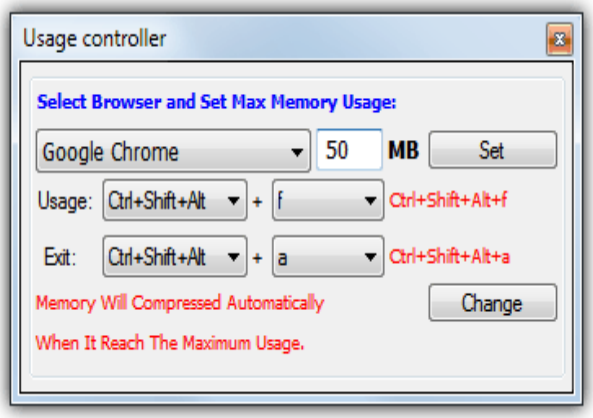


Figure 4.4: All browsers zip usage controller

5. CONCLUSION

A review on various techniques adopted in a view to control memory hogging was presented in this paper. It is evident enough that memory hogging among various browsers is and still a thorny issue. It is desired that computer applications use little memory and execute faster with a view to allow as many programs to be loaded in the main memory for execution. With browsers being among such applications, this still remains an issue under investigation. Many third party applications have been developed in quest to reduce memory consumption. These applications include Firemin, Wise Memory Optimizer, SpeedyFox and All browsers memory zip. After all these tools have been analyzed, it has been found out that, memory control is not efficient, poor compatibility issues, overhead to users and decrease in browser performance. An interesting issue has been found on the browser reference architecture. The contemporary architecture in use today aggravates this problem. It has been found out that, this model lacks memory control mechanism which would complement these third party applications. Today, browsers have become a major platform through which resources accessed via the Internet are availed to the user. Based on this fact, memory as prime resource has remained a major limitation while trying to run applications through the browser. This has become a major drawback in multi programming environment. A new approach to incorporate a memory analyzer in the architecture has been suggested. It is hoped that this shall control memory hogging and reduce overhead to the browser application while optimizing memory.

6. 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. Mockus, R. T. Fielding, and J. Herbsleb, (2002) Two case studies of open source software development: Apache and Mozilla. In ACM Trans. Software Engineering and Methodology, pp. 11(3), 309–346, 2002.

[3] 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.

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

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

[6] Adam Overa, 2011: Web Browser Grand Prix 3: IE9 Enters The Race. Efficiency Benchmarks: Memory Usage and Management. Retrieved from http://www.tomshardware.com/reviews/internet-explorer-9-chrome-10-opera-11,2897-11.html

[7] Adèr, H.J. & Mellenbergh, G.J. (2008). *Advising on Research Methods: A consultant’s companion.* Huizen, the Netherlands: Johannes van Kessel Publish.

[8] Ahmed E. Hassan, Michael W. Godfrey, and Richard C. Holt, n.d. Software Engineering Research in the Bazaar.

[9] 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.

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

[11] 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

[12] Avant Force, (2016). Retrieved from http://www.maxthon.com/about-us/

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

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

[15] Gnome desktop environment. Home Page. Retrieved from Http: //gnome.org.

[16] Karl Gephart (2013): Optimize Firefox's Performance with these Memory Add-Ons! : Retrieved from http://www.drakeintelgroup.com/2013/06/25/Firefox-memory-addons/

[17] Krause, Ralph (March 2000). Browser Comparison. Retrieved from http://www.linuxjournal.com/article/5413

[18] 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

[19] Maxthon International Ltd, (2014). Retrieved from http://www.maxthon.com/about-us/

[20] Nick Veitch, (August 2010). 8 of the best web browsers for Linux. Retrieved from http://www.techradar.com/news/software/applications/8-of-the-best-web-browsers-for-linux-706580/3

[21] Nyce, J. M. & Kahn P., (1991).From Memex To Hypertext: Vannevar Bush and the Mind's Machine. Academia press, San Diego.

[22] Opera Software, (February, 2003). "Opera version history". Retrieved from http://www.opera.com/docs/

[23] Pour, Andreas (January, 2003). "Apple Announces New "Safari" Browser". KDE Dot News. Retrieved from https://dot.kde.org/2003/01/08/apple-announces-new-safari-browser

[24] Sagar, A., Pratik, G., Rajwin, P. and Aditya, G., (2010). Market research on web browsers. Retrieved from http://www.slideshare.net/sagar_agrawal/research-on-web-browsers.

[25] T. Mikkonen and A. Taivalsaari, (2007) "Web Applications – Spaghetti Code for the 21 Century". Retrieved from http://research.sun.com/techrep/2007/abstract-166.html (presented in the SERA Conference, Prague, Czech Republic, August 21, 2008)

[26] T. Mikkonen and A. Taivalsaari, (2008) "Web Browser as an Application Platform: The Lively Kernel Experience" http://research.sun.com/techrep/2008/abstract-175.html (presented in the SEAA Conference, Parma, Italy, September 4, 2008)

[27] T. Mikkonen and A. Taivalsaari, (2011). "Apps vs. Open Web: The Battle of the Decade," *Proc. 2nd Workshop Software Eng. for Mobile Application Development* (MSE 11), 2011; Retrieved from http://www.mobileseworkshop.org/papers6-Mikkonen_Taivalsaari.pdf.

[28] Tim Berners-Lee (1999). Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. Harper San Francisco.

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

[30] Wayner, Peter (January, 2005). "BASICS; Custom Tailor A Web Browser Just for You", The New York Times, ISSN 0362-4331, OCLC 1645522. Retrieved from http://query.nytimes.com/gst/fullpage.html?res=9C0DE6D9163BF934A15752C0A9639C8B63