JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE
J. Softw. Maint. Evol.: Res. Pract. 2006; **00**:1–7 *Prepared using* smrauth.cls *[Version: 2003/05/07 v1.1]*

Research

A reference architecture for web browsers

Alan Grosskurth*, † and Michael W. Godfrey‡

David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada

SUMMARY

A reference architecture for a domain captures the fundamental subsystems common to systems of that domain, as well as the relationships between these subsystems. The presence of a reference architecture can aid both during maintenance and at design time: it can improve understanding of a given system, it can aid in analyzing trade-offs between different design options, and it can serve as a template for designing new systems and reengineering existing ones. In this paper, we examine the history of the web browser domain and identify several underlying forces that have contributed to its evolution. We develop a reference architecture for web browsers based on two well known open source implementations, and we validate it against five additional implementations. We analyze the maintenance implications of different strategies for code reuse, and we identify several underlying evolutionary phenomena in the web browser domain; namely, *emergent domain boundaries*, *convergent evolution*, and *tension between open and closed source development approaches*.

KEY WORDS: Software architecture, reference architecture, software evolution, component reuse, web browser.

INTRODUCTION

A *reference architecture*[20] for a domain captures the fundamental subsystems and relationships between them that are common to existing systems in the domain. It aids in the understanding of these systems, some of which may not have their own specific architectural documentation. It also serves as a template for creating new systems by identifying areas in which reuse can occur, both at the design level and the implementation level. While reference architectures exist for many mature software domains such as compilers and operating systems, we are not aware of any reference architectures proposed for web browsers.

*Correspondence to: Alan Grosskurth, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
†E-mail: agrossku@uwaterloo.ca
‡E-mail: migod@uwaterloo.ca

The web browser is perhaps the most widely used software application in history. It has evolved significantly over the past fifteen years; today, web browsers run on diverse types of hardware, from cell phones and tablet PCs to regular desktop computers. Web browsers are used to conduct billions of dollars worth of Internet-enabled commerce each year. A reference architecture for web browsers can help implementors to understand trade-offs when designing new systems, and can assist maintainers in understanding legacy code. Comparing the architecture of older systems with the reference architecture can provide insight into evolutionary trends occurring in the domain.

In this paper, we present a reference architecture for web browsers that has been derived from the source code of two existing open source systems and validate our findings against five additional systems. We explain how the evolutionary history of the web browser domain has influenced this reference architecture, and we identify underlying phenomena that can help to explain current trends. Although we present these observations in the context of web browsers, we believe many of our findings may represent more general evolutionary patterns which apply to other domains. This paper is organized as follows: the next section provides an overview of the web browser domain, outlining its history and evolution. We then describe the process and tools we used to develop a reference architecture for web browsers based on the source code of two existing open source systems. Next, we present this reference architecture and explain how it represents the commonalities of the two systems from which it was derived. We then provide validation for our reference architecture by showing how it maps onto the conceptual architectures of five additional systems. Finally, we summarize our observations about the web browser domain, discuss related work, and present conclusions.

THE WEB BROWSER DOMAIN

Overview

The World Wide Web (WWW) is a universal information space operating on top of the Internet. Each resource on the web is identified by a unique Uniform Resource Identifier[16] (URI). Resources can take many different forms, including documents, images, sound clips, or video clips. Documents are typically written using HyperText Markup Language[14] (HTML), which allows the author to embed hypertext links to other documents or different places in the same document. Data is typically transmitted via HyperText Transfer Protocol[15] (HTTP), a stateless and anonymous means of information exchange. A *web browser* is a program that retrieves documents from remote servers and displays them on screen, either within the browser window itself or by passing the document to an external helper application. It allows particular resources to be requested explicitly by URI, or implicitly by following embedded hyperlinks.

Although HTML itself is a relatively simple language for encoding web pages, other technologies may be used to improve the visual appearance and user experience. Cascading Style Sheets[3] (CSS) allow authors to add layout and style information to web pages without complicating the original structural markup. JavaScript, now standardized as ECMAScript[4], is a host environment for performing client-side computations. Scripting code is embedded within HTML documents, and the corresponding displayed page is the result of evaluating the JavaScript code and applying it to the static HTML constructs. Examples of JavaScript applications include changing element focus, altering page and image loading behaviour, and interpreting mouse actions. Finally, there are some types of content that the web browser cannot display directly, such as Macromedia Flash animations and Java applets. *Plugins*, small programs that connect with the browser, are used to embed these types of content in web pages.

In addition to retrieving and displaying documents, web browsers typically provide the user with other useful features. For example, most browsers keep track of recently visited web pages and provide a mechanism for "bookmarking" pages of interest. They may also store commonly entered form values as well as usernames and passwords. Finally, browsers often provide accessibility features to accommodate users with disabilities such as blindness and low vision, hearing loss, and motor impairments.

History and evolution

Although key concepts can be traced back to systems envisioned by Vannevar Bush in the 1940s and Ted Nelson in the 1960s, the WWW was first described in a proposal written by Tim Berners-Lee in 1990 at the European Nuclear Research Center (CERN)[13]. By 1991, he had written the first web browser, which was graphical and also served as an HTML editor. Around the same time, researchers at the University of Kansas had independently begun work on a text-only hypertext browser called Lynx; they adapted Lynx to support the web in 1993. In the same year, the National Center for Supercomputing Applications (NCSA) released a graphical web browser called Mosaic, which allowed users to view images directly interspersed with text. As the commercial potential of the web began to grow, NCSA founded an offshoot company called Spyglass to commercialize its technologies and Mosaic's co-author left to co-found his own company, Netscape. In 1994, Berners-Lee founded the World Wide Web Consortium (W3C) to guide the evolution of the web and promote interoperability among web technologies. In 1995, Microsoft released Internet Explorer (IE), based on code licensed from Spyglass, igniting a period of intense competition with Netscape known as the "browser wars." Microsoft eventually dominated the market, and Netscape released its browser as open source under the name Mozilla in 1998. Figure 1 shows a timeline of the various releases of several prominent web browsers.

Since 1998, a large number of Mozilla variations have appeared, reusing the browser core but offering alternative design decisions for user-level features. Firefox is a standalone browser with a streamlined user interface, eliminating Mozilla's integrated mail, news, and chat clients. Galeon is a browser for the GNOME desktop environment[5] that integrates with other GNOME applications and technologies. The open source Konqueror browser has also been reused: Apple has integrated its core subsystems into its OS X web browser, Safari, and Apple's modifications have in turn been reused by other browsers. Internet Explorer's closed source engine has also seen reuse: Maxthon, Avant, and NetCaptor each provide additional features to IE such as tabbed browsing and ad-blocking. Although each browser engine typically produces a similar result, there can be differences as to how web pages look and behave; Netscape 8, based on Firefox, allows the user to switch between IE-based rendering and Mozilla-based rendering on the fly.

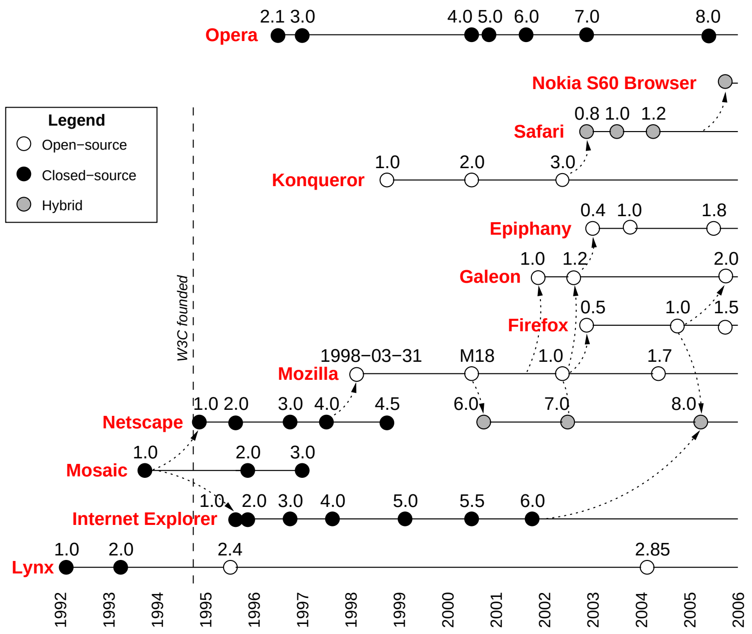

Figure 1. Web browser timeline

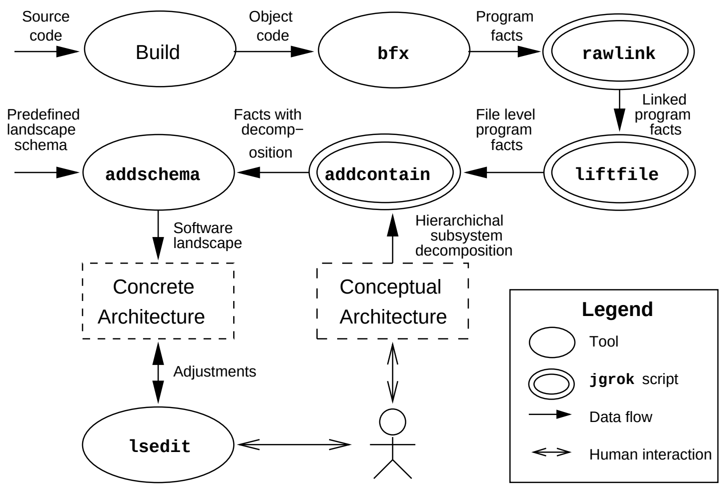

Figure 2. Extraction process for concrete architecture

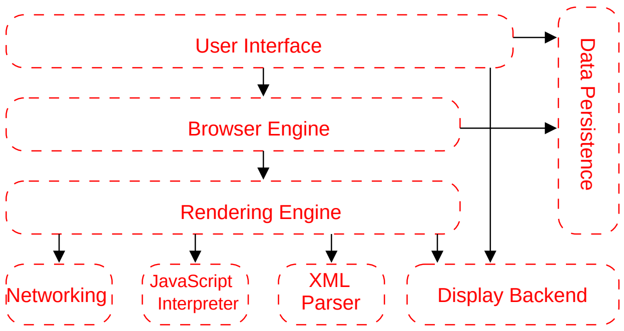

Figure 3. Reference architecture for web browsers

DERIVING A REFERENCE ARCHITECTURE

Using the source code and available documentation for two mature web browser implementations, we derived a reference architecture for the web browser domain. We used a process similar to that described by Hassan and Holt[26]. For each browser, a conceptual architecture was proposed based on domain knowledge and available documentation. The concrete architecture of each system was then extracted from its source code using the QLDX[8] reverse engineering toolkit, and this concrete architecture was then used to refine the conceptual architecture. A reference architecture was then proposed based on the common structure of these refined architectures, and it was validated against five other browser implementations.

A REFERENCE ARCHITECTURE FOR WEB BROWSERS

The reference architecture we derived is shown in Figure 3. It comprises eight major subsystems plus the dependencies between them:

1. The *User Interface* subsystem is the layer between the user and the *Browser Engine*. It provides features such as toolbars, visual page-load progress, smart download handling, preferences, and printing. It may be integrated with the desktop environment to provide browser session management or communication with other desktop applications.
2. The *Browser Engine* subsystem is an embeddable component that provides a high-level interface to the *Rendering Engine*. It loads a given URI and supports primitive browsing actions such as forward, back, and reload. It provides hooks for viewing various aspects of the browsing session such as current page load progress and JavaScript alerts. It also allows the querying and manipulation of *Rendering Engine* settings.
3. The *Rendering Engine* subsystem produces a visual representation for a given URI. It is capable of displaying HTML and Extensible Markup Language (XML) documents,

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.