A Reference Architecture for Web Browsers

Alan Grosskurth and Michael W. Godfrey
Software Architecture Group (SWAG)
School of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1 CANADA
{agrossku,migod}@uwaterloo.ca

Abstract

A reference architecture for a domain captures the fundamental subsystems common to systems of that domain as well as the relationships between these subsystems. Having a reference architecture available 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 re-engineering existing ones.

In this paper, we examine the history of the web browser domain and identify several underlying phenomena 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 two additional implementations. Finally, we discuss our observations about this domain and its evolutionary history; in particular, we note that the significant reuse of open source components among different browsers and the emergence of extensive web standards have caused the browsers to exhibit "convergent evolution."

Keywords: *Software architecture, reference architecture, software evolution, component reuse, web browser.*

1 Introduction

A *reference architecture*[27] for a domain captures the fundamental subsystems and relationships that are common to the existing systems in that domain. It aids in the understanding of these systems, some of which may not have their own specific architectural document. It also serve as a template for designing new systems, 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, no reference architecture has been proposed yet for web browsers.

The web browser is perhaps the most widely-used software application in history. Although the domain is only fifteen years old, it has evolved significantly over that time. It provides business and home users with convenient access to a wealth of information and services. Internet-enabled commerce currently accounts for billions of dollars worth of annual sales and is constantly growing.

The requirements for web browsers can differ significantly depending on the needs of the intended users. For example, handheld computer users typically want fast browsers with small memory footprints and streamlined user interfaces, while desktop users are often willing to trade-off some efficiency for additional features such as support for multiple languages. Additionally, web standards are constantly evolving, putting pressure on browsers to add support for the latest specifications. A reference architecture for web browsers can help implementors to understand trade-offs when designing new systems, and can also assist maintainers in understanding legacy code.

In this paper, we derive a reference architecture for web browsers from the source code of two existing open source systems and validate our findings against two 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 represent more general evolutionary patterns which apply to other domains.

This paper is organized as follows: Section 2 provides an overview of the web browser domain, outlining its history and evolution. Section 3 describes 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. Section 4 presents this reference architecture and explains how it represents the commonalities of the two sys-

tems from which it was derived. Section 5 provides validation for our reference architecture by showing how it maps onto the conceptual architectures of two other systems. Section 6 discusses related work, and Section 7 presents conclusions.

2 The Web Browser Domain

2.1 Overview

The World Wide Web (WWW) is a shared information system operating on top of the Internet. It consists of documents, as well as other types of files such as images and video clips, which are transmitted via HyperText Transfer Protocol (HTTP), a stateless and anonymous means of information exchange. A web browser is a program that retrieves these 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. Documents are written using HyperText Markup Language (HTML) and contain hyperlinks which point to other places in the same document, or in different documents. Every resource on the Web is identified by a unique Uniform Resource Identifier (URI). The web browser presents an interface to the user that 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[4] (CSS) allow authors to add layout and style information to web pages without complicating the original structural markup. JavaScript, now standardized as ECMAScript[7], 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 directly display, such as Macromedia Flash animations and Java applets. Plugins, small programs that connect with other programs, 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 accomodate users with disabilities such as blindness, low vision, hearing loss, and motor impairments.

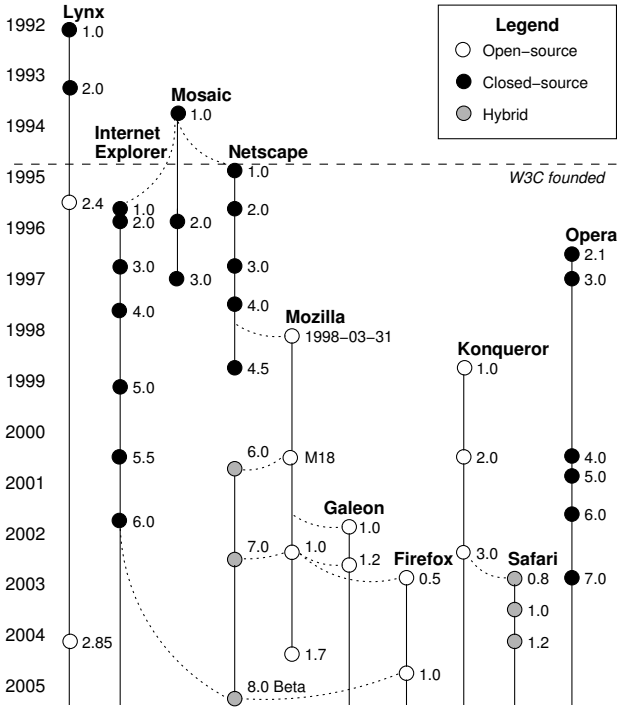

Figure 1. Web Browser Timeline

2.2 History and Evolution

The origins of the World Wide Web (WWW) can be traced back to a proposal written by Tim Berners-Lee in 1990 at the European Nuclear Research Center (CERN). A year later he had written the first web browser, which also served as a simple HTML editor. This browser was text-only and it allowed users to browse the web by typing in commmands that referenced numbered links. Around the same time, researchers at the University of Kansas had begun to work on their own hypertext browser called Lynx independent of the WWW. Lynx was also a text-only browser, but it allowed users to browse documents using navigational keys, rather than typing in commands. In 1993, Lynx was adapted to support the WWW, and it quickly became the preferred text-only browser. In 1995, it was released under a free software license. Figure 1 shows when the various releases of Lynx occurred, along with the releases of other popular browsers.

Also in 1993, the National Center for Supercomputing Applications (NCSA) released a web browser called Mosaic to the Internet community. As one of the first graphical browsers for the web, it allowed users to view images directly interspersed with text, as well as scroll through large documents. It had an easy point-and-click interface

that set a new standard for web browsers. During the same year, the first commercial Internet domain name was registered by Digital Equipment Corporation (DEC). As the commercial potential for the Internet began to grow, the University of Illinois forked off an offshoot called Spyglass to commercialize and support technologies developed at NCSA. Around the same time, the creator of Mosaic, Marc Andreeseen, left NCSA to co-found Netscape Communications Corp. The first version of their Netscape web browser was released in 1994. Netscape introduced the notion of continuous document streaming, which allowed users to view parts of documents as they were being downloaded, rather than wait for the entire download to finish. Also in 1994, the World Wide Web Consortium (W3C) was founded to promote interoperability among web technologies.

In 1995, Microsoft released the Windows 95 operating system. They included with it their own web browser, Internet Explorer (IE), which was based on code licensed from Spyglass. A period known as "browser wars" ensued, characterized by heated competition between Microsoft and Netscape. During this time, each browser introduced numerous innovations and proprietary enhancements to the web in an effort to attract more users. Although Netscape started off with over 90% market, Microsoft eventually took over the market, likely because their browser was included for free with Windows and could not be removed. In 1998, Netscape released most of the source code for the upcoming version of their browser under the in-house project name, Mozilla. Much of the code was rewritten, and eventually Mozilla released version 1.0 in 2002, which provided strong support for emerging web standards, such as CSS. Netscape now creates their browser by re-branding particular releases of Mozilla and adding in proprietary features. By this time, the closed source browser Opera[19] had also appeared, with its origins tracing back to a research project at the Norwegian telecom company, Telenor.

Since the rollout of Mozilla, a large number of variations have appeared. While the core of the browser remains the same, these variations offer alternative design decisions with respect to user-level features. Mozilla's emphasis on cross-platform support sacrifices tight integration with each particular platform; Galeon[8] remedies this by integrating with the Gnome Desktop Environment, and Camino[3] with Apple's Mac OS X. Mozilla's complex user interface and integrated mail client make it too cumbersome for some tastes; Firefox is a standalone browser developed by the Mozilla Foundation to provide lighter, more streamlined user interface. (In fact, Mozilla has recently been officially discontinued in favour of Firefox[15].)

In addition to Mozilla-based browsers, there are also several browsers based on Konqueror[10]. Developed for the K Desktop Environment (KDE), a user-friendly, open source desktop for UNIX-like systems, Konqueror is a file manager and universal document viewer as well as a web browser. In 2002, Apple used the core of Konqueror as the basis for their own OS X web browser, Safari. Although Safari is closed source, Apple has released their changes to the KHTML core engine back to the community as the open source Webcore[23] engine. This code has since been reused in several other OS X browsers, such as OmniWeb[18].

Finally, Internet Explorer's closed source core has been used as the basis for several closed-source Windows browsers: Maxthon[13], Avant[2], and NetCaptor[17] each provide features not found in IE such as tabbed browsing and advertisement blocking. More interestingly, version 8 of Netscape's browser is expected to be based on Firefox and support IE-based rendering as well as Mozilla-based rendering, allowing the user to switch between the two on the fly. Since there are subtle rendering differences between the engines, this feature may be useful because it would allow the user to choose the engine which produces the better results for a particular web site. Additionally, it may benefit web developers who need to test compatibility with both engines. However, it can be argued that including two different engines undermines the goal of web standards, which is to ensure web sites function identically across all engines.

2.3 Basic Usage

Although web browsers have evolved significantly since the early days of the web, their basic operation still remains relatively simple. A user begins by typing in a URI to view. Using HTTP, the browser sends a request to the appropriate remote web server for the document. The document is downloaded onto the user's computer, and a visual representation is rendered and displayed on the user's screen. If the document contains content other than basic HTML or Extensible Markup Language (XML), the browser may open a third-party application to display it. By clicking on hyperlinks in the document, the user can then navigate to other related documents, which will be requested and displayed by the web browser in a similar manner.

3 Deriving a Reference Architecture

Using the source code and available documentation for two different web browsers, we derived a reference architecture for the web browser domain. This reference architecture represents the abstract architecture of the domain, and was derived by following a process similar to that which is described by Hassan and Holt in [31]:

1. Two mature browser implementations were selected from which to derive the reference architecture.

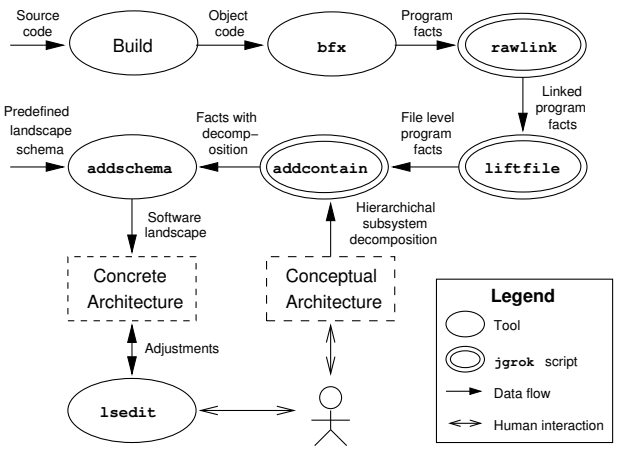

Figure 2. Extraction process for concrete architecture

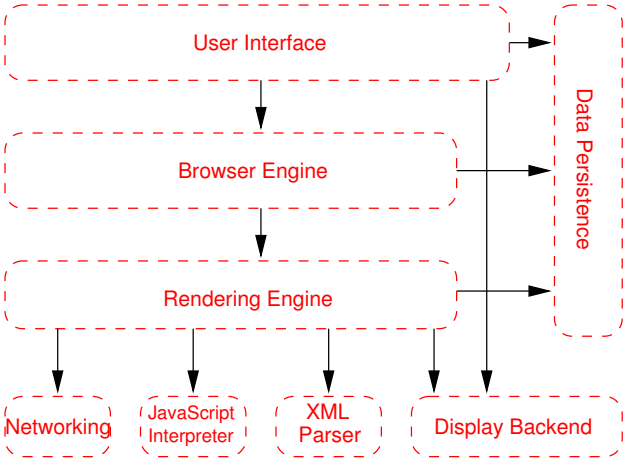

Figure 3. Reference architecture for web browsers

2. For each brower:
   (a) A conceptual architecture was proposed based on domain knowledge and available documentation.
   (b) The concrete architecture was extracted from the source code and used to refine the conceptual architecture.
3. A reference architecture was proposed based on the common structure of the conceptual architectures.
4. The reference architecture was validated against other browser implementations.

The two implementations chosen to serve as a basis for derivation were Mozilla and Konqueror. They were chosen because they are mature systems, have reasonably large developer communities and user bases, provide good support for web standards, and are entirely open source. Both Internet Explorer and Opera meet the first three requirements, but were not suitable for examination because they are closed source.

3.1 Extraction Methodology

The concrete architecture of each system was extracted from its source code using QLDX[20], a reverse engineering toolkit developed at the University of Waterloo for exploring and visualizing software architectures. The toolkit consists of `bfx`, a C and C++ fact extractor which operates on binary object files; `jgrok`, a relational calculator capable of manipulating these facts; and `lsedit`, a tool for viewing and editing software landscapes.

The process used to extract the concrete architecture is shown in Figure 2. First, the source code for the system was compiled into binary object code using the standard GNU toolchain. Next, the program facts were extracted using `bfx`. The program facts were then linked using a specialized `jgrok` script. Since the systems studied were relatively large, the relations between the entities were propagated from the function level to the file level, using another specialized `jgrok` script. Next, a hierarchichal subsystem decomposition was created based on the system's conceptual architecture. This containment structure was then applied to the file-level program facts, and a standard schema was added to produce the software landscape. This landscape respresents a preliminary version of the concrete architecture of the system, and it was explored and adjusted futher using `lsedit` to arrive at the final version.

The size of the extraction artifacts was at most within a factor of two of the size of the build artifacts, and typically much smaller. The extraction process was almost entirely automated; the only manual tasks were deriving the hierachichal subsystem decomposition and adjusting the landscape in `lsedit`. If the system was small or the directory structure of the source code corresponded well with the architectural structure, these steps did not require much effort. On the other hand, if the system was large and the architectural structure was not reflected in the directory structure, a significant amount of effort was involved in developing an accurate subsystem decomposition.

4 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 translates a URI into a visual representation. It is capable of displaying HTML and XML documents, optionally styled with CSS, as well as embedded content such as images. It is responsible for page layout and may contain "reflow" algorithms which incrementally adjust the position of elements on the page. This subsystem also includes the HTML parser.

4. The *Networking* subsystem implements file transfer protocols such as HTTP and FTP. It translates between different character sets, and resolves mime types for files. It may include a cache of recently retrieved resources.

5. The *JavaScript Interpreter* evaluates JavaScript (also known as ECMAScript) code, which may be embedded in web pages. JavaScript is an object-oriented scripting language developed by Netscape. Certain JavaScript functionality, such as the opening of popup windows, may be disabled by the *Browser Engine* or *Rendering Engine* for security purposes.

6. The *XML Parser* subsystem parses XML documents into a Document Object Model (DOM) tree. This is one of the most reusable subsystems in the architecture. In fact, almost all browser implementations leverage an existing *XML Parser*, rather than rewriting their own from scratch.

7. The *Display Backend* subsystem provides drawing and windowing primitives, a set of user interface widgets, and a set of fonts. It may be tied closely with the operating system.

8. The *Data Persistence* subsystem stores various data associated with the browsing session on disk. This may be high level data such as bookmarks or toolbar locations, or it might be lower level data such as cookies, cache, or security certificates.

The reader may wonder why we have placed the HTML parser within the rendering engine subsystem, while isolating the XML parser in a subsystem of its own. The answer is because although arguably less important to the functionality of the system, the XML parser is a generic, reusable component with a standard, well-defined interface. This is in contrast to the HTML parser, which is often tighly integrated with the rendering engine for performance reasons, and can provide varying levels of support for broken or non-standard HTML. That is, this *design decision* seemed to be a common feature of web browser architectures.

4.1 Mozilla

The Mozilla Suite[16] is one of the most prominent and widely-used open source projects today. It was started in 1998 when Netscape Communications released the source code for the development version of their popular Netscape Communicator product on the Internet under a free software licence. Now, almost seven years later, most of that system has been completely redesigned and rewritten, and a large number of new features have been added. Mozilla was written with several design goals in mind: support for web standards as well as broken web pages, support for multiple platforms, and fast rendering. We examined version 1.7.3, which consists of approximately 2,400 kLOC. Most of the source code is written in C++ although large parts of the user interface are written in JavaScript and a small amount of legacy code is written in C. We built and extracted the Linux version of Mozilla which uses the GTK toolkit.

The mapping of Mozilla's conceptual architecture onto the reference architecture is shown in Figure 4. We note the following observations about Mozilla's architecture:

- The *User Interface* subsystem is split into two subsystems: the XPFE toolkit and the actual user interface. The reason for this is that Mozilla reuses the XPFE toolkit as a basis for the user interfaces of other applications in the Mozilla suite including the mail/news client and the HTML editor.

- All data persistence is provided by Mozilla's profile mechanism, which is responsible for storing 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. This is likely because it contains more functionality; for example, it is responsible for rendering the application's user interface and well as web pages.

- The *Rendering Engine* and *Browser Engine* subsystems are tightly coupled to each other. As a result,

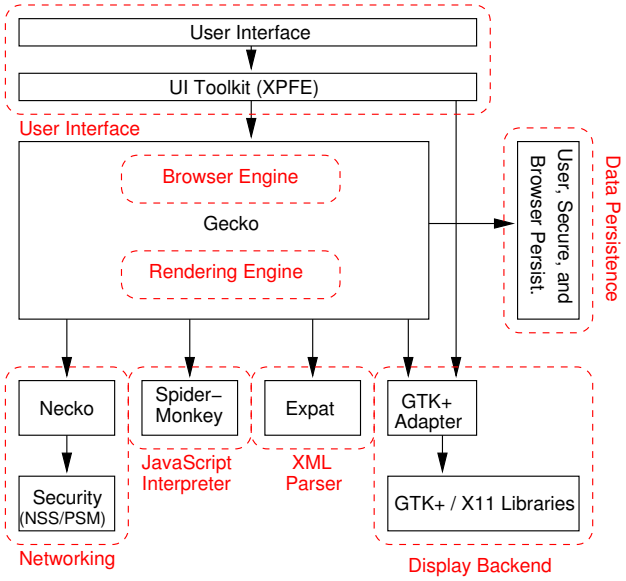

Figure 4. Architecture of Mozilla

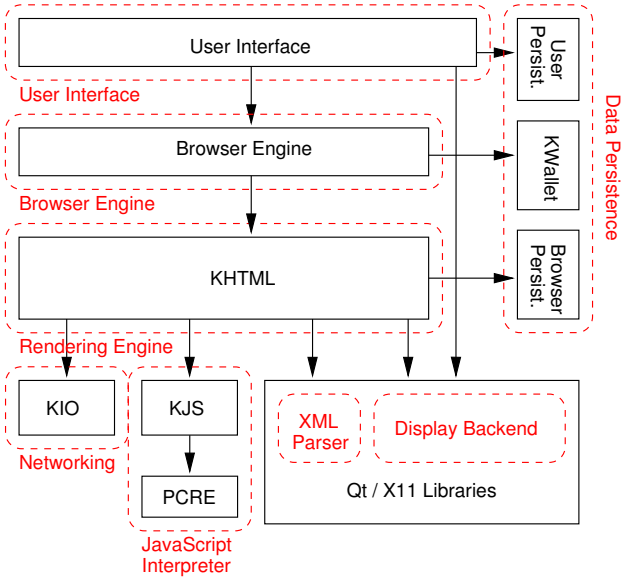

Figure 5. Architecture of Konqueror

it would be difficult to reuse the *Rendering Engine* by itself.

- All graphical elements in the user interface and web pages are specified in Extensible User Interface Language (XUL), which abstracts away the details of different platform-specific display and widget libraries. XUL is then mapped onto these each of these libraries using specially written adapter components. This architecture distinguishes Mozilla from other browsers in which the user platform-specific display and widget libraries are used directly, and allows Mozilla to be ported to different platforms with minimal difficulty.

4.2 Konqueror

Konqueror[10] is the official web browser of the K Desktop Environment (KDE)[9]. It can also serve as a file manager and a general-purpose file viewer. The project was started in January 1999, and its main design goals are speed, standards-compliance, and integration with KDE. We examined release 3.3.2, which consists of approximately 613 kLOC, including the required KDE libraries. Konqueror is written entirely in C++, as is most of the code in KDE.

We found Konqueror's codebase to be extremely well organized. Modules were split up cleanly into subdirectories and there was often a concise design document included with the code explaining the main abstractions and design decisions. This may be in part due to the extensive documentaion provided by the KDE Quality Team that details various design guidelines and best practices for KDE application development. This group also makes a conscious effort to involve nondevelopers with areas such as documentation, user interface design, issue tracking, and testing.

The mapping of Konqueror's conceptual architecture onto the reference architecture is shown in Figure 5. Konqueror makes extensive use of various KDE libraries: KHTML performs parsing, layout, and rendering of web pages; KJS interprets embedded JavaScript code; KWallet stores data such as passwords, cookies, and form data with strong encryption and error detection; and KIO is an asynchronous virtual file system which automatically provides encoding and decoding over common protocols. We note the following observations about the conceptual-to-reference architecture mapping:

- The *XML Parser* and *Display Backend* subsystems are both provided by the Qt[21] toolkit, which serves as the basis for all KDE applications. That is, these subsystems are external to the browser itself.

- The Perl Compatible Regular Expressions (PCRE) library is used as a backend for the regular expression functionality of the *JavaScript Interpreter*. PCRE is a mature and well tested component used in many other high-profile open source projects including Python and Apache.

- *Data Persistence* is provided at three levels. First, some high-level data such as bookmarks and history are stored by Konqueror itself. Second, other high-level data such as form completions are stored by KHTML. Third, secure data such as passwords are stored by KWallet, which allows this data to be shared with other KDE applications.

Overall, we found that Konqueror's developers have made a concious effort to implement the browser on top of existing libraries which take care of difficult tasks. In contrast, Mozilla has developed almost all these libraries in-house, delegating only to other libraries only when necessary. A consequence of this is that Konqueror is closely tied to UNIX-like operating systems and the Qt toolkit, while Mozilla supports several different operating systems and display toolkits. However, as we will see in the next section, Apple was able to adapt Konqueror to their own needs by removing many of its dependencies.

5 Validating the Reference Architecture

Two additional implementations were chosen against which to validate the reference architecture: Lynx and Safari. Lynx was chosen because it is the oldest web browser still regularly used and maintained. Safari was chosen because it represents an interesting mix of open and closed source technology, and was developed with usability as a key design goal.

5.1 Lynx

Lynx[12] is a text-only web browser for use on cursor-addressable, character cell terminals. Its history dates back to before the age of the World Wide Web and HTML; it began as an interface for an "organization-wide information system."[6] Hypertext capabilities were then added, complete with its own link syntax and URI scheme. It next evolved to support the Gopher protocol and distributed hypertext, functioning also as a database interface. The wwwlib library, which provided the first support for WWW protocols, was later grafted on making Lynx into a true web browser. We examined release 2.8.5 of Lynx, which consists of approximately 122 kLOC.

Lynx's age and development process are the primary reasons why its codebase is so large and complex. Although it was developed by a single student through its early stages, we found that its diverse and constantly changing requirements resulted in a system composed of small fragments of code with no coherent overall structure. In addition, much of the code is very low-level and is specific to either the UNIX or VMS platform, which increases the overall complexity. To its credit, however, Lynx still remains among the most popular console browsers on UNIX-based systems.

The mapping of Lynx's conceptual architecture onto the reference architecture is shown in Figure 6. Lynx uses libwww, which provides a wide variety of functionality such as HTML parsing and support for both the HTTP and FTP protocols. The libgnutls library provides optional support for secure protocols. Lynx also uses the curses library for displaying information on character-cell terminals. Lynx's

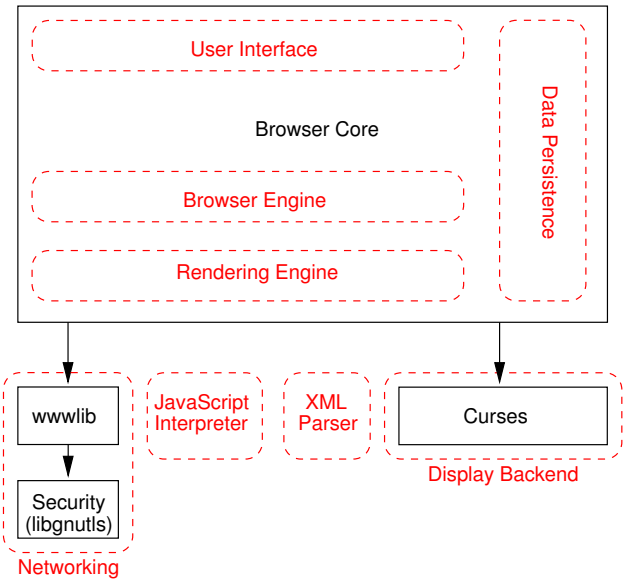

Figure 6. Architecture of Lynx

conceptual architecture shows a clear separation between three main subsystems: browser core, networking, and display backend. We note the following observations about the conceptual-to-reference architecture mapping:

- There is no clear separation between the *User Interface*, *Browsing Engine*, *Rendering Engine*, and *Data Persistence* subsystems. This is likely because these subsystems are much simpler in Lynx than in other browsers due to its text-only nature. For example, the rendering engine outputs web pages in linear form rather than attempting to layout elements at appropriate coordinates, and the user interface relies solely on keyboard input rather than dealing with menus, widgets, and mouse events.

- Lynx does not contain a *JavaScript Interpreter* subsystem or an *XML Parser* subsystem. This is because the majority of Lynx's codebase was written before JavaScript existed, and no one has since volunteered to add support for it. As a result, Lynx cannot be used to browse web sites that rely on JavaScript for normal interaction. However, many sites only use JavaScript to augment functionality provided by HTML, so Lynx users can still use these sites, albeit with decreased functionality.

Overall, the lack of modularity and text-only nature of Lynx make its conceptual architecture much simpler than our reference architecture. However, we are still able to identify three core subsystems which correspond to some of the subsystems in the reference architecture.

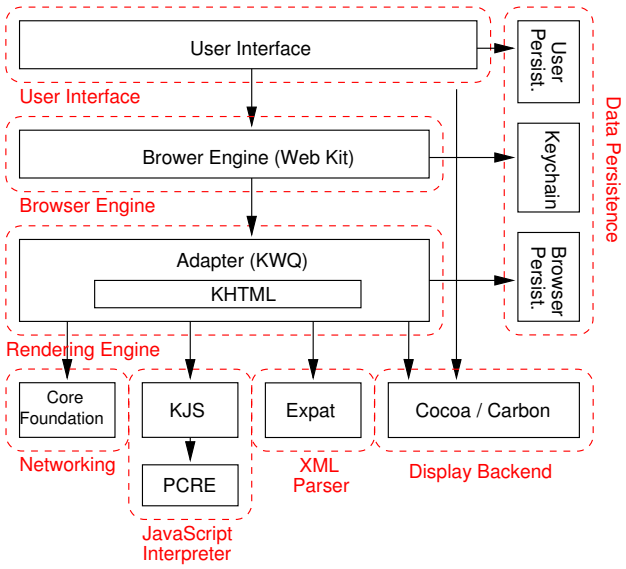

Figure 7. Architecture of Safari

5.2 Safari

Safari[22] is a web browser developed by Apple Computer for its Mac OS X operating system. The first version was released in January 2003. The main design goals for Safari are usability, speed, standards-compliance, and integration with OS X. Safari reuses the KHTML rendering engine and the KJS JavaScript interpreter from the KDE project. Their modified version is called WebCore, and is released under the GNU Lesser General Public License (LGPL). However, the rest of Safari's code is proprietary, including the browser engine (WebKit) and the user interface. We examined the source code of release 125 of WebCore and JavaScriptCore, which consists of 114 kLOC of C++ code and 22 kLOC of Objective C++. Since we could not extract the proprietary parts, their structure was inferred from Apple's developer documentation[1].

The conceptual-to-reference architecture mapping for Safari is shown in Figure 7 We note the following observations about Safari's archtiecture:

- The *Rendering Engine* is composed of the KHTML core engine wrapped in the KWQ adapter. KWQ is written in Objective C++, which allows it to present an Objective C API to KHTML, which is written in C++. This was needed for integrating Safari into OS X.
- *Networking* functionality is provided by OS X's Core Foundation networking library, used in place of KIO.
- The *XML Parser* subsystem is provided by the Expat XML parser, used in place of the XML parser provided by the Qt toolkit.
- The *Display Backend* subsystem is composed of two complementary libraries: Carbon and Cocoa. Carbon provides a lower-level C API for display routines, while Cocoa provides a higher-level Objective C API.
- Persistent data is handled by three separate system-wide services that are built into OS X: Preferences, Keychains, and Caches. The use of these services allows Safari to to integrate smoothly with other OS X applications.

Overall, Safari's conceptual architecture corresponds well with our reference architecture. Safari reuses the core engine from Konqueror, substitues a Mac OS X look and feel, and makes use of other components and libraries native to OS X in place of the Linux- and KDE-specific components of Konqueror.

5.3 Summary

There are several reasons why a web browser's architecture would differ from our reference architecture. Some of the subsystems in the reference architecture may be implemented as a single subsystem for simplicity, while others may implemented across multiple subsystems for greater flexibility. Furthermore, new subsystems may be added to provide additional capabilities not found in traditional web browsers, while other subsystems may be omitted to make the browser more lightweight.

Lynx's conceptual architecture is much simpler than our reference architecture. Some subsystems are missing because they correspond to relatively modern features which either are not applicable to text-only browsers, or simply are not supported yet in Lynx. Other subsystems are tighly coupled as a result of Lynx's overall lack of modularity.

Safari's conceptual architecture corresponds quite closely to our reference architecture. This makes sense because Safari is based on the same rendering engine and JavaScript interpreter as Konqueror; furthermore, it seems as though Apple has used Konqueror as a blueprint for Safari, substituting OS X technologies for the corresponding KDE technologies. Additionally, we observe that Safari uses the Expat XML parser, which is also found in Mozilla.

Table 1 shows various statistics about the different web browsers studied. We note the following observations:

- Konqueror achieves nearly the same degree of standards-compliance as Mozilla with one-quarter of the amount of code. This may be due to the fact that Mozilla supports many different platforms, while Konqueror only supports UNIX-like systems running X11 with the Qt toolkit.
- Lynx, while smaller than the other browsers, is nonetheless very large for a text-based browser. For

Table 1. Approximate web browser statistics

Project	Rel.	Lang.	Files	kLOC	Size*	Start
Mozilla	1.7.3	C++	10,500	2,400	29	1998
Konq.	3.3.2	C++	3,145	600	17	1996
Lynx	2.8.5	C	200	122	2.1	1992
Safari	1.2	C++, Obj C	>750	>136	>2.1	2003

*Represents the compressed tarball size in megabytes.

comparisions sake, Links[11], a more recent text-only browser with a comparable feature set, consists of only 26 kLOC, approximately one-fifth the size of Lynx. This may be due to the large amount of legacy code in Lynx.

- We are unable to obtain complete size information for Safari because a large portion of the code is closed source. The numbers shown correspond only to the WebCore engine, and thus represent a lower-bound on the total size.

We are currently investigating how the conceptual architectures of the Mosaic[14], Dillo[5], and Galeon[8] web browsers correspond to our reference architecture. We would also like to examine web browsers designed specifically for embedded devices, but at the present time we do not know of any mature open source implementations.

6 Related Work

There has been some previous research involving reference architectures. Eixelsberger has recovered a reference architecture from a family of embedded, real-time train control systems, each around 150 kLOC[27]. He used a formal Architectural Description Langugage (ADL) to describe each system, and then performed commonality analysis. Batory, Coglianese, Goodwin, and Shafer have defined a reference architecture for avionics as part of a project to build a domain-specific software architecture (DSSA) environment for assisting the development of avionics software.[24]. Hassan and Holt have defined a reference architecture for web servers, and shown how it maps to the conceptual architectures of three systems[31].

A product line architecture specifies the architecture for a group of products sharing a common, managed set of features[25, 26]. Product line architectures are similar to reference architectures, although they generally represent a group of systems intended to be produced by a single organization, while reference architectures represent the entire spectrum of systems in a domain.

Finally, there have been some previous case studies examining various aspects of Mozilla's architecture and development process. Godfrey and Lee have extracted Mozilla's architecture as part of a study investigating data exchange between different reverse engineering tools[30]. Mockus, Fielding, and Herbsleb have used Mozilla as part of a case study of open source software projects[32]. Fischer, Pinzger, and Gall have analyzed the proximity of features in Mozilla based on data in its bug-tracking database, Bugzilla[28].

7 Conclusions

We have examined the history and evolution of the web browser domain, developed a reference architecture for web browsers based on two existing implementations, and validated this reference architecture by mapping it onto two additional implementations. Furthermore, we have observed several interesting evolutionary phenomena while studying web browsers; namely, emergent domain boundaries, convergent evolution, and tension between open and closed source development approaches.

As the web browser domain has evolved, its conceptual boundaries—both external and internal—have become increasingly more defined. However, there are still discrepancies as to the nature of these boundaries. For example, Microsoft has claimed that Internet Explorer is a fundamental part of the Windows operating systems, providing rendering functionality to other applications such as help browsers and wizards. This extended boundary posed a problem for third-party browsers such as Netscape who sought to compete with IE. In a similar example, we have seen email and usenet client functionality integrated with the web browser starting with Netscape, and continuing with the Mozilla Suite. This integration has potentially made it more difficult for external clients to compete. Further examples of domain integration include FTP clients and local file managers. It will be interesting to observe how the web browser domain adapts to support embedded devices, such as cell phones and PDAs; these platforms often have limited amounts of memory, making it undesirable to have multiple competing applications installed at once.

The large amount of effort devoted to creating high-quality open source browser implementations has had a tremendous influence on the domain. During the "browser wars," core browser components included proprietary extensions in order to attract customers. Today, increased standardization and pressure to comply with these standards has led to reuse of core browser components. Rather than duplicate effort, browsers often attempt to differentiate themselves by providing interface enhancements; however, these features seem to be easily duplicated. For example, after tabbed browsing was pioneered by NetCaptor, it quickly began appearing in other browsers such as Opera and Mozilla. Similarly, popup blocking and automatic web form filling are now commonplace, suggesting

that web browser domain is exhibiting a form of *convergent evolution*[29].

The availability of mature browser components has also resulted in tension betwen open and closed source development approaches. The Mozilla project was founded with the intention of creating a mature, open source browser platfrom that could be used as the basis for other browsers, both open and closed. Indeed, the last two releases of Netscape have been based on Mozilla and have been closed source. A similar situation has occurred with Apple's Safari, which is a closed source browser based on Konqueror's open source engine. Although not required by the licence, Apple has voluntarily contributed their changes to open source code back to the community. Conversely, Internet Explorer represents a closed source browser component that can potentially be embedded in an otherwise open source product. Interstingly enough, the upcoming version of Netscape promises to embed both the Mozilla and IE engines, allowing users to switch on the fly.

While we have seen applications composed of both open and closed source components before, the interaction usually takes place on the perimeter, as is the case with closed source binary modules for the Linux kernel. We believe the heterogeneous combination of core open and closed source software components within individual systems makes the web browser domain unique and interesting.

Acknowledgements

We thank Ali Echihabi for his contributions to an earlier project out which this paper has grown, as well as Ric Holt for his feedback and advice.

References

[1] Apple developer documentation. `http://developer.apple.com/documentation`.

[2] Avant web browser home page. `http://www.avantbrowser.com`.

[3] Camino web browser home page. `http://caminobrowser.org`.

[4] Cascading Style Sheets home page. `http://www.w3.org/Style/CSS`.

[5] Dillo web browser home page. `http://dillo.org`.

[6] An early history of Lynx. `http://www.cc.ku.edu/~grobe/early-lynx.html`.

[7] ECMAScript language specification. `http://www.ecma-international.org/publications/standards/Ecma-262.htm`.

[8] Galeon web browser home page. `http://galeon.sourceforge.net`.

[9] K Desktop Environment home page. `http://kde.org`.

[10] Konqueror web browser home page. `http://konqueror.org`.

[11] Links web browser home page. `http://links.sourceforge.net`.

[12] Lynx web browser home page. `http://lynx.isc.org`.

[13] Maxthon web browser home page. `http://www.maxthon.com`.

[14] Mosaic web browser home page. `http://archive.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html`.

[15] Mozilla application suite transition plan. `http://www.mozilla.org/seamonkey-transition.html`.

[16] Mozilla project home page. `http://www.mozilla.org`.

[17] Netcaptor web browser home page. `http://www.netcaptor.com`.

[18] Omniweb web browser home page. `http://www.omnigroup.com/applications/omniweb`.

[19] Opera web browser home page. `http://www.opera.com`.

[20] QLDX reverse engineering toolkit home page. `http://swag.uwaterloo.ca/qldx`.

[21] Qt application development framework home page. `http://www.trolltech.com/products/qt`.

[22] Safari web browser home page. `www.apple.com/safari`.

[23] Webcore framework home page. `http://developer.apple.com/darwin/projects/webcore`.

[24] D. Batory, L. Coglianese, M. Goodwin, and S. Shafer. Creating reference architectures: an example from avionics. In *Proceedings of the 1995 Symposium on Software Reusability (SSR '95)*, pages 27–37, 1995.

[25] J. Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[26] P. Clements and L. M. Northrop. *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[27] W. Eixelsberger, M. Ogris, H. Gall, and B. Bellay. Software architecture recovery of a program family. In *Proceedings of the 20th International Conference on Software Engineering (ICSE '98)*, pages 508–511, 1998.

[28] M. Fischer, M. Pinzger, and H. Gall. Analyzing and relating bug report data for feature tracking. In *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE '03)*, pages 90–99, 2003.

[29] D. J. Futuyma. *Evolutionary Biology*. Sinauer Associates, Sunderland, MA, USA, 3rd edition, 1998.

[30] M. Godfrey and E. H. S. Lee. Secrets from the monster: Extracting mozilla's software architecture. In *Second International Symposium on Constructing Software Engineering Tools (CoSET '00)*, June 2000.

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

[32] A. Mockus, R. T. Fielding, and J. Herbsleb. A case study of open source software development: the apache server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE '00)*, pages 263–272, 2000.