Conceptual Architecture of Firefox

Andre Campos - 0230481 - drick@uvic.ca
Bryan Lane - 0434698 - blane@uvic.ca
Neal Clark - 0429078 - nclark@uvic.ca
Sunpreet Jassal - 0323709 - ssjassal@uvic.ca
Stephen Hitchner - 0430473 - hitchner@uvic.ca

June 2, 2007

Contents

List of Figures

Glossary

AOM	Application Object Model
API	Application Programming Interface
CSS	Cascading Style Sheet
DOM	Document Object Model
Firefox	Web browser created by the Mozilla Foundation
GTK+	aka GIMP toolkit, one of the two most popular widget toolkits for XWindows
HTML	Hyper Text Markup Language
kLOC	thousand lines of code
MFC	Microsoft Foundation Class library
MIME	Multipurpose Internet Mail Extensions
Mozilla	original public name of Mozilla Application Suite (aka SeaMonkey), created by the Mozilla Foundation
Necko	Networking Library of Firefox
NGLayout	Browser Engine of Firefox
NSPR	Netscape Portable Runtime. NSPR is a platform abstraction library, presenting a uniform interface to Firefox, no matter what platform it's running on.
OS	Operating System
OS X	an Apple operating system
OSI	Open Systems Interconnection. A general network reference model
PKCS	Public Key Cryptography Standard
PKI	Public Key Infrastructure
SSL	Secure Socket Layer. A cryptography protocol
TLS	Transport Layer Security. A cryptography protocol that is the successor of SSL
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

WINAPI	Windows Application Programming Interface
X.509	Standard formats for public key certificates and certification path validation algorithm
X11	aka XWindows. Networking and display protocol that provides a standard toolkit to build GUIs on *NIX.
XML	Extensible Markup Language
XP	Cross platform, as in XPCOM, XPFE, XPInstall, XPIDL.
XPCOM	Cross Platform COM (component object model) implementation
XUL	XML User interface Language

1 Introduction

Firefox is a product that is released by the Mozilla Foundation. Firefox is written in C/C++ and contains over 2,400 kLOC. The conceptual architecture is similar to most modern web browsers. The application is comprised of several independent components layered together to form what is commonly called a layered architecture.

A layered architecture is used to simply the design and maintainability of the code. Like the OSI model, the conceptual architecture provides interfaces for higher level components to interact with lower level components. This allows each component to be built in isolation, as long as each components adheres to a defined interface. The layered architecture promotes code reuse by allowing other applications the ability to easily include individual components (Epiphany, the official web browser of the GNOME project uses the Mozilla Engine but provides different user interface). As the software application evolves components can be swapped out for new implementations without affecting the components in layer above or below.

A reference architecture diagram of the Firefox architecture is depicted in Figure 1.

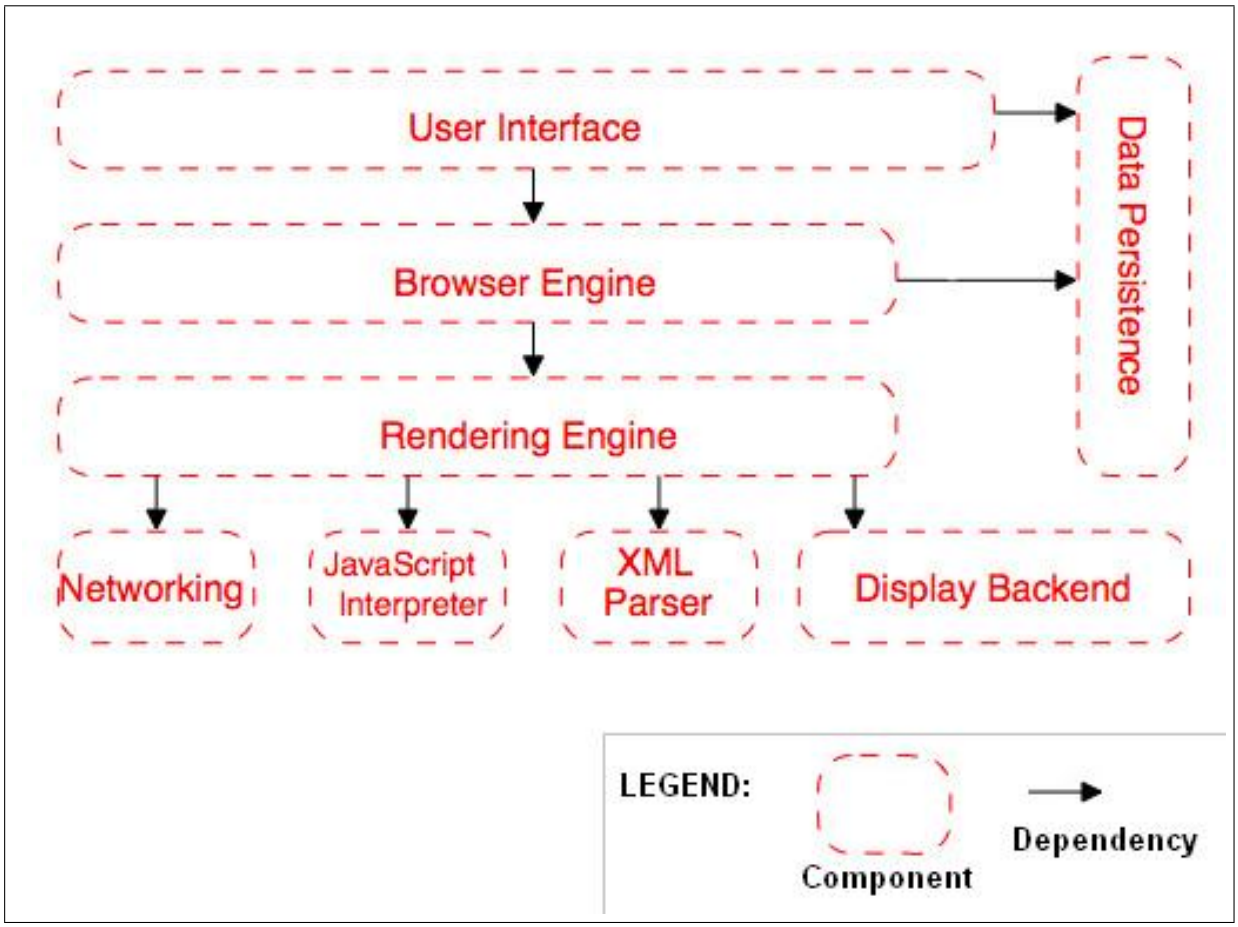

Figure 1: Reference architecture of common browsers including Firefox (derived from [1]).

Each conceptual component is introduced briefly below. The Firefox implementation of each component is discussed in detail in later sections.

- The User Interface component (Section 3.1) provides the methods with which a user interacts with the Browser Engine. The User Interface provides standard features (preferences, printing, downloading, toolbars) users expect when dealing with a desktop application.
- The Browser Engine component (Section 3.2) provides a high-level interface to the Rendering Engine. The Browser Engine provides methods to initiate the loading of a 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 (Section 3.3) produces the visual representation of a given URL. The Rendering Engine interprets the HTML, 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 (Section 3.5) provides functionality to handle retrieve URLs using the common Internet protocols of HTTP and FTP. The Networking components handles 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 (Section 3.6) component executes the JavaScript code that is embedded in a website. Results of the execution a passed to the Rendering Engine for display. The Rendering Engine may disable various actions based on user defined properties.
- The XML Parser component (Section 3.7) is used to parse XML documents.
- The Display Backend component (Section 3.8) 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 (Section 3.4) manages user data such as bookmarks and preferences.

Figure 2 depicts the overall architecture of Firefox with details about how each individual component is implemented. The implementations will be discussed below.

2 Evolution

Firefox is an open-source project that is managed by the Mozilla Foundation. Each component is divided into sub-modules. Each of these modules is owned by a specific individual that is in charge of managing the development of that that module. Mozilla lists all module owners on their website [5]. Individuals who wish to help with the development join a team responsible for a given module. The module owner is required to review and approve all changes for a given module.

The Mozilla Foundation oversees the overall architecture of Firefox and provides best-practices documentation. The foundation manages all the module owners and as the application evolves

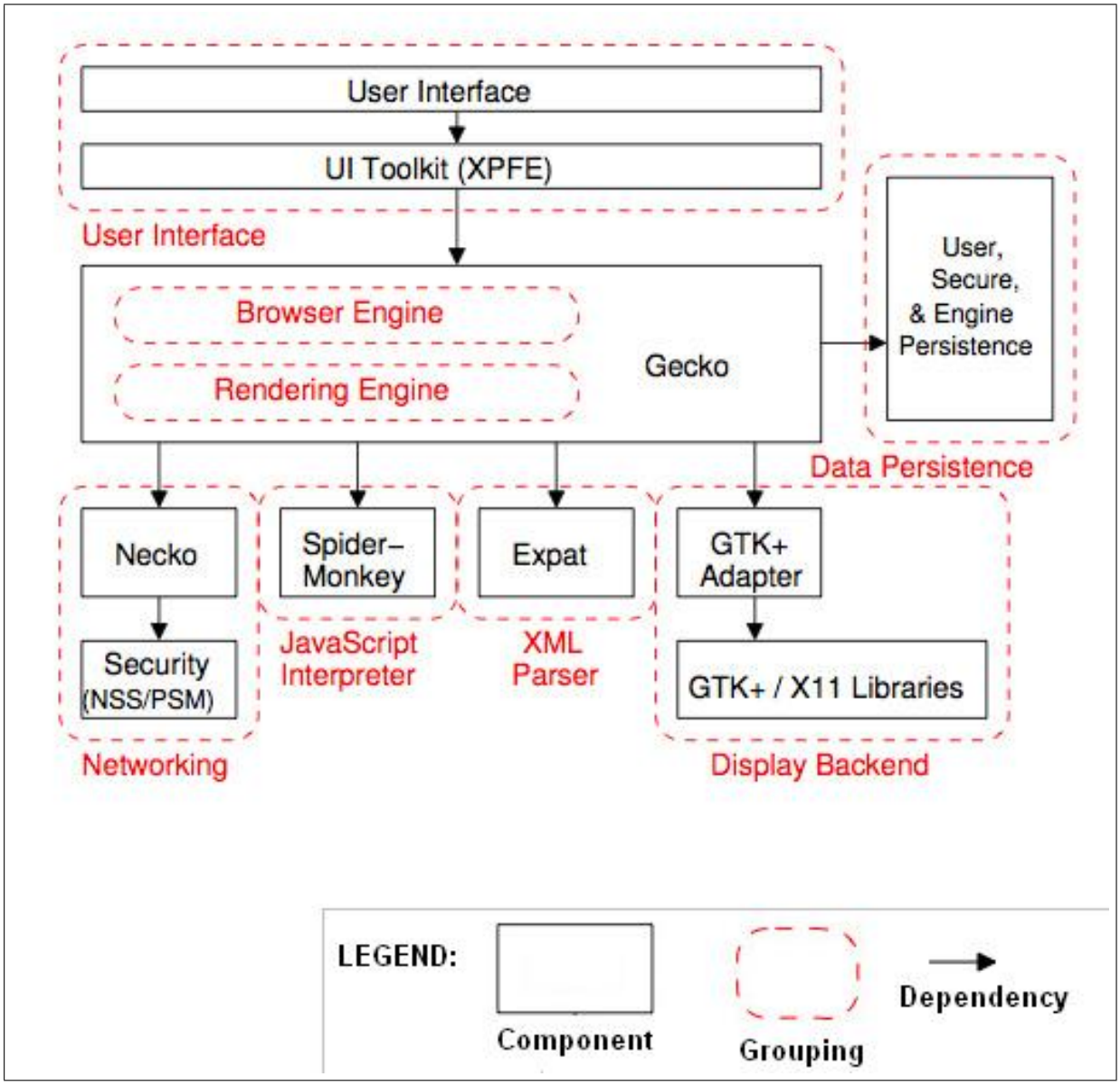

Figure 2: Detailed component architecture of Firefox. Different Display Backend components are used depending on the host OS (from [1]).

directs the module owners about what changes and features need to be added. To maintain a high standard, several levels of code reviews are required before code is added into the main Firefox distribution. Module owners review code that contributed to individual modules and may require additional code reviews from other peers depending if the change affects other modules. "Super-reviews" are used to ensure all patches fit properly with the Firebox code base as a whole; these reviews are done by developers that are in charge of the Firefox architecture.

3 Components

3.1 User Interface

The XPToolkit is a collection of loosely related facilities, from which application writers can pick and choose, which provide a platform independent API to some commonly exploited platform-specific machinery, e.g., bringing up a dialog. Not all platform independent facilities fall under the XPToolkit. JavaScript, for example, is a distinct service. Not all the platform specific implementation details can be forced into the XPToolkit. Applications will still contain platform specific code; though they can minimize the amount by exploiting the XPToolkit [14].

Starting with the top-down approach, if XPToolkit is the go-to component in terms of UI and Gecko is the layout engine, how do they interact? XPToolkit uses the XML User Interface Language (XUL) to define UI components, namely widgets. Since Gecko already provides the facility to parse XUL documents (Gecko actually delegates the XML parsing to the XML Parser component) into an object model, there is no need for duplication; XPToolkit uses Gecko to parse XUL into the Application Object Model (AOM).

Once XPToolkit has the AOM, it generates two layers (as shown in Figure 3 below): frames and widgets, and services. The former is responsible for the look-and-feel of the user interface while the latter is in charge of implementing the services, such as printing. These two layers are completely independent; the only way the communicate is through the AOM, who is responsible for the message-passing back and forth.

3.2 Browser Engine

The Firefox 2 Browser Engine or NGLayout module is responsible for coordinating the various other modules download, parse, and apply CSS styles [9]. Once the DOM has been loaded the content is passed off to the rendering engine to be drawn to the display device. The NGLayout component uses the following sub-components:

- Layout: the layout sub-component is responsible for managing the layout of the DOM for rendering. The Layout sub-system coordinates the Style and DOM components and facilitates the layout of the rendered DOM into a format supported by the graphics sub-system [6].
- Rendering: the rendering sub-component is responsible for translating the loaded DOM and Style information into graphics primitives and then drawing the processed content to the display device.
- Style: the style sub-component is responsible for applying styles to the parsed html. The Style subsystem modifies the DOM to apply CSS1, CSS2, HTML 3.0 and limited HTML 4.0 styles to the parsed HTML document.
- DOM: the DOM (Document Object Model) is a interface or facade which provides an abstraction layer above the underlying components representing web site content. The DOM also provides bindings for JavaScript, Python, C++ and XPCOM. The Firefox DOM fully supports the W3C Level 1 DOM standard and already implements many of the more important W3C Level 2 DOM standards.

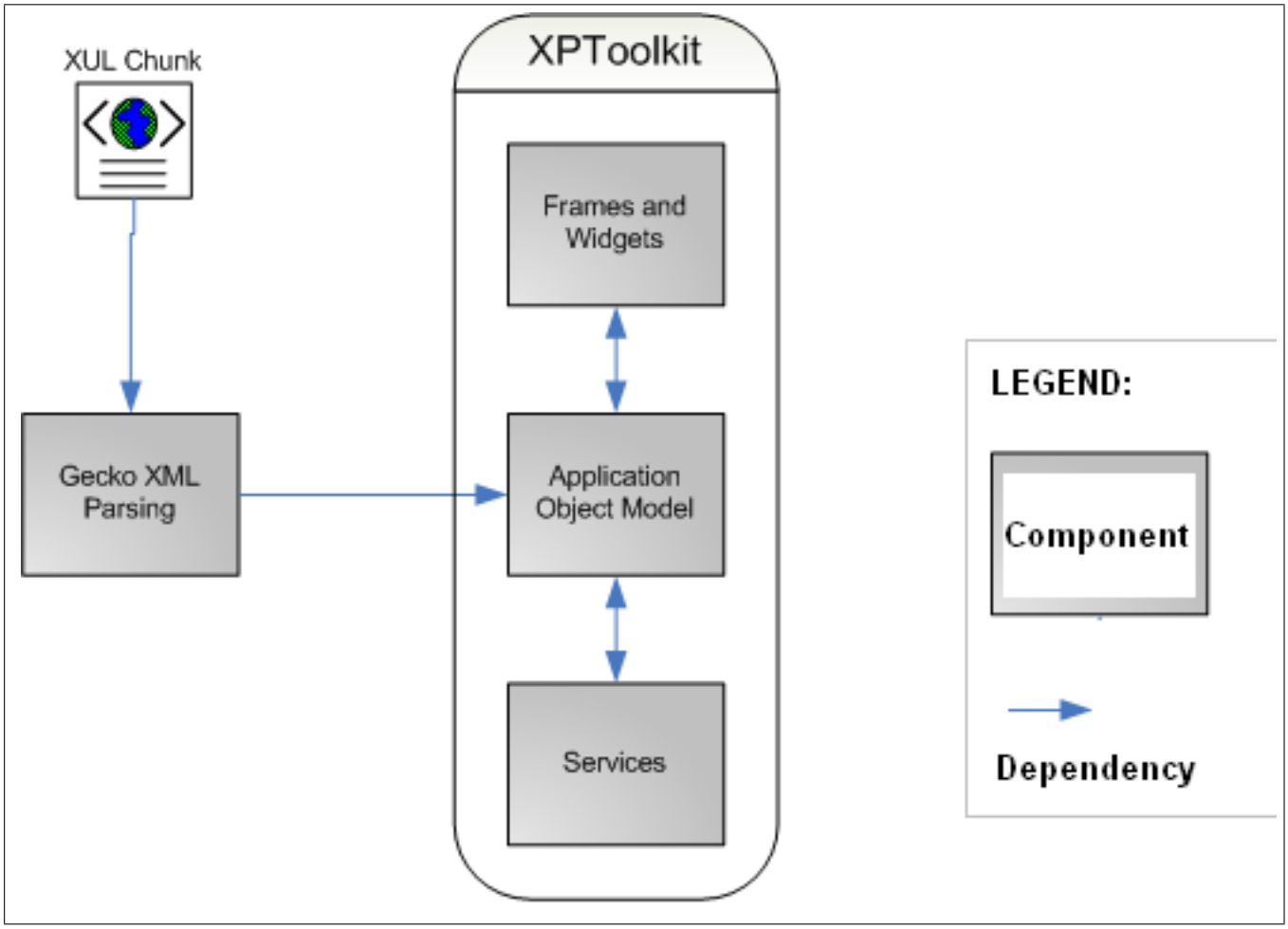

Figure 3: XPToolkit Architecture

- HTML Parser: The html parser sub-component is responsible for parsing incoming content and loading the DOM. The Parser is fully backwards compatible with Navigator 4.x and Internet Explorer and provides limited DTD support.

The NGLayout component communicates with the configuration system to control how pages are loaded and to configure other the functionality that NGLayout provides.

3.3 Rendering Engine

The Rendering engine is a sub-system of NGLayout and uses a set of platform independent and platform dependant data structures to render web content to the screen. Layout is controlled by Gecko's modern CSS2 implementation and is rendered using the Mark II rendering method whereby the rendering engine uses placeholders for anticipated content that has not yet arrived. The placeholders are then replaced as content becomes available thereby allowing for fast rendering of pages.

The rendering engine is responsible for drawing the user interface and any web content to the screen. The rendering engine renders content based on a set of platform independent

data structures including: geometrical primitives, colour definitions, font specifications, image abstraction, rendering context, device context, font metrics, image, regions, and alpha blender. The rendering engine uses a copy of the current DOM to render web pages using the graphics primitives listed above,

3.4 Data Persistence

This layer can be split into two components: Browser persistence, and user data; they are completely independent.

Firefox uses mozStorage to persist browser information, which includes settings for extensions and Firefox components. *mozStorage* is a database API built on top of sqlite [13]. Since it is only available to trusted callers, web pages do not have access to it. There is rudimentary support for transactions and limitations when it comes to concurrency. The bottom line is that mozStorage is not a heavy-duty database system and should not be treated like one. It essentially interacts with Gecko (Firefox native components) and the extensions.

Firefox persists user data through cookies, DOM Storage, and Flash Local Storage. Cookies are widely used on the internet, but are somewhat limited. DOM Storage is an alternative introduced by Web Applications 1.0 that allow for persisting more data in a secure fashion. Finally, Flash Local Storage requires an external plug-in. In any event, the user persistence component interacts with both Gecko and XPToolkit.

3.5 Networking

The networking module is really two components, the network library (Necko, previously known as Netlib) and the network security (NSS and PSM).

Necko is a platform-independent API that handles multiple layers of the networking model (Network, Session and Presentation layers). Although part of the Mozilla codebase, it is possible to create your own network client using this API. Necko depends on XPCOM and NSPR [7].

Necko provides a variety of services. The nsIIOService is the main Necko service that manages protocol handlers and creates URI objects from URI strings. The nsIRUI service provides URI implementations for different protocols, including accessor functions. The nsIChannel represents a single-use logical connection to a resource (protocol implementation dependent). The nsIProtocolHandler service manages a protocol, creates the nsIRUI object, and creates the nsIChannel. The nsIStreamListener is passed to the nsIChannel and has functions related to managing download requests. The nsITransport service represents the physical connection, and contains both synchronous and asynchronous I/O methods to be used by different protocols.

NSS (Network Security Services) is a set of libraries used to supply Firefox with security-enabled communications [8] [10]. NSS is built on NSPR (Netscape Portable Runtime) and supports a variety of security standards and certificates including:

- SSL v2 and v3
- TLS v1 (RFC 2246)

- PKCS #s 1, 3, 5, 7, 8, 9, 10, 11 and 12
- X.509 v3
- S/MIME (RFC 2311 and RFC 2633)

PSM (Personal Security Manager) is a set of libraries that perform many standard PKI functions such as setting up SSL connections and verifying signatures. PSM is built on NSS [11].

3.6 JavaScript Interpreter

SpiderMonkey is Mozilla's JavaScript interpreter, implemented in C. Formerly known as JS-Ref (JavaScript Reference), SpiderMonkey builds a DLL that contains the JavaScript runtime elements, and compiles an interpreter program that is linked to the library to run scripts [12, 4].

SpiderMonkey has no dependencies on any other part of the system.

3.7 XML Parser

Firefox stands on four main pillars: XUL, an XML dialect used to construct user interfaces; JavaScript, a scripting language; RDF, an XML dialect used to store data; and, XPCOM, an object discovery and management system [3]. A parser decoupled from a particular dialect is highly desirable in order to parse structured data and to maintain the parser code effectively. Additionally, Firefox supports many other XML technologies recommended by the W3C, including XHTML, SVG, MathML and XLink.

Firefox uses an existing XML Parser, Expat, to read, write and transform XML documents to and from a variety of XML dialects mentioned above. As a result, the parser is one of the most reusable components in the architecture. The parser typically returns a DOM tree which the client code can traverse to extract the appropriate information. The Rendering Engine uses the XML Parser component to parse XML, XUL, SVG and other dialects of XML. The resulting tree data structures are then used for further processing in the rest of Firefox.

3.8 Display Backend

The Display Backend is tightly coupled with the host operating system. The Display Backend provides a coherent, cross-platform, interface used by the User Interface, for all drawing and windowing functionality provided by the various operating systems that are supported by Firefox. The interface is made up of many different widgets that are used by the User Interface.

The Display Backend is divided into two sub-components: an OS specific graphics adapter and the OS graphics libraries. The adapter translates internal calls from the User Interface into calls that are understood by the host OS's graphics libraries. For each host operating system there is a different implementation that uses the native graphics libraries provided by the OS. For UNIX type operating systems Firefox implements a graphics adapter that uses the GTK+/X11 libraries. For the Mac OSX version Firefox implements a graphics adapter that uses the native Cocoa framework. For the Windows version Firefox implements a graphics adapter using the WINAPI and MFC system calls.

3.9 Extensibility

Unlike Mozilla, Firefox comes with a powerful extension capability, which allows extensions to alter the behaviour of the browser at various stages of the architecture. New themes for skinning can be added. New XML tags to the XUL language and even new XPCOM objects can be added using extensions [3]. As Figure 4 shows, the extensions are included in the same layer as the User Interface layer but they interact with the Persistence engine (where they are stored) and the User Interface engine (which allows the user to access some extensions). Some extensions run without a UI, so not all extensions interact with the User Interface layer.

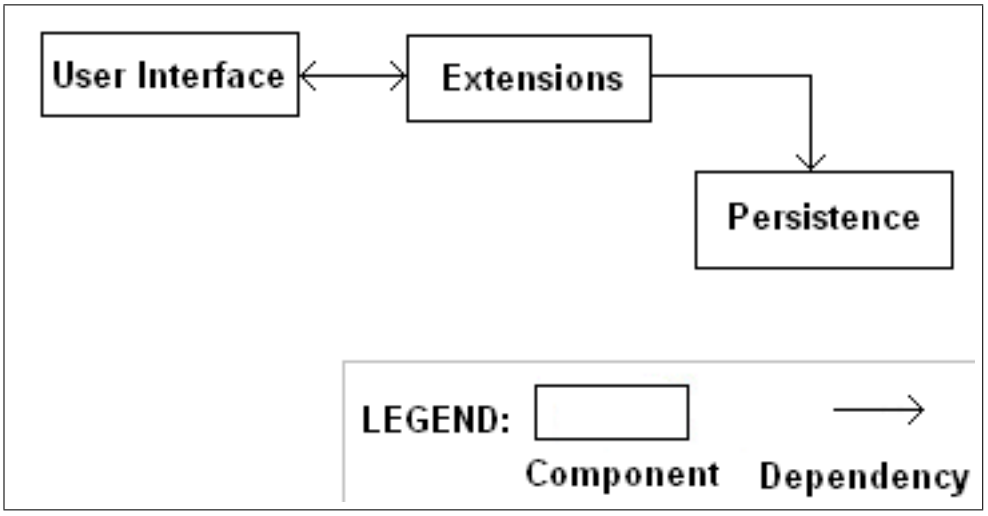

Figure 4: Extension component interactions

Extensions, however, depend on the interfaces exposed by Firefox web browser. A new version of Firefox can potentially change those interfaces making some extensions incompatible. So extension writers have to regularly update their extensions to maintain compatibility with newer versions of the browser.

4 Conclusion

Firefox web browser's architecture resembles a layered architecture. Components are allowed to interact only with a specific layer underneath. For instance, the User Interface is only allowed to talk to the Gecko browser engine. This architecture affords ease of maintainability and reusability as the number of components affected by a change in the code base is limited.

Examining the conceptual architecture by browsing through various documents, we found that it corresponds closely to the reference architecture proposed in [2]. The User Interface layer, in the reference architecture, comprises of UI toolkit and User Interface layer of the conceptual architecture. The Browser Engine and Rendering Engine support is offered by a single component in Firefox, Gecko. In addition, Gecko provides a parser for HTML, for efficiency reasons. The networking component is comprised of Necko (networking library) and NSS/PSM for security. Spidermonkey provides an interpreter for JavaScript language. Expat provides support for XML parsing. Display component is platform dependent and therefore, uses GTK+/X11 on UNIX flavors and MFC on Windows.

References

[1] Alan Grosskurth and Michael Godfrey. A case study in architectural analysis: The evolution of the modern web browser. *EMSE*, 2007.

[2] Allan Grosskurth and Michael Godfrey. A reference architecture for web browsers. *In Journal of Software Maintenence and Evolution: Research and Practice*, pages 1–7, 2006.

[3] Nigel McFarlane. *Rapid Application Development with Mozilla*, pages 11–23. Prentice Hall, 2003.

[4] mozilla.org. JavaScript Reference Implementation (JSRef) README. http://lxr.mozilla.org/mozilla/source/js/src/README.html.

[5] mozilla.org. Module Owners. http://www.mozilla.org/owners.html.

[6] mozilla.org. Mozilla Layout Engine. http://www.mozilla.org/newlayout/.

[7] mozilla.org. Necko Interfaces Primer. http://www.mozilla.org/projects/netlib/necko_interface_overview.html.

[8] mozilla.org. Networking Library Documentation. http://www.mozilla.org/projects/netlib/.

[9] mozilla.org. NGLayout architecture. http://www.mozilla.org/newlayout/overview.html.

[10] mozilla.org. Overview of NSS. http://www.mozilla.org/projects/security/pki/nss/overview.html.

[11] mozilla.org. Personal Security Manager (PSM). http://www.mozilla.org/projects/security/pki/psm/.

[12] mozilla.org. SpiderMonkey (JavaScript-C) Engine. http://www.mozilla.org/js/spidermonkey.

[13] mozilla.org. Storage - MDC. http://developer.mozilla.org/en/docs/Storage.

[14] mozilla.org. XPToolkit architecture. http://www.mozilla.org/xpfe/aom/AOM.html.