Conceptual Architecture of Mozilla Firefox (version 2.0.0.3)

SEng 422 Assignment 1
Dr. Ahmed E. Hassan

June 1, 2007

Iris Lai
Jared Haines
John,Chun-Hung,Chiu
Josh Fairhead

Table of Contents

List of Figures

Abstract

Mozilla Firefox is an open source web browser developed by Mozilla Corporation. It has rich browsing features such as tabbed browsing and extensions. This report surveys the conceptual architecture of Mozilla Firefox version 2.0.0.3 as the first deliverable for the SEng 422 term project. We developed a conceptual architecture for Mozilla Firefox based on the reference architecture of Mozilla Browser. This report shows the subsystems of Mozilla Firefox, the components of each subsystem, and the relationships between them.

Introduction

The intent of this report is to discuss the conceptual architecture of Mozilla Firefox 2.0.0.3. Firefox is an open source cross-platform web browser developed by the Mozilla Corporation. Firefox has rich web browsing features which include Tabbed Browsing, Spell Checking, Search Suggestions, Session Restore, Web Feeds (RSS), Live Titles, Integrated Search, Live Bookmarks, Pop-up Blocker, Streamlined Interface, and Accessibility. Another important feature of Firefox is that it can be customized by extensions, themes, and advanced preferences. Firefox is built on top of Mozilla's application platform and reusable components. For example, the Firefox project is built on Gecko which is also used in other Mozilla projects, such as Camino.

The conceptual architecture of Mozilla Firefox is developed based on the reference architecture of Mozilla browser. The reference architecture of Mozilla browser (see Figure 1.) shows fundamental subsystems and relationships between them. Mozilla Firefox fundamental subsystems include User Interface, Browser Engine, Rendering Engine, Networking, XML Parser, JavaScript Interpreter, and Data Persistence. User Interface is the layer between the user and the browser. The web addresses are sent to the Browser Engine. Browser Engine and Rendering Engine read web contents which are written in languages like HTML, CSS, JavaScrip, and then render it and display it to the user.

In Firefox conceptual architecture, the layout engine is called Gecko. Gecko itself is a browser engine, as well as a rendering engine. It talks to other components, such as networking, XML Parser, and so on. Figure 1 shows that the Display Backend is composed of GTK+ Adapter and GTK+ / X11 Libraries. This is a specific Mozilla application which is running on Linux operating system. Mozilla Firefox is a cross-platform browser. There are extra components in the Display Backend subsystem which will be discussed later in this report.

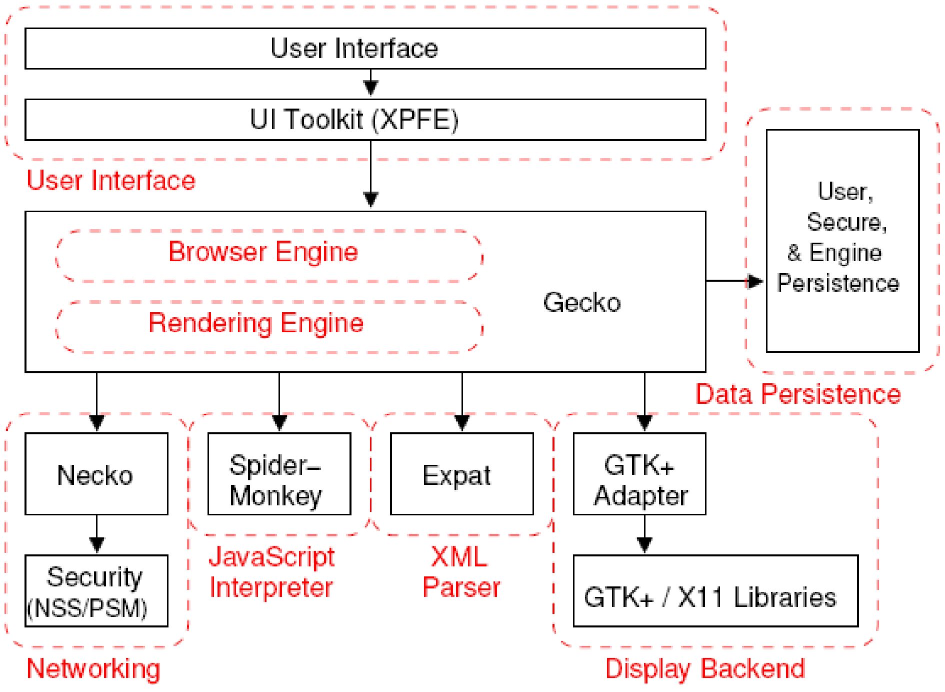

Figure 1: Reference Architecture of Mozilla Browser

Conceptual Architecture of Mozilla Firefox

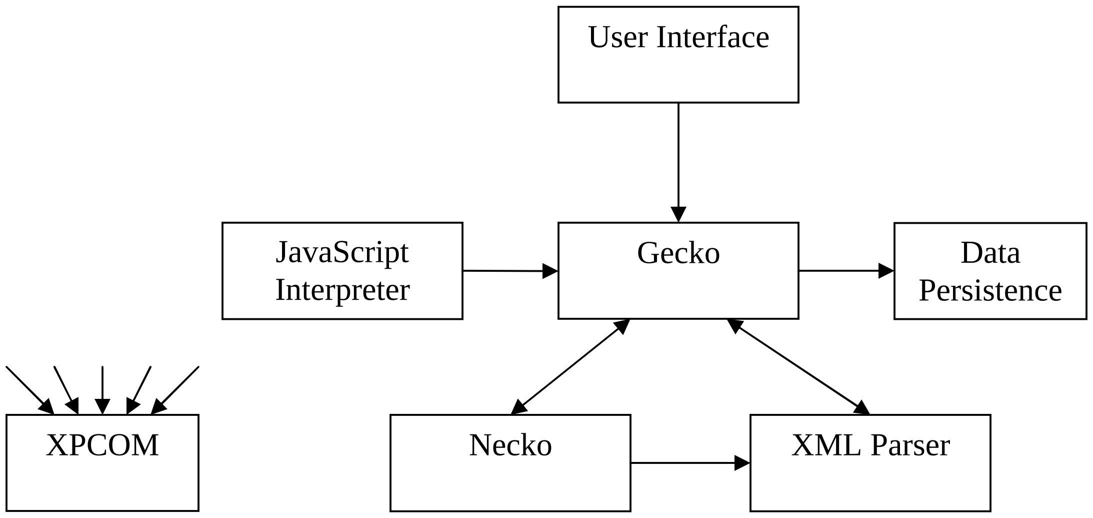

Figure 2: Conceptual Architecture of Mozilla Firefox

User Interface

The User Interface (UI) of the Mozilla browser (Firefox 2.0) is one major layer between the users and the browser/rendering engine (GECKO). It provides various features such as bookmarking web pages, setting internet preferences, visualizing web pages, and downloading files, etc.

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. Mozilla's Cross-Platform Front End (XPFE) is a development environment based upon XUL to make Mozilla applications like Firefox or Thunderbird.

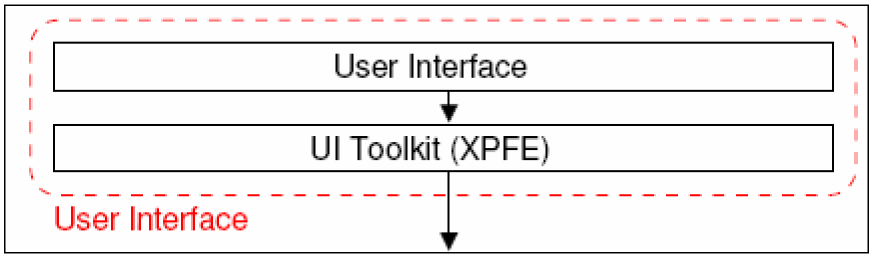

Figure 3: User Interface Conceptual Architecture

The whole XPFE framework is made up of the following parts:

1. XUL - xml language that describes the user interface
2. CSS – used for customizing or decorating the user interface
3. JavaScript – used for linking and programming the interface components
4. XBL(eXtensible Binding Language) – defines re-useable XUL components
5. XPCom/XPConnect – interfaces XPFE with binary code (C/C++, etc)
6. XUL templates – framework for importing data
7. RDF (resource description framework) – retrieves and stores data
8. DTD – foreign localization
9. XPInstall – standard installer for XPFE applications

Most components in Firefox's UI are created by using XUL and HTML 4.0 and are decorated by CSS1 and CSS2. XUL stands for XML User Interface Language which is supported by Gecko, the core browser/rendering engine of Firefox 2.0. XUL is a Mozilla's XML-based technology that allows users to customize the existing interface components or create different types of cross-platform user interfaces that connect or disconnect from the internet. (Mixture of HTML, CSS, JavaScript, and XML)

XUL provides users abilities to create the following user interface elements:

- Toolbars with buttons or other content
- Menus on a menu bar or pop up menus

- Tabbed dialogs
- Trees for hierarchical or tabular information
- Keyboard shortcuts

The User Interface interacts with Gecko, which is the heart of the Mozilla framework. Gecko is cross-platform browser/rendering engine that is used by Mozilla for interpreting and displaying web content. The user interface elements are created and defined in XUL and rendered by Gecko.

Gecko

Gecko is Firefox's rendering and layout engine. It is responsible for parsing and rendering HTML and XML documents, as well as the XUL-defined user interface. Gecko is comprised of the following components:

- **HTML Parsers:**
  This component is responsibleresponsible for parsing HTMLHTML documents.
- **Content Model:**
  The content model arranges parsed document data into a tree structure based on the Document Object Model (DOM). Calls to any of the DOM APIs will modify the Content Model.
- **Style System:**
  The Style System is responsible for parsing CSS data.
- **Image Loader:**
  This component loads image data.
- **Frame System:**
  This component places the Content Model's DOM elements into "frames," and uses the Style System's style information to calculate the size of each frame. This size information is used to arrange the frames into a new tree whose structure represents the visual layout of document. Items in the Frame Tree retain pointers to their corresponding items in the Content Model, so any change to the Content Model will also produce change in the Frame Tree.
- **Graphics Interface:**
  Gecko contains a platform-specific interface for instructing the native OS to draw information on the screen.

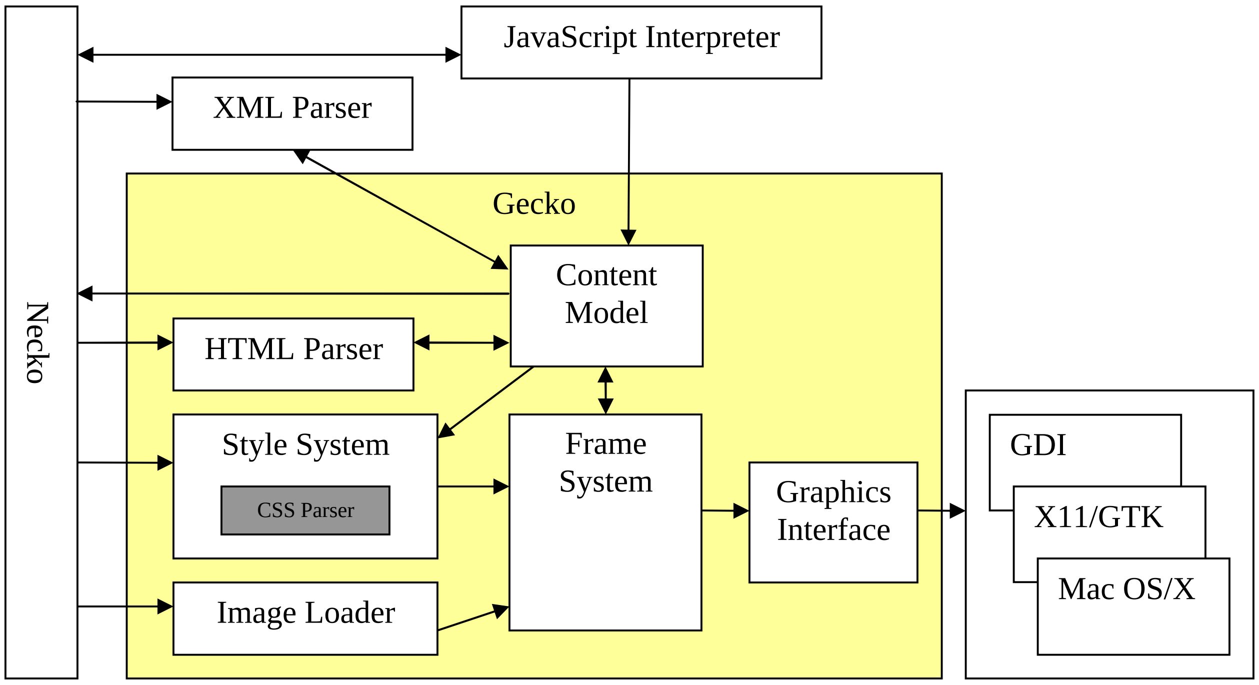

Figure 4: Gecko Conceptual Architecture

Data flow:

1. A Document Parser receives an HTML or XML document. It parses the document, and passes the data to the Content Model. Any CSS information is passed to the Style System.
2. The Content Model arranges the data into a DOM tree.
3. The Style System parses the CSS data.
4. The Image Loader loads any images that will be displayed in the document.
5. The Frame Constructor receives the style information, DOM tree, and image data, and constructs a Frame Tree.
6. The Frame Tree is passed to the OS via the Graphics Interface, and the document is drawn on the screen.

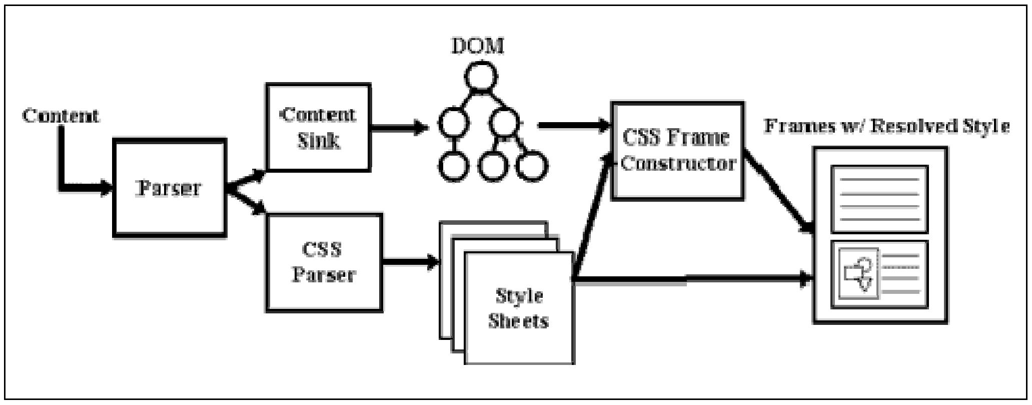

Figure 5: Data flow within Gecko

Necko

Necko is a networking library created by Mozilla. It provides a platform-independent API for the lower layers of the network stack. Necko uses the Mozilla Network Security Services (NSS) library for implementing secure network communications over SSL.
Necko Components:

- **Application –** The application that is using the API, in this case Firefox.
- **URL object –** A URL object used for defining a URL connection.
- **Network Service –** A global object that facilitates the construction and opening of URLs.
- **Protocol Handler –** There is a unique protocol handler for each protocol type that is supported.
- **Protocol Connection –** A protocol connection for a URL instance. A protocol connection provides any protocol-specific accessors required by the protocol.
- **File/Socket Transport –** Represents the physical connection.
- **NSS –** Network Security service is a set of libraries designed to support cross-platform development of communications applications that support SSL, S/MIME, and other Internet security standards.

Necko started out as the networking library known as netlib. Necko takes a more modular approach as apposed to netlib’s monolithic core. The main design goals if the Necko project is an improvement in the memory footprint, code maintainability and network performance from the old netlib architecture. The memory footprint of Necko is substantially smaller than netlib since the minimal amount of networking support will be loaded at any one time. The modular design will help improve the maintainability of Necko by tailoring the design to a more component based architecture. This architecture is well suited for an open source development environment since it allows individuals to easily contribute custom components. Necko will also include instrumentation mechanisms that will allow a quantitative analysis of performance.

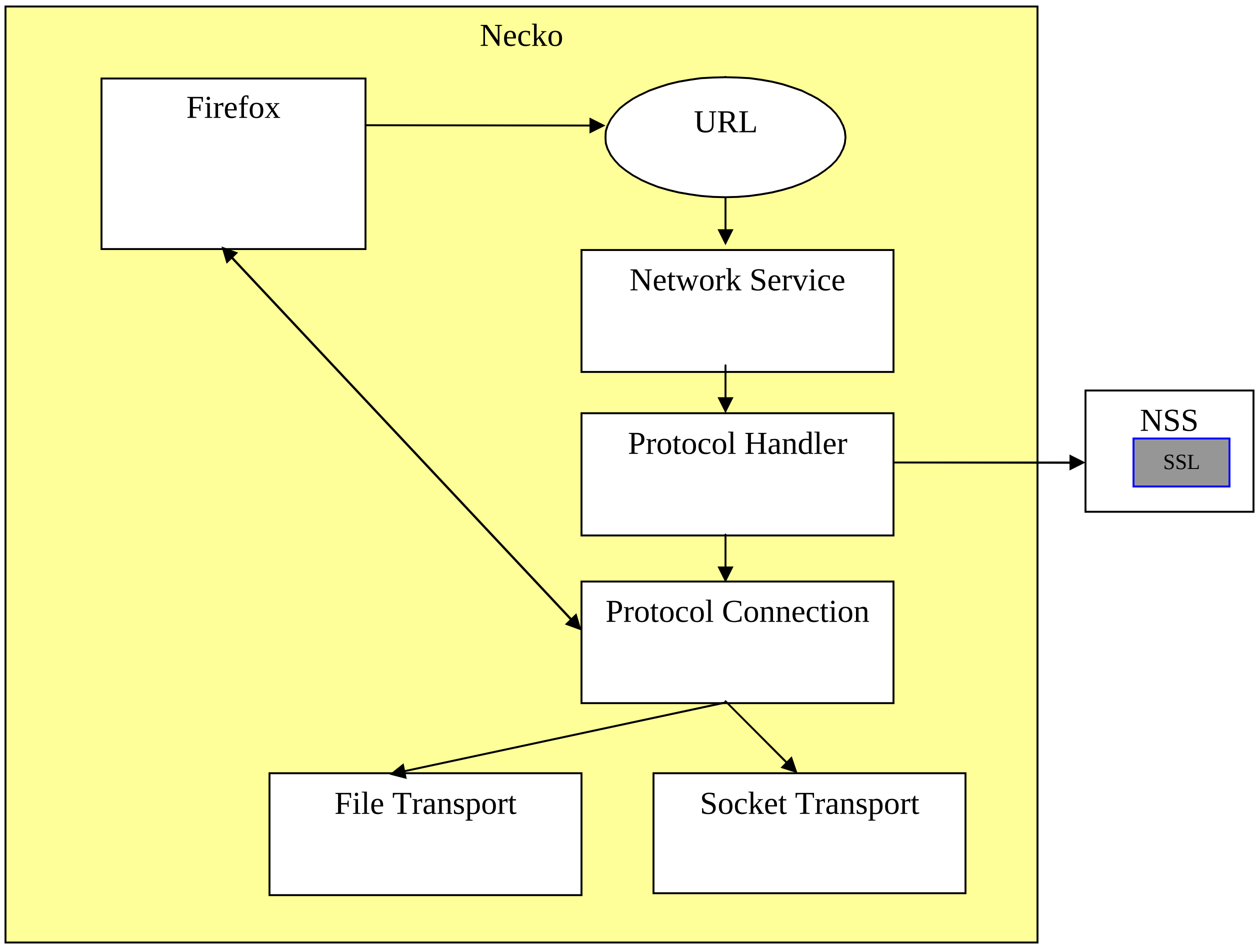

Figure 6: Necko Conceptual Architecture

Data flow:
The service manager within Firefox calls the network service with a URL object. The service manager then looks up a protocol handler that can handle the protocol specified within the given URL. If the protocol is SSL, then the NSS component handles the secure communication. The protocol handler then creates a protocol connection object for the URL. Once the connection is created, Firefox can then send/receive data from the connection. The protocol connection then sends the data to the file/socket transport that represents the physical connection.

Necko consists of the application thread and a maximum number of transport threads. When incoming data is received, a transport thread marshals the data to the application thread where it can be used. When a data transfer request is received, the application thread marshals the data to an appropriate transport thread. The data marshalling can be done in two ways – synchronous and asynchronous. With synchronous concurrency, the thread is blocked while waiting for incoming data from the transport and is used for I/O stream connections. With asynchronous concurrency, the application can share their thread between parsing the protocol and performing other activities.

The primary interfaces within Necko are not thread safe as most of Firefox runs on the main thread. However, most Necko protocols utilize a background thread for I/O operations. For some of the more complicated protocols, the handler needs to operate on the background thread for improving performance.

Each developer would need to ensure proper concurrent communication between their component and the application thread. There would be a need to ensure the proper internal implementation of protocols among various developers. Since many of the components are designed to work for an array of protocols, it would be wise for a single developer to develop the component for all protocols, as apposed to a single developer implementing their protocol over a set of components. A single developer working on all the protocols for a specific component will ensure that there is consistent and efficient development of the component. For example, it would be wise for a single developer to implement each different type of Protocol Handler.

XML Parser

The XML Parser is the component responsible for handling eXtensible Markup Language (XML) documents like XHTML, MathML, SVG, RDF, and XUL. In the implementation, the XML parser is based on Mozilla expat parser, and it is included in Gecko. At the conceptual level, it has a specific function which could be separated from the rendering engine and browser engine in Gecko.

The XML Parser supports the following document types:

- **XHTML (eXtensible HyperText Markup Language)**
  HTML, reformatted in XML, in order to be used in conjunction with other languages like MathML and SVG.
- **MathML**
  An implementation of the W3C's Mathematical Markup Language.
- **SVG**
  SVG stands for Scalable Vector Graphics, and that it is an XML language for sophisticated 2-dimensional graphics. SVG is to graphics what XHTML is to text, MathML is to mathematical equations and CML is to the description of chemical molecules.
- **RDF**
  RDF, or Resource Description Framework, is a W3C standard for representing meta-information. In Mozilla, we're using this to aggregate and display information about all kinds of Internet resources, including email, UseNet news, site maps, bookmarks, and browser history.
- **XUL**
  XUL is Mozilla's XML-based user interface language that lets you build feature rich cross-platform applications that can run connected to or disconnected from the Internet. These applications are easily customized with alternative text, graphics, and layout so they can be readily branded or localized for various markets.

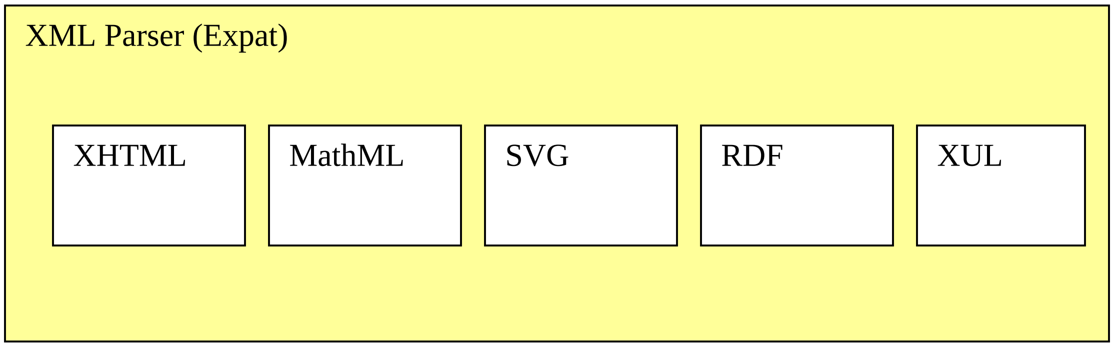

Figure 7: XML Parsers

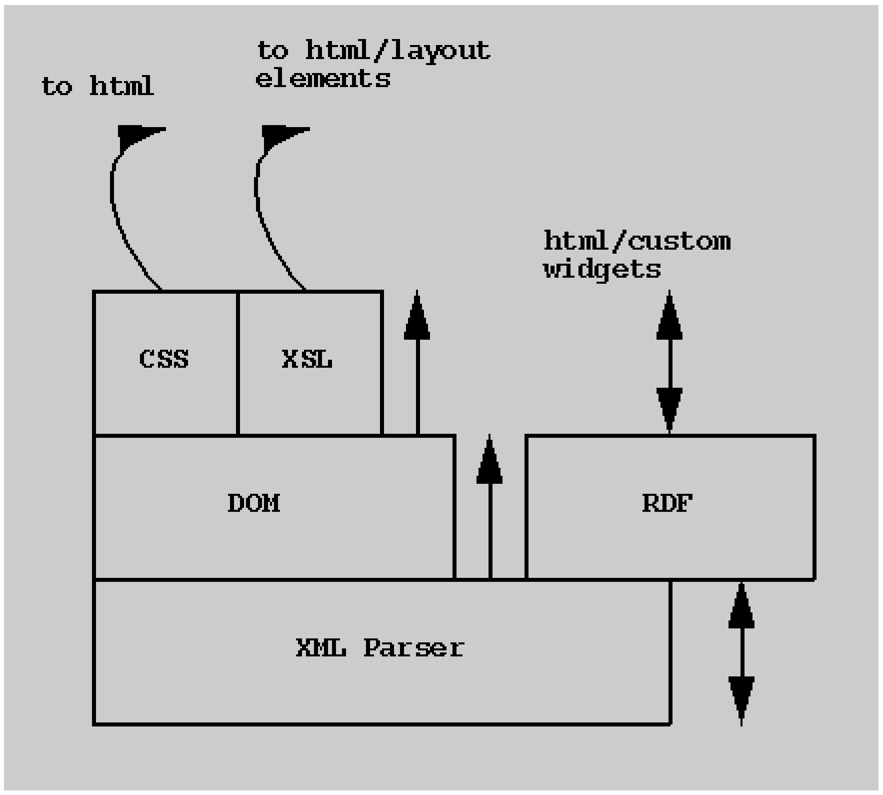

Figure 8: XML Parser Architecture

(Source: http://www.mozilla.org/rdf/doc/guha-1.gif)

JavaScript Interpreter

Firefox's JavaScript engine exposes a public API applications can call on for JavaScript support. The JavaScript Interpreter includes SpiderMonkey which is a C implementation of JavaScript. Like the XML parser, the JavaScript interpreter is strongly tied to Gecko.

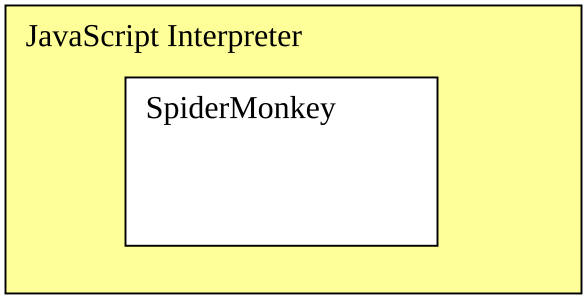

Figure 9: The JavaScript Interpreter

XPCOM

The Cross-Platform Component Object Model (XPCOM) manages creation, ownership, and deletion of objects within the application. XPCOM components talk to one another via interfaces written in XPIDL, XPCOM's Interface Definition Language (IDL). While most components coexist within the main application thread, XPCOM does support threads so it is possible to run components concurrently.
Some important XPCOM modules:

- **Component Manager**
  The Component Manager is responsible for creating instances of XPCOM components.
- **Service Manager**
  The Service Manager provides access to XPCOM services. Services differ from components in that only one instance of a service may exist at any given time, whereas the Component Manager returns a new instance of a component each time.
- **Category Manager**
  The Category Manager provides access to components that have been grouped into categories. Components within a category usually share common interfaces and functionality.
- **XPConnect**
  XPConnect provides a bridge between JavaScript and XPCOM components. It allows JavaScript code to use XPCOM components, as well as allowing XPCOM components to interact with JavaScript Objects.

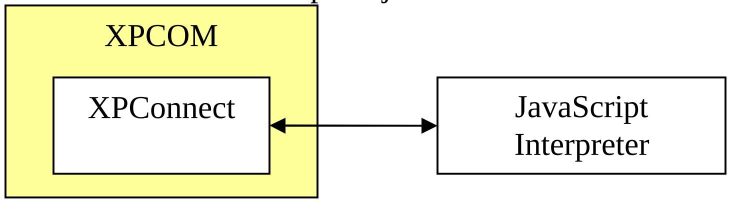

Figure 10: XPCOM Conceptual Architecture

XPCOM also provides some core classes for file and memory management, threads, and basic data structures.

Data Persistence

The data persistence mechanism is called DOM Storage. The DOM Storage API provides a way to store meaningful amounts of client-side data in a persistent and secure manner.

- Session Storage - This is a global object that maintains a storage area that's available for the duration of the page session.
- Global Storage - This is a global object that maintains multiple public, and private, storage areas that can be used to hold data over a long period of time (e.g. over multiple pages and browser sessions).

References to persistent storage items are accessed by the HTML code rendered through Gecko. Methods within the DOM Storage API are accessed via references within the HTML supplied by the server. The methods for accessing the session and global storage are outlined in the whatwg specification [R1, R2].

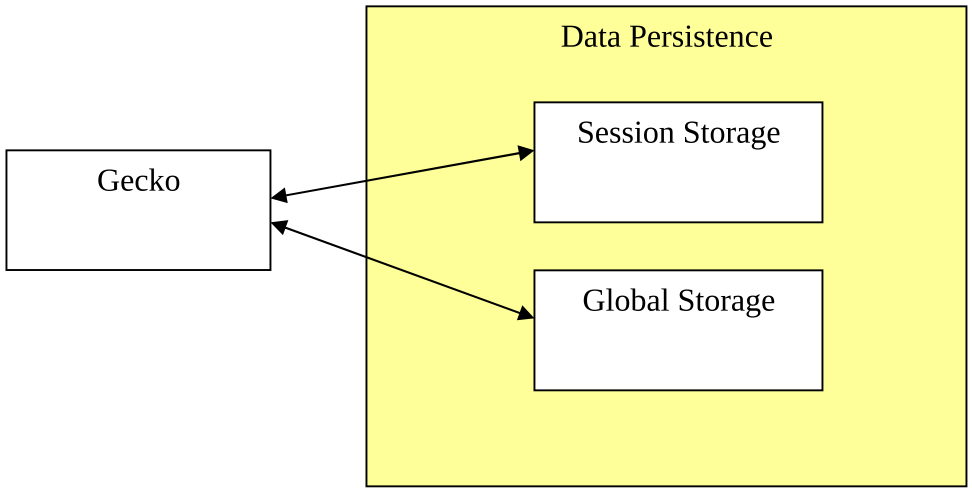

Figure 11: Data Persistence Conceptual Architecture

Conclusion

After building conceptual architecture of Mozilla Firefox, we have a good understanding of the application. Firefox includes the following subsystems: User Interface, Gecko, Necko, XML Parser, JavaScript Interpreter, XPCOM, and Data Persistence. We have discussed each subsystem and its components in the previous sections. Besides subsystems, we also found out that XPCOM is the core of Mozilla platform. XPCOM is a lower layer that it provides object level services to other subsystems. This is the reason we separate it from other systems. The future work in SEng 422 term project is to develop concrete architecture of Firefox. We will use tools such as jGrok and lsedit to extract architecture from Firefox source code. The concrete architecture will be developed based on the conceptual architecture that we discussed in this report.

References

[1] Session Storage specifications - http://www.whatwg.org/specs/web-apps/current-work/#sessionstorage
[2] Global Storage Specifications - http://www.whatwg.org/specs/web-apps/current-work/#globalstorage
[3] Documentation on DOM Storage - http://developer.mozilla.org/en/docs/DOM:Storage
[4] Data Flow Inside Gecko - http://developer.mozilla.org/en/docs/Gecko_Embedding_Basics#Appendix:_Data_Flow_Inside_Gecko
[5] XPConnect - http://developer.mozilla.org/en/docs/XPConnect
[6] XPCOM Part 3 – Setting up XPCOM - http://www.ibm.com/developerworks/webservices/library/co-xpcom3.html
[7] XPCOM - http://developer.mozilla.org/en/docs/XPCOM
[8] Multithreading in Necko - http://www.mozilla.org/projects/netlib/necko_threading.html
[9] Necko: the new netlib project - http://www.mozilla.org/docs/netlib/necko.html
[10] XML User Interface Language - http://www.mozilla.org/projects/xul
[11] XUL Tutorial - http://www.xulplanet.com/tutorials/xultu/intro.html
[12] XUL Reference - http://developer.mozilla.org/en/docs/XUL_Reference
[13] Introduction to XUL - http://developer.mozilla.org/en/docs/Introduction_to_XUL
[14] Alan Grosskurth, Michael W. Godfrey, Architecture and evolution of the modern web browser, http://grosskurth.ca/papers/browser-archevol-20060619.pdf
[15] Alan Grosskurth and Michael W. Godfrey, A Reference Architecture forWeb Browsers, http://ieeexplore.ieee.org/iel5/10097/32336/01510168.pdf?arnumber=1510168
[16] Alan Grosskurth and Michael W. Godfrey, A Case Study in Architectural Analysis: The Evolution of the Modern Web Browser, http://plg.uwaterloo.ca/~migod/papers/2007/emse-browserRefArch.pdf
[17] Antonio D'souza, Kristina Hildebrand, Gilad Israeli, Conceptual Architectual of Mozilla, http://www.cs.uwaterloo.ca/~kdhildeb/cs746/conceptual.pdf