An Analysis of the Google Chromium Software Architecture and Its Extensibility Options

Joshua Keating

INFO 498 Software Architecture

University of Washington

System Description

Chromium (and its proprietary sibling, Chrome) is a web browser. As such, it is used to navigate the World Wide Web. Chromium/Chrome (hereafter referred to as Chrome) is developed by Google and was released in September 2008 [1]. The web browser, typically referred to as the browser is used for three main functions: retrieving information, displaying that information, and allowing access to additional information. This means that the browser acts as the primary interface that users interact with the web. Beyond this core operation, most browsers offer a way to extend functionality through the additions of extensions or add-ons. In this Chrome is no exception [2]. These extensions can modify the behavior of Chrome or add functionality. Extensions are typically built with a combination of files that provide some user interface and interact with the browser in a certain way. Extensions typically communicate with web pages or servers as well as other browser features [3].

Keywords: browser, software architecture, extensions, multi-process, WebKit

1. Architecture Description

Modern browsers are complex systems of software and the Chromium project is at the forefront of browser development. Despite the ever changing nature of a currently developed software project, the core of the browsers architecture has remained the same throughout much of its public release. At a high level, Chrome is comprised of a number of components that work together to create a browser that is stable, performance, stability, and secure. At its core Chrome is built around a multi-process architecture. This type of architecture lends itself to the sandboxed design that Chrome is built around. This helps immensely in improving the overall stability and security of the browser. By eliminating the potential of a malfunctioning web application or page impacting the entire system, Chrome can continue

[1] See https://arstechnica.com/information-technology/2008/09/google-unveils-chrome-source-code-and-linux-port/

[2] Chrome blogpost from 2009 https://chrome.googleblog.com/2009/12/google-chrome-for-holidays-mac-linux.html

[3] Overview of Chrome extensions https://developer.chrome.com/extensions/overview

to function even if one of its processes is compromised. The high level overview of Chrome includes the following components:

1.1. Browser

This is the core component of Chrome and is responsible for driving the engine of the browser as well as keeping track of persistent data (data that persist between sessions). The browser is tasked with running the User Interface (UI) and managing tab and extension processes. The browser also communicates with the Network and the rendering engine. It uses local procedure calls to connect to all its components but the network where it uses remote calls.

1.2. WebKit or Blink[4]

In Chrome the rendering engine that is used is WebKit but in Chromium the Bink engine is used. As this paper is not focused on comparing rendering engines, the overview of WebKit will be given. At a high level, the differences between these two are not relevant. Chrome uses its rendering engine to define the layout of web pages. WebKit communicates with the browser to obtain privileges outside of rendering through local connection calls. WebKit is dependent on three additional components beyond the browser:

1. V8
    - V8 is the Javascript engine that is used by Chrome.[5] Developed by Google, it is high performance and is responsible for interpreting Javascript that is passed on from the rendering engine. Due to this one way relationship, WebKit will depend on the V8 engine to process any Javascript that is contained in a page.
2. Extensions
    - Extensions in Chrome are handled in the same way as its other components. They occupy their own process and depend on the Browser and WebKit to display and have access to system resources. As this is a main focus of the paper, it will be covered in depth in section 2.
3. Display
    - The Display component is responsible for widget creation, graphics rendering, and font rendering.[6] It is a dependency of WebKit and the User Interface.

[4]Blog post on the switch from WebKit to Blink https://blog.chromium.org/2013/04/blink-rendering-engine-for-chromium.html

[5]See https://developers.google.com/v8/intro

[6]This paper was invaluable in writing this report https://archrometects.files.wordpress.com/2009/10/assignment-01-conceptual-architecture-of-google-chrome-archrometects.pdf

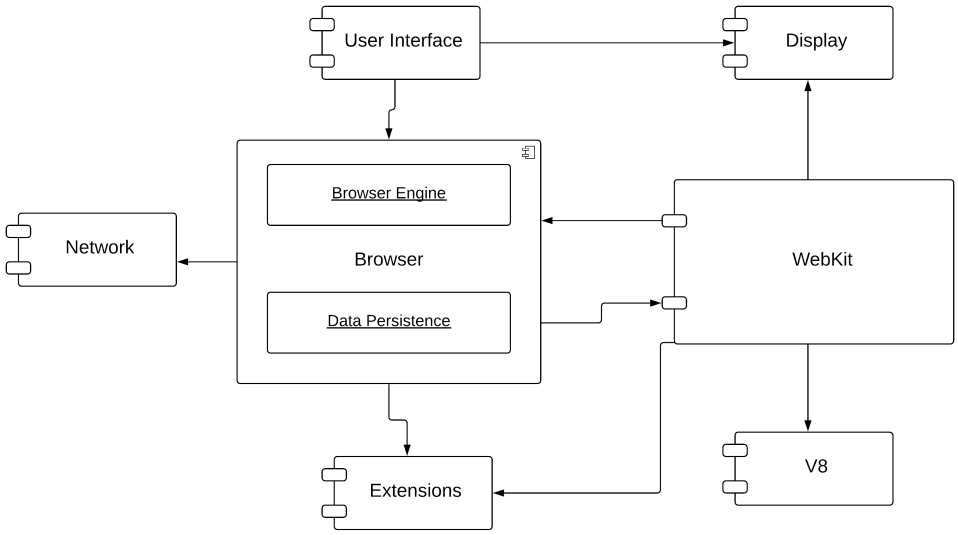
1.3. User Interface

The User Interface is the component that is most visible to the user. It allows the user to access the browser and interface with its elements. At a high level, this is made up of client and non-client areas. The difference here is intuitive, non-client areas are not typically directly accessed (window borders and title bar) and client area covers the area below the URL bar. The User Interface is dependent on the Browser to communicate what to display.

1.4. Network

The Network stack is primarily responsible for handling network communication. It handles URL requests and is dependent on the Browser and the V8 engine. Besides from fetching resources from the network, this component is not coupled with other components heavily.

Figure 1: Google Chrome high level architecture[7]



2. Extensions in Chrome

Extensions are a way to add functionality to the browser that does not exist in its base state. By defining extensions outside of the Browser and WebKit components, Chrome maintains its sandboxed, decoupled approach to architecture. Each extension is run in its own process and are created with the use of the Browser logic. Chrome has its own framework of defining extensions that are used in the browser. Extensions are assigned a limited amount of responsibility and access based on an opt-in basis. At a high level, an extension in Chrome needs to interface with at least one of the current tabs in the session. Extensions are architected in such a way so that the extension will registers callbacks to

Chrome for a specific situation using a defined interface.[8] Based on the interface, the tab will change their content. For the specific extension that I will be developing, I will be using three of these interfaces:

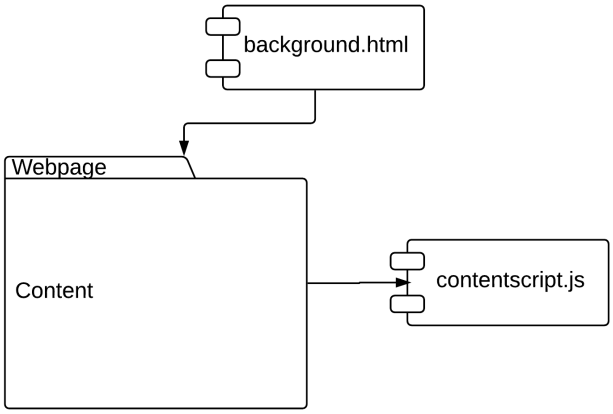
2.1. Content Script

A content script is JavaScript that runs on a given page that is currently loaded. Content scripts will read details of the web pages the browser visits and make changes based on defined criteria. The relationship between the script and the background page is presented in Figure 2. In my case, the script will alter the current selector and change some CSS.

2.2. Background Page

The Background Page exists as the controller of the extension and can be either persistent or event based. They are recommended to be event based. By using event based pages, page control logic can be loaded in on an as needed basis.

Figure 2: Example of the relationship between the Background Page and Content Script



2.3. Page Action

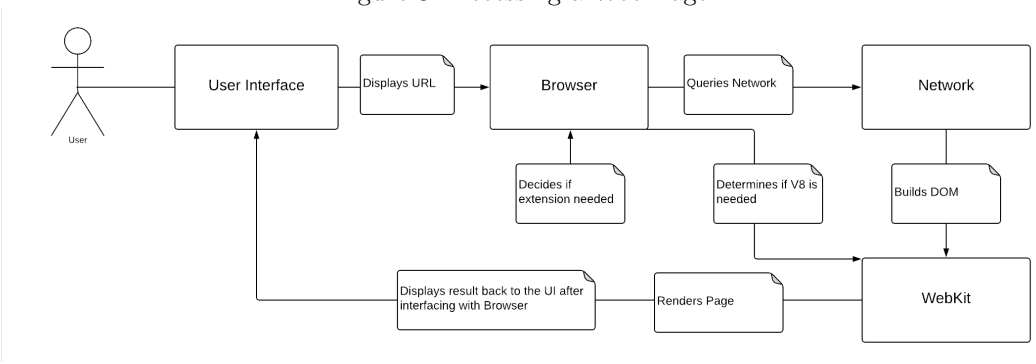
Page Actions are a part of the UI and are displayed to the right of the URL bar. They can be configured with a popup as a way for the user to interact with a specific page. I plan on using the popup to alter extension behavior on a site by site basis.

3. Flow of Application

This figure outlines the flow of data through the components:

[8]Useful guide to Chrome extensions https://gist.github.com/jjperezaguinaga/4243341

Figure 3: Accessing a Web Page



4. Architecture Analysis

Google Chrome uses multiple architectural styles under its overarching multi-process architecture. If forced to give a label, the N-tier architecture might be suitable as a generalization. A subset of a layered/tiered architecture, the N-tier style is a type of multilayered architecture where the presentation, processing, business and data management levels are separated. N-tier is not confined to any set number of layers and can be expanded to any number of possible layers. In the instance of Chrome, the User Interface as the Presentation tiers works with the Browser component alongside the WebKit component as the Logic tier to handle the view and interaction with the web. In utilizing this type of architecture, Chrome is able to engender a number of quality attributes. These qualities are:

- **Modifiability/Maintainability:** Because of the separation of concerns that the multi-process architecture promotes, different components should not be negatively affected by changes.
- **Reusability:** Because the layers are cohesive, they are also reusable.
- **Security:** By giving each piece of the system its own process, components are sandboxed and can fail without breaking others.
- **Usability:** Separation of components makes makes usability more intuitive and easy to localize and globalize.[9]

By designing a software system in this way, Chrome allows itself a high degree of modifiability. It offers tools for developers to create extensions that augment the functionality of the browser. Using a semi-strict layering design Chrome is able to promote code reusability and component modularity. By separating the rendering engine from the browser and encouraging subsystems to agree on a common interface, components can be developed independently from one another as they do not have to rely on an upper layer dependency.

[9]See https://msdn.microsoft.com/en-us/library/ee658094.aspx