A pattern for Web Browser Infrastructure

Paulina Silva
Departamento de Informática
Universidad Técnica Federico
Santa María
Valparaíso, Chile
pasilva@alumnos.inf.utfsm.cl

Raúl Monge
Departamento de Informática
Universidad Técnica Federico
Santa María
Valparaíso, Chile
rmonge@inf.utfsm.cl

Eduardo B. Fernandez
Department of Computer &
Electrical Engineering and
Computer Science
Florida Atlantic University
Florida, USA
fernande@fau.edu

ABSTRACT

Currently most of software development creates systems that are connected to the Internet, which allows to add functionality within a system and facilities to their *Stakeholders*. This leads to depend on a *web client*, such as the *Web Browser*, which allows access to services, data or operations that the system delivers. However, the Internet influences the attack surface of the system, and unfortunately many stakeholders and developers are not aware of the risks to which they are exposed. The lack of security education of software developers, the scarce and scattered documentation for browsers (and standardization), could become a big problem in large architectural developments which depend on browser to perform their services. A Reference Architecture of the *Web Browser*, using Architectural Patterns, could be a basis for understanding the security mechanisms and its architecture, which interacts with a bigger web system. This would give a unification of ideas and terminology, giving a holistic view of independent implementation details for both the browser and the system it communicates with. We developed a Browser Infrastructure Pattern which describes the infrastructure to allow the communication between a Web Client and a Server in the Internet. With this work we propose an Architectural Pattern as the first piece of our Reference Architecture for Web Browser and security concerns.

Keywords

Web Browser, Web Client, Modular Architecture, Browser Architecture, Reference Architecture, Browser Infrastructure Pattern

Introduction

Background

Patterns are encapsulated solutions to recurrent problems and define a way to express requirements and solutions concisely, as well as providing a communication vocabulary for designers [1]. The description of architectures using patterns makes them easier to understand, provides guidelines for design and analysis, and can define a way of making their structure more secure.

Security patterns describe solutions to the problems of controlling (stopping or mitigating) a set of specific threats through some security mechanism, defined in a given context. The most common use of security patterns is to help application developers -who are not security experts- to add security in their designs. Patterns of this kind also are used to reinforce a legacy system.

The aim of a Reference Architecture is to provide a guide for developer, who are non security experts, in the development of architectures for concrete versions of the system or to extend it. With the use of architectural patterns we describe the Browser Architecture as a Reference Architecture (RA). An RA is created by capturing the essentials of existing architectures and by taking into account future needs and opportunities, ranging from specific technologies, patterns and business models. It can also be derived from domain models.

A Security Reference Architecture is a Reference Architecture where security services have been added in appropriate places to provide some degree of security for a system environment. The basic approach that we will use to build a Security Reference Architecture is by applying a systematic methodology from [2, 3, 4], which can be used as a guideline to build secure web browser systems and/or to evaluate their security levels. We started to build a Reference Architecture as a first step, in a student work, and now we are trying to improve it using security patterns and misuse patterns. By checking if a threat, expressed as a misuse pattern, can be stopped or mitigated in the security reference architecture, we can evaluate its level of security.

In this work, a Browser Infrastructure Pattern is presented as a first step in to the process of developing a Reference Architecture (RA) and Security Reference Architecture (SRA) for the Web Browser. Threat analysis and security patterns were done in a previous work, and we will improve the architecture in the construction of the SRA.

Browser Infrastructure Pattern

Intent

The Browser Infrastructure allows to request a web resource in the Internet a **Browser User** (BU), which is a user who uses a browser within a **Host**. The pattern lets visualize the communication between the components that make the Web Browser and the Provider (i.e, a Server), to whom the

WOODSTOCK '97 El Paso, Texas USA
© 2015 ACM. ISBN 123-4567-24-567/08/06.
DOI: 10.475/123_4

request is made.

Example

Within the Host it is possible a lack of resources to fullfill the user needs. The request of external services or resources is the main reason of the Internet existence. This kind of task it is possible to do in a lot of ways, it all depends on what the Provider wish to deliver to others.

Context

Browser User is a Host user which uses a Browser, and the Provider is an entity which can be accessed in the Internet. The contact between each other is normally done by Web Applications or Servers which communicates using the HTTP protocol. A Browser let the Browser User access and visualize the external resources a Provider has and a Browser User may need.

Problem

Some Browser Users could need resources from a Provider, but the user maybe will need them in a special format or they should be presented in the screen of the computer to be visualized. In this case, if appropiate tools are not available, the resource could not be helpful if it can not be used correctly. How can the Host and Provider be prepared to this situation? The solution to this problem must resolve the following problems:

- Transparency: the user behind the Host should not be aware of what it is done, while a request to a Provider has been issued.
- Stability: The *Browser* must be capable of working, even if a web page had a problem to be seen or there is an internal problem.
- Isolation: Each *request* must not interrupt others.
- Heterogeneity: It does not matter the type of Provider to which the Browser communicates, it should be possible to interact with whatever type is it, and also it should be capable to show adequately the content of the obtained resource.
- Availability: The user of the Host, should be capable to request at any time.

Solution

A *Web Browser* can satisfy the request of a user of the Host by the Browser User, either by one or more instances of the Browser User, which allows for a variety of options to browse in the Internet. A Browser must be able to deliver a fast and stable navigation, without affecting each accesed sites.

Structure

The Browser Client (BC) is an entity that represents the main process of a Web Browser and comprises the minimum number of components which constitute a Browser. A Host (H) houses and interacts with the BC. H is composed mainly by Hardware (HW) and a Operative System (OS). At the same time, a Provider (P) has also HW and SO, but additionally has a Web Server (WS) which is responsible for receiving external requests. Browser Client (BC), GPU Instance (GPUI), Extension (Ex) and Plugin are instances of Process (Pr), which resides within a H. Most Browsers use a central component to do operations that need to affect the Host of the Browser. Figure 1 shows the Class diagram for the Browser Infrastructure Pattern. For each resource a BC requests, a Sandbox hosting each Web Content Renderer (WBR) instance created will allow the browsing and visualization of the obtained resource. The BC component acts as a broker for the requests comming from the Sandbox (which host a WBR), this allows a fine control over the sent messages (using IPC/IPDL/COM) between communicating process. If there is a need for the H resources from a GPU Instance, Extension and Plugins they need to communicate directly with the Sandbox, which in its instead will ask to the BC for resources. A user who makes a request to a Internet resources using a Web Browser, will be called Browser User (BU). BU uses BC to make requests to one o more Providers, where the latter uses a Web Server (WS) to receive requests and reply back (Figure 1).

Dynamics

Some use cases are the following:

- Make Request (actor: Browser User)
- Cancel Request (actor: Browser User)
- Save Resource (actor: Browser User)
- Receive Request (actor: Provider)
- Ask for Resources (actor: Host)

We show in detail Make Request below. (Figure 2):

Summary

A Browser User needs a URL resource which can be obtained by using the HTTP protocol, as required by the Provider. The Browser Client will be used by a User Browser to perform the display of the URL resource.

Actor

Browser User

Preconditions

The Host must have one or more Browser Client for the Host user. In addition to being connected to a network or the Internet. The Provider you want to contact must also be available.

Description

Note: Messages between the Browser Client and Sandbox can be both synchronous and asynchronous [5, 6]. We do not explain in this work synchronization details, because it is out of our scope. We are interested only in spcifying who interacts with whom.

1. A Browser User requires a browser to access a URL for some resource in a Provider, this is done by using an already instanced Browser Client in the Host. Inside the Sandbox there is an instance of Web Content Renderer pattern.
2. The Sandbox requires the Host resources to obtain what is behind the URL. A request is made from the Sandbox to the Browser Client through a communication channel such as IPC, IPDL or COM (depending

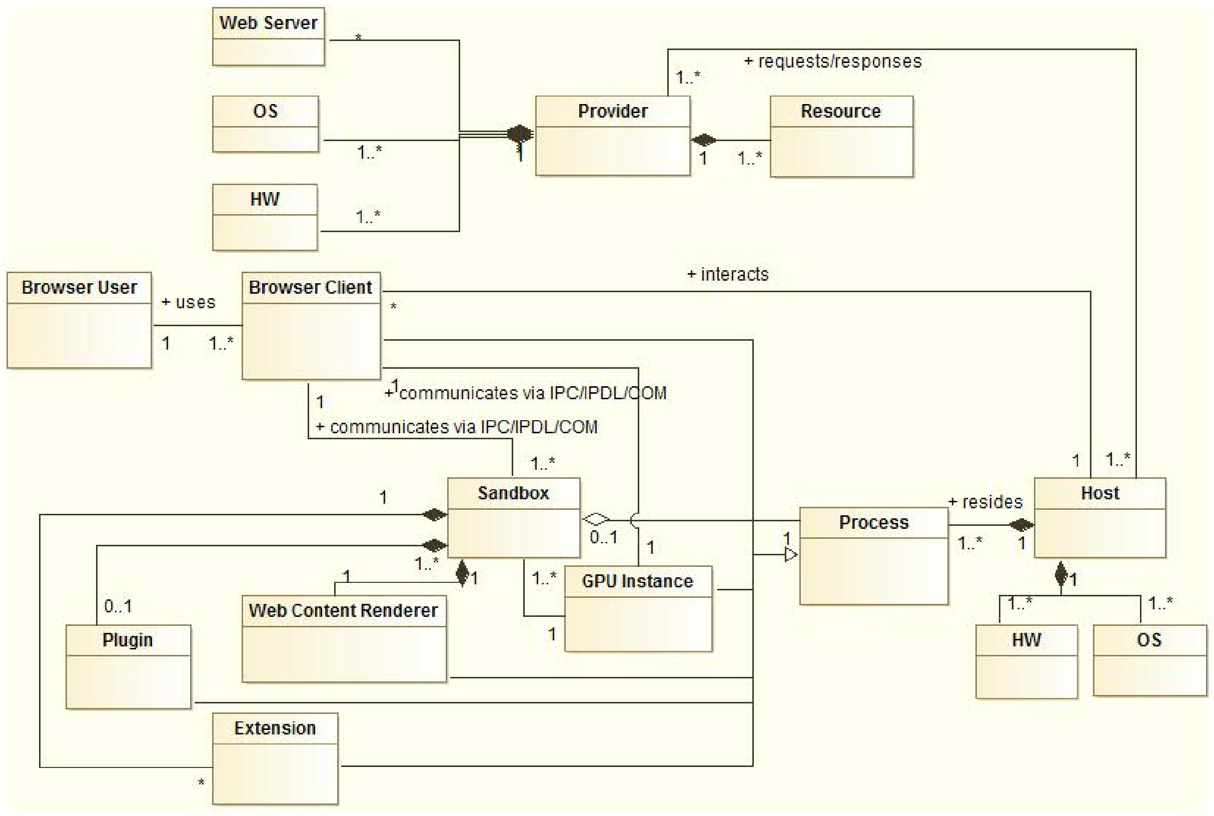

Figure 1: High-level Components of the *Browser*.

on the *Browser* used), using a limited API to communicate to a process of greater privilege.

3. The Browser Client receives the request, and verifies through its policy engine if the Sandbox action is allowed.

4. If the Sandbox action is permitted, the Network API within the Browser Client, to obtain a host resources (via system calls), it is used. The Browser Client communicates internally with the Host, and the latter must review its policies to ensure that the Browser Client has the privilege of making a request to the Host resources.

5. If access to the resource is allowed, the Browser Client may *request* through the Network API. If the *request* is not a *pre-flight*, the Provider will receive the *request* and work on it.

6. The Provider will send a *response* to the *request* received. Depending on how it is implemented the Browser Client, it may or not have to wait for the response (synchronous or asynchronous) of the Provider.

7. Once the response obtained it is stored in the cache, unless it is specified in another way.

8. The response to the *request* is sent by a communication channel to the Sandbox which originated and then the Web Content Renderer. If a response was received by the *request*, the Web Content Renderer is ready to prepare the parsing of the content or use a plugin or GPU to support the display of the resources obtained by the URL. Otherwise, the Web Content Renderer within the Sandbox will create an error page.

9. The *Renderer* obtains a bitmap to be sent to the Client Browser, so that the Host can present it. Before doing this, the BC should check that the Sandbox which host the Web Content Renderer possess the permissions to do so.

10. If the permissions are sufficient, the Browser Client sends the bitmap, as a parameter, in the system call made to the Host. Finally, the Host must check that the system call made by the Browser Client has the required permissions.

Alternative flow

- The Provider is not available.

- The resource pointed by the URL does not exists.

- The request is cancelled.

Postconditions

The *Browser* receives the resource indicated by the URL and it is displayed by the peripheral device output for the Browser User.

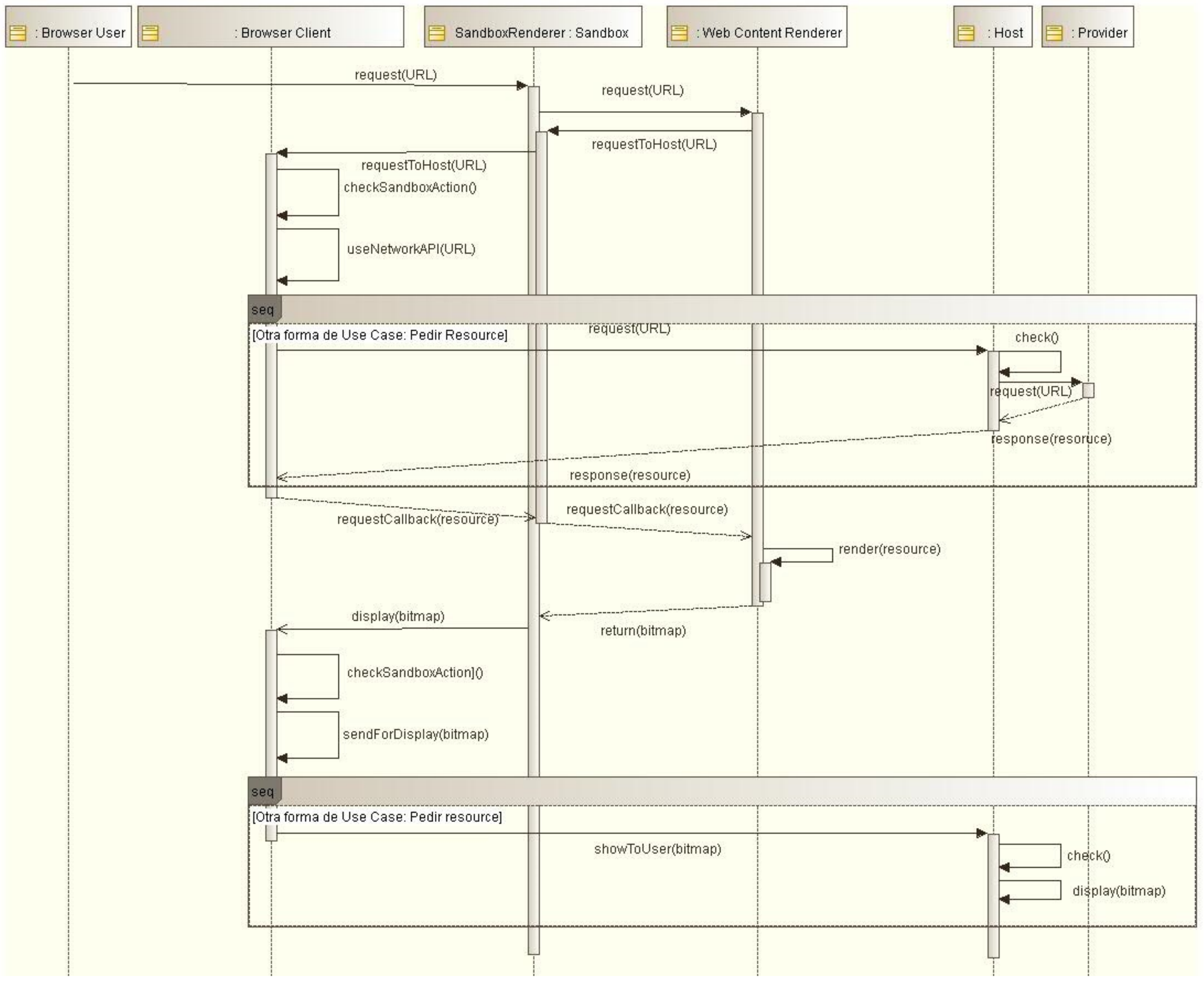

Figure 2: Sequence Diagram: Make Request.

Implementation

- The sandbox may be implemented in various ways. Google Chrome [7] is based on not reinventing the wheel and use the protection mechanisms which offers the OS (e.g, Windows or Linux) of the Host to protect the user. This prevents any process to access the file system, and having an restrictive API in the web Content Renderer. Google Chrome, Firefox and Internet Explorer affirm that Sandboxes are an important piece to the Browser because it realizes the principle of least privilege [8, 7, 9]. The minimum configuration for the Sandbox includes 2 processes: The privileged process or Broker who is represented by the Browser Client, and the processes hosted by Sandboxes or targets.

- To enforce the Same Origin Policy, Google Chrome, Firefox and Internet Explorer use different schemes; for example: Google Chrome leaves his work to the Renderer (Web Content Renderer in this case) to leave isolated pages/resources from various sites.

Consequences

The Browser Infrastructure pattern provides the following benefits:

- Transparency: The user navigation is done almost automatically, only in rare cases the user will have to make a decision on the resource asked.

- Stability: Because the Browser Client, Sandbox, and GPU Instance Plugin are independent Host processes, the failure of one will not generate problems in other (crash, memory corruption, etc.).

- Isolation: Depending on the type of isolation you can separate the different requests, so they do not interfere with each other, unless it is desired.

- Heterogeneity: Because each Browser Client tries to follow the standards of the W3C [10], every page that follows these guidelines can be viewed, as well as other resources.

- Availability: Each process is independent and has its own thread of execution, these were specifically created to help to maintain the User Interface smooth.

At the same time, this pattern has the following liabilities:

- Since independent processes are used to browse a resource (depending on the scheme using the browser), it is possible that a lot of resources of the Host are to be used to keep everything open.

- Provider who have not met the specifications of the W3C, their resource are displayed incorrectly by the Web Browser.

Example Resolved

With the given pattern it is now possible to navigate smoothly to all resources on the Internet we want. It is possible to provide through the isolation of the components: speed, security and stability. The Browser User will only concern about the navigation, unless it is required for its explicit permission to enter certain Host resources that are privileged (e.g, the file system). Each Host user can use the Browser Client they want, because each one is isolated by using separate processes.

Known Uses

- Currently, the separation of the components of the *Browser* in various processes, with different levels of access, is called as Modular Architecture [11]. This enables the separation of concerns in the browser, which gives greater stability, isolation, safety and speed.

- Google Chrome is based on the modular architecture, where each Renderer Process communicates with the Browser Kernel [12]. This proposal is used as a reference in the Mozilla project Electrolysis, as you can see in [13, 14], specially in the development of the sandbox and architecture.

- Internet Explorer, a proprietary browser, does not give much information about its structure or details of its implementation; [15] talks about Loosely-Coupled architecture [16] and its components, but without going into much details.

- Firefox, meanwhile has two implementations: monoprocess and multiprocess/modular. Electrolysis is the name of the modular architecture being implemented, but has not yet been fully completed.

Related Patterns

- The Web Content Renderer pattern, which is under development, represents the subsystem hosted by a Sandbox that allows the parsing of a resource obtained through a request.

- The Browser Kernel pattern, a pattern we are developing, represents the subsystem that represents the Web browser central component. This acts as a Reference Monitor pattern [17] for all requests the Renderer does.

- The Sandbox is another name for the pattern Controlled Execution Domain [17].

Conclusions and Future Work

A Web browser appears to be a medium complexity software for users and developers without security experience, but unfortunately this piece of software allows a variaty of attack vectors, to the user using it as well the system with which interacts. Therefore it is important to understand its structure and how it interacts with internal or/and external Stakeholders.

A part of our Reference Architecture has been built through the abstraction of found documentation, through the Browser Infrastructure pattern. We created our first architectural pattern for the infrastructure of *Web Browser*; to help others to understand holistically the components, interactions and relationships of this system. Furthermore it has been possible to characterize the Stakeholders and one of the most important use case. From what we have known, this is the second Reference Architecture for the *Browser* built [18]. The proposed work allows a better understanding of this system called Web Browser by using our partially Reference Architecture, this is also helpful to understand existing threats. Furthermore, as it is not subject to specific implementations, it is possible to generalize certain results in other browsers.

Future work will be related to the creation of a Security Reference Architecture for the *Web Browser* using the same methodology presented here. Other patterns related to Browser Infrastructure pattern will be obtained in order to complete the AR we already begun, such as the Web Content Renderer and Browser Kernel pattern. An example of the type of work to be carried out can be seen in [4] where this study carries out activities to build secure software and evaluate the safety levels of a system already built.

We plan to build more Misuse patterns, for the Browser Infrastructure pattern, to continue the study of the possible threats in the *Browser*, as a way to educate Developers and Stakeholders. While at the same time these patterns will allow the construction of the Security Reference Architecture for the browser. In the same line, in addition to finding potential threats existing in the system, we need to find countermeasures or security defenses to prevent or foresee such threats through security patterns on the reference architecture built. This is possible to perform under the same exercise already conducted in this work, looking for threats at each action for each use case of the Browser.

1. REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software.* Pearson Education, 1994.

[2] E. Fernandez, M. Larrondo-Petrie, T. Sorgente, and M. VanHilst, “A methodology to develop secure systems using patterns,” *Chapter*, vol. 5, pp. 107–126, 2006.

[3] E. B. Fernandez, N. Yoshioka, H. Washizaki, J. Jurjens, M. VanHilst, and G. Pernu, *Using Security Patterns to Develop Secure Systems*, H. Mouratidis, Ed. IGI Global, 2011. [Online]. Available: http://www.igi-global.com/chapter/using-security-patterns-develop-secure/48405

[4] E. B. Fernandez, R. Monge, and K. Hashizume, “Building a security reference architecture for cloud systems,” in *Proceedings of the WICSA 2014*

Companion Volume. ACM, 2014, p. 3.

[5] B. McCloskey, "Multiprocess Firefox." [Online]. Available: https://billmccloskey.wordpress.com/2013/12/05/multiprocess-firefox/\#ipc

[6] G. Chromium, "Inter-process Communication (IPC)." [Online]. Available: https://www.chromium.org/developers/design-documents/inter-process-communication

[7] "Sandbox - The Chromium Projects." [Online]. Available: http://www.chromium.org/developers/design-documents/sandbox

[8] M. V. Yason, "Diving into IE 10's Enhanced Protected Mode Sandbox."

[9] "Security/Sandbox - MozillaWiki." [Online]. Available: https://wiki.mozilla.org/Security/Sandbox

[10] W. W. W. C. W3C, "About." [Online]. Available: http://www.w3.org/Consortium/

[11] T. Vrbanec, "The evolution of web browser architecture," pp. 472–480, 2013.

[12] "Multi-process Architecture - The Chromium Projects." [Online]. Available: https://www.chromium.org/developers/design-documents/multi-process-architecture

[13] "Security/ProcessIsolation/ThreatModel." [Online]. Available: https://wiki.mozilla.org/Security/ProcessIsolation/ThreatModel

[14] "Features/Security/Low rights Firefox - MozillaWiki." [Online]. Available: https://wiki.mozilla.org/Features/Security/Low{_}rights{_}Firefox

[15] M. Crowley, *Pro Internet Explorer 8 & 9 Development: Developing Powerful Applications for The Next Generation of IE*, 1st ed. Berkely, CA, USA: Apress, 2010.

[16] "IE8 and Loosely-Coupled IE (LCIE) - IEBlog - Site Home." [Online]. Available: http://blogs.msdn.com/b/ie/archive/2008/03/11/ie8-and-loosely-coupled-ie-lcie.aspx

[17] E. Fernandez-Buglioni, *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013.

[18] A. Grosskurth and M. W. Godfrey, "A reference architecture for web browsers," 2005, pp. 661–664, uRL: `http://grosskurth.ca/papers.html#browser-refarch`.