A Reference Architecture for web browsers: Part I, A pattern for Web Browser Communication

Paulina Siva[1], Raúl Monge[1] and Eduardo B. Fernandez[2]

[1]Departamento de Informática, Universidad Técnica Federico

[2]Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University

Paulina Silva pasilva@alumnos.inf.utfsm.cl, Raúl Monge rmonge@alumnos.inf.utfsm.cl, Eduardo B. Fernandez fernande@fau.edu

Abstract. *Currently, most software development is focused in creating systems 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 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 awareness in security among software developers and the scarce and scattered documentation for browsers (and standardization) could become a big problem in large architectural developments that depend on browsers to perform their services. A Reference Architecture (RA) of the web browser, using architectural patterns, could be a starting point for understanding its security mechanisms and architecture, which interacts with a bigger web system. This would unify ideas and terminology, giving a holistic view of independent implementation details for both the browser and the system it communicates with. We developed a Web Browser Communication pattern that describes the infrastructure to allow the communication between a Web Client, a web browser, and a Server in the Internet. We will focus, in this article, on interaction aspects of the web browser infrastructure, showing how the principal components/subsystems of the browser interact. With this work we propose an architectural pattern as the first piece of our Reference Architecture (RA) for web browsers and security concerns.*

Categories and Subject Descriptors

D.2.11 [Software Architectures]: Patterns

General Terms

Design

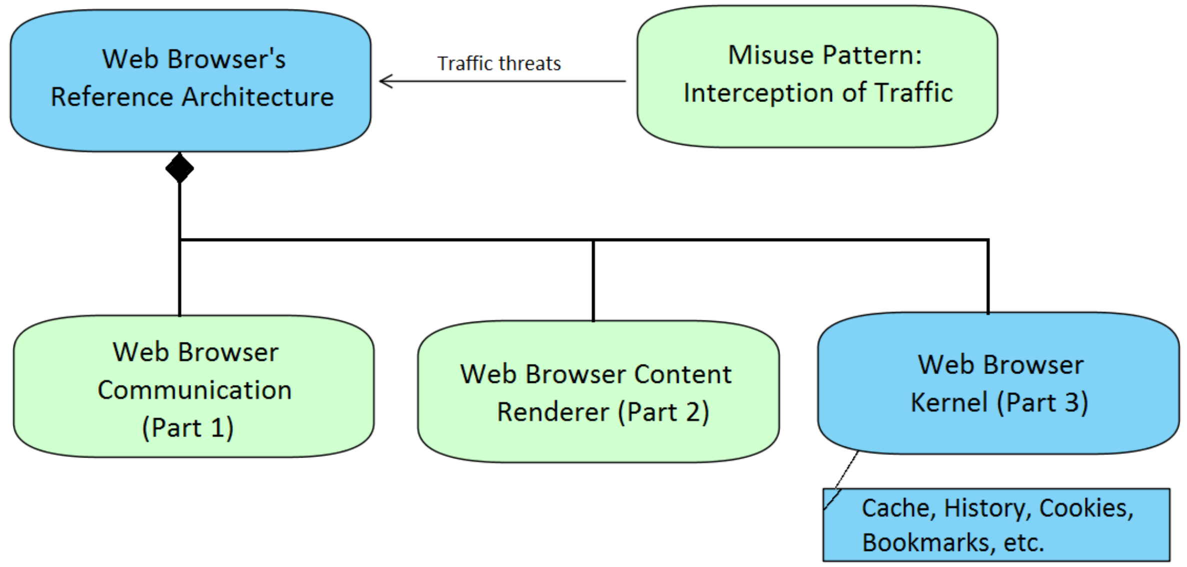
Keywords

Browser, Web Client, Browser Architecture, Reference Architecture, Pattern

1. Introduction

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.

The aim of a Reference Architecture is to provide a guide for developers, who are not security experts, to develop architectures for concrete versions of the system or to extend such system. With the use of architectural patterns, we describe the web 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. The pattern diagram in Figure 1 shows relationships between patterns and we can see how different models relate to each other, where round rectangles represent patterns and the arcs indicate dependencies between patterns.


Web Browser's Reference Architecture
Traffic threats
Misuse Pattern: Interception of Traffic
Web Browser Communication (Part 1)
Web Browser Content Renderer (Part 2)
Web Browser Kernel (Part 3)
Cache, History, Cookies, Bookmarks, etc.

interaction aspects of the web browser infrastructure, showing how the principal components/subsystems of the browser interact. In future work we will discuss the subsystem within this pattern related to the principal functionalities of the web browser like caching, cookies, Browser User's bookmarks, etc. called Browser Kernel. The audience to which our paper is focused are browser developers, web application developers, researchers and for teaching purposes. Being the first two the most important, since they are the responsible for the implementation and correct use of secure mechanisms while a user is browsing in the Internet. Secure communication is an important matter, but in this work the focus will be on the Browser and how to control the responses it receives.

2. Web Browser Communication Pattern

2.1 Intent

The web browser communication pattern describes the architecture for the processing of a user request for web resources in the Internet.

2.2 Example

Within the host is possible that resources needed to fulfill the user needs are limited. The request of external services or resources is the main reason of the Internet existence. If a user needs to make a bank transaction, such as deposit money to another party, the browser's user will supply an URL to the browser for accessing the online banking service of the bank the user belongs to.

2.3 Context

Users need to access services or information in the Internet, for which they use browsers. A browser starts by accepting a URL from a user and sending a request to the corresponding IP address. It also receives responses that a user expects.

2.4 Problem

Internet users need resources from providers/servers, but they may need them in a specific format, for example to be visualized on the screen of a computer. In this case, if appropriate tools are not available, the resource could not be helpful and it cannot be used correctly. How can the host and server provide these functions? The solution to this problem must resolve the following forces:

- *Transparency:* the user should not be concerned about how his/her request is performed.
- *Stability:* The browser must be capable of working, even if a web page cannot be displayed properly or there is an internal problem in the server.
- *Isolation:* Each request must not interrupt others.
- *Heterogeneity:* It does not matter the type of provider with which the browser communicates, it should be possible to interact with any type, and it should be capable of showing the content of the obtained resource adequately.

2.5 Solution

Provide a device, the web browser, with the functions needed to understand user requests (redirection, user interaction, etc.) and send them to the provider. A web browser satisfies a request of a Browser User of the Host.

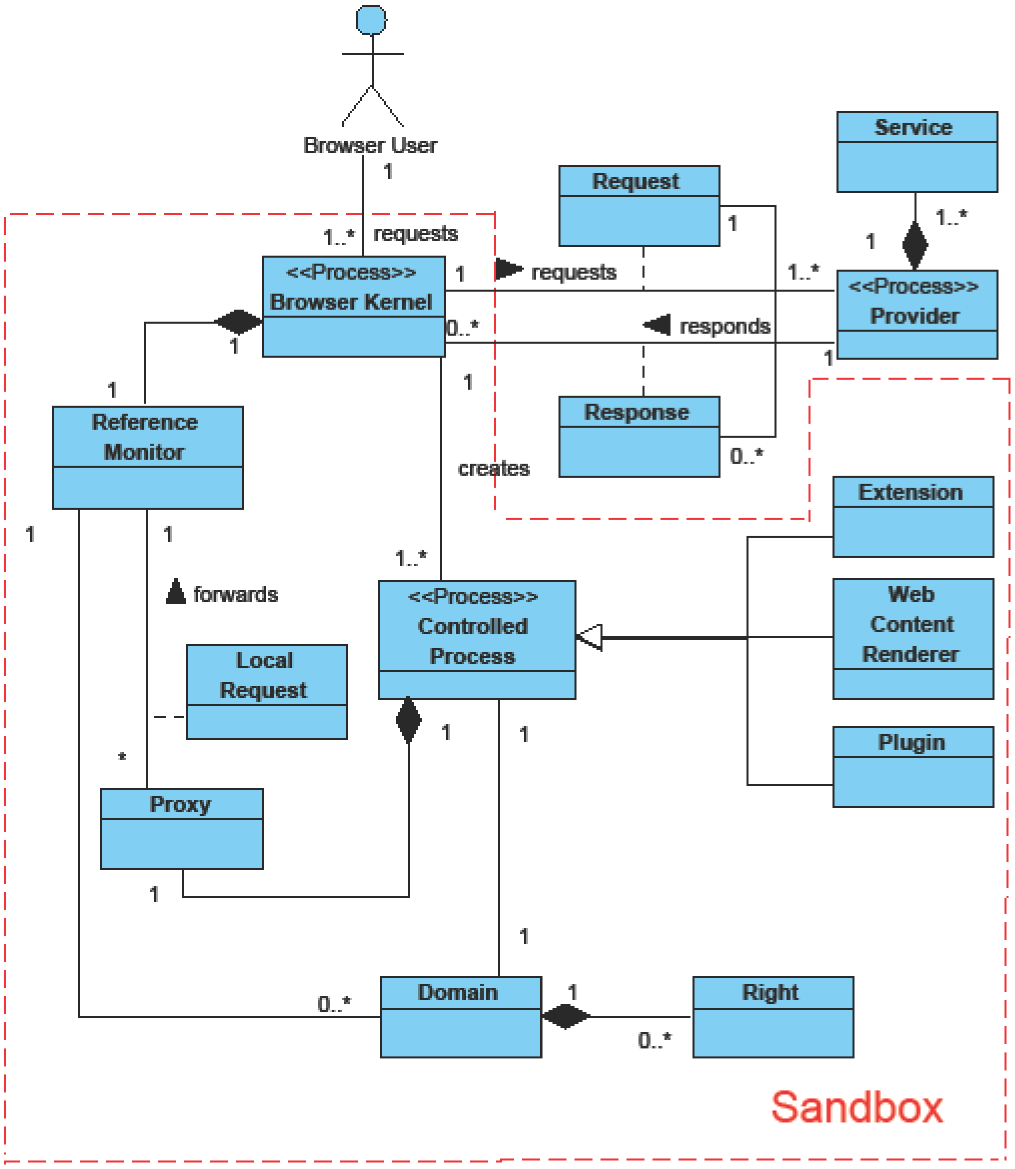


Figure 2. High-level Components of the web browser.

2.5.1 Structure

In Figure 2 the **Browser Kernel** is an entity that represents the main process of a web browser, which is constantly communicating with many hosts. A user who makes a **Request** to an Internet resources using a web browser, will be called **Browser User**. At the same time, a **Provider** is responsible for receiving external requests. According to the request, a **Provider** will send a message back of the **Service** (or resource) to the supplied **Browser User** needs in a **Response** message. Most Browsers use a central component, a **Browser Kernel**, to do important operations like saving for the **Browser User** bookmarks, web resources, cookies, caching and others. Figure 2 shows the class diagram for the **Web Browser Communication pattern**. For each new resource a **Browser Kernel** requests a **Web Content Renderer** instance is created or reused; this will inherit the **Controlled Process** properties and its methods. A **Plugin** and an **Extension** are elements that extend the functionality of the browser; the extender being for the exclusive use of the browser while the plugin can be shared with other systems, such as the Adobe Reader Plugin. A **Sandbox** is a Controlled Execution Domain [5] created for a single **Controlled Process** instance. The

Sandbox may allow the process memory isolation and the access control of each communication between processes, such as an instance of a **Controlled Process** with the **Browser Kernel**; this applies to a **Web Content Renderer**, a **Plugin** and an **Extension** as well. To communicate with the **Browser Kernel**, a **Proxy** created within a **Controlled Process** forwards a **Local Request** to the **Reference Monitor** inside the **Browser Kernel**. This **Proxy** acts as a security measure, a protection proxy, so every communication with the **Browser Kerne**l is controlled. For every message sent from a **Controlled Process**, the **Reference Monitor** will check the **Domain**'s **Rights** to permit the access. The access control which the **Sandbox** delivers to each **Controlled Process** allows the isolation between different **Domains**. Depending on the manufacturer, a **Plugin** could not be sandboxed.

2.5.2 Dynamics

Some use cases are the following:

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

We show in detail Make Request below. (Figure 3):

Summary

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

Actor

Browser User

Preconditions

The Host must have one or more Browser Kernel for the Browser 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 Browser Kernel and Controlled Processes can be both synchronous and asynchronous [6, 7]. We do not explain in this work synchronization details, because it is out of our scope. We are interested only in specifying who interacts with whom.

1. A Browser User requires a browser to access a URL for some resource in a Provider, this action can be a click in a web page, a Browser's User's request, a redirection from another web page, etc. Usually a Browser Kernel is already instanced in the host. A Browser Kernel also has a Reference Monitor to control the incoming messages from the Controlled Process instances.
2. The Browser Kernel creates (Figure 3) or reuses (Figure 4) an instance of a Web Content Renderer (a Controlled Process subclass), and sets for the new instance the Domain and Rights. A Proxy is created in each Controlled Process instance as a mean

to communicate with the Browser Kernel. If the Controlled Process is reused, a new Proxy is not necessary.

3. Once the Controlled Process is ready, the Browser Kernel creates a Request which is sent to the Provider.
4. The Provider will send a Response to the received Request. Depending on how it is implemented the Browser Kernel, it may or may not have to wait for the response (synchronous or asynchronous) of the Provider.
5. The Response is sent by a communication channel to the Controlled Process of origin, realized it as a Web Content Renderer. If a Response was received for a Request, the Controlled Process is ready to prepare the parsing of the content or use it in a plugin or extension to support the display of the resources obtained by the URL. Otherwise, the Web Content Renderer will create an error page.
6. In case the resource is a web page, the Web Content Renderer obtains a bitmap, a graphic representation of the resource, to be sent to the Browser Kernel, so the Browser User can see it. Even if the resource is not a web page (binary, script, etc.), the Controlled Process sends through its Proxy a Local Request to the Reference Monitor within the Browser Kernel, so the traffic can be controlled before the Browser Kernel shows it to the Browser User or is used for other purposes.
7. If the permissions are sufficient (Domain and Rights the Controlled Process are correct), the Reference Monitor allows the message with the bitmap to be forward to the Browser Kernel.

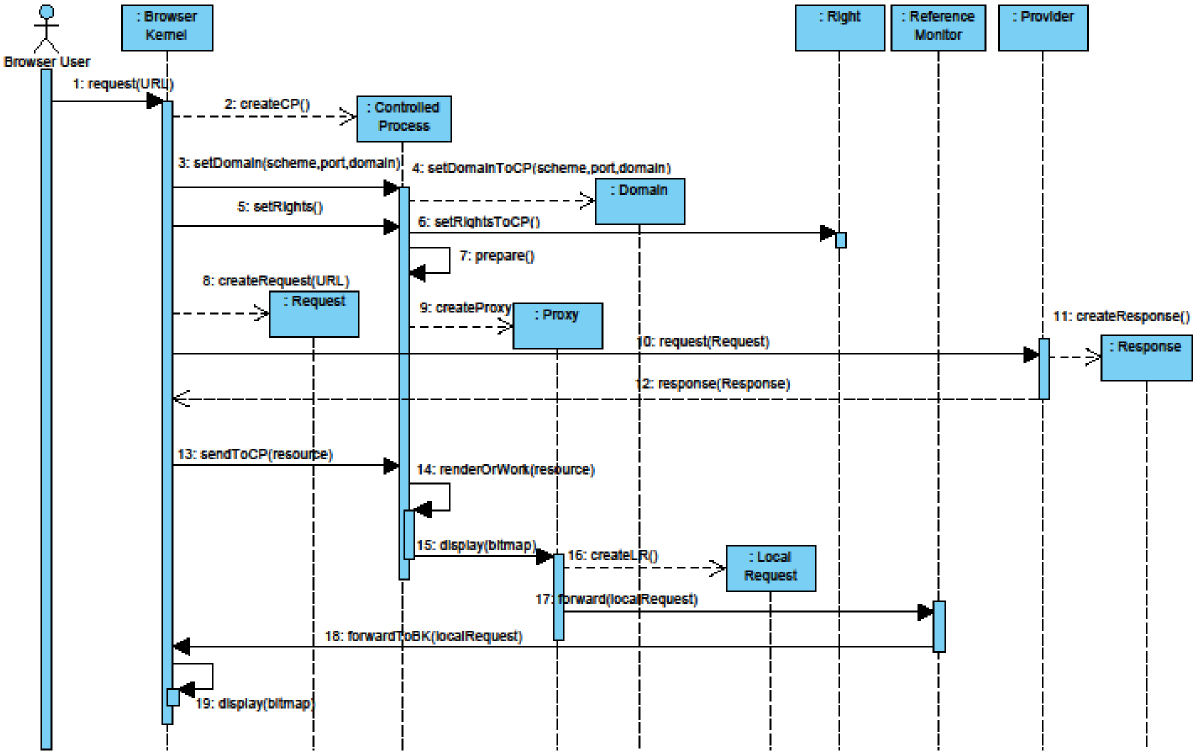


Figure 3. Sequence Diagram: Make Request with a new Controlled Process.

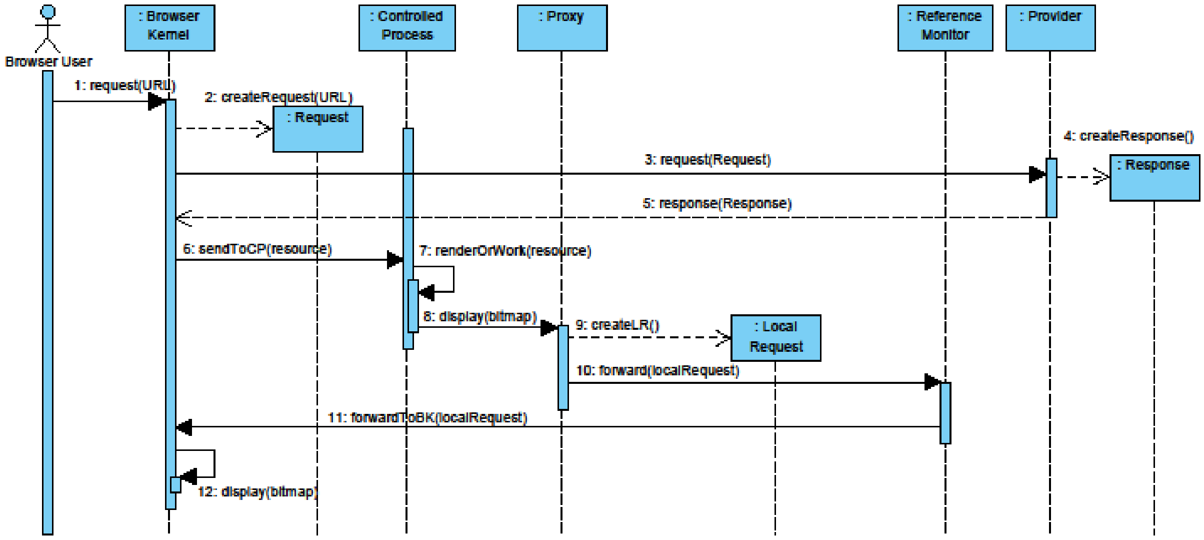


Figure 4. Sequence Diagram: Make Request, while reusing a Controlled Process.

Alternative Flow

- The Provider is not available.
- The resource pointed by the URL does not exists.
- The request is cancelled.

Post conditions

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

2.6 Implementation

- While it is true that a Browser Kernel does not need sandboxing and only creates sandboxed processes or Controlled Process, the concept called Sandbox is defined here as in [5]; in which the Browser Kernel acts as the User that is related to the isolated process.
- The Sandbox may be implemented in various ways. Google Chrome [8] is based on not reinventing the wheel and use the protection mechanisms provided by the OS (i.e. Windows or Linux) of the host to protect the user. This prevents any process to access the file system, and having a restrictive API in the Web Content Renderer. Google Chrome, Firefox and Internet Explorer claim that sandboxes are an important piece to the Browser because realizes the principle of least privilege [9, 8, 10]. Some operating systems provide tools for creating sandbox or in other cases a special library is used (i.g. Google Chromium). To create a sandbox, the minimum configuration includes 2 processes: The privileged process or broker represented by the Browser Kernel and the processes hosted in sandboxes or Targets.
- The Same Origin Policy (SOP) [19] is defined by the origin, and is used to separate different resources by its domain, scheme and port. It is the minimum security mechanism a browser has while requesting cross-origin resources, and divides the different kind of contents so they cannot interfere with each other. To enforce the

Same Origin Policy, Google Chrome, Firefox and Internet Explorer use different schemes [16, 20, 21]; for example, Google Chrome leaves pages/resources isolated with the help of the Renderer (Web Content Renderer in this case).

- The SSL/TLS protocol complements our pattern while giving security for communication channels between the browser and the provider.

2.7 Consequences

The Web Browser Communication 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 Kernel and Controlled Process are independent processes, the failure of one is isolated from the other processes (i.e. crash, memory corruption, etc.).
- *Isolation:* Depending on the type of isolation you can separate the different request, so they do not interfere with each other, unless it is desired.
- *Heterogeneity:* Because each web browser tries to follow the standards of the W3C [11], every page that follows these guidelines can be viewed, as well as other resources.

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 the host's resources are wasted to keep everything open.
- Resources from Providers which do not comply the specifications of the W3C, will be displayed incorrectly by the web browser.

2.8 Example Solved

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 (i.e. the file system). Each Browser User can select the wanted Browser Kernel, because each one is isolated by using separated processes.

2.9 Known Uses

- Currently, the separation of the components of the browser in various processes, with different levels of access, is called as Modular Architecture [12]. 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 [13]. This proposal is used as a reference in the Mozilla project Electrolysis, as you can see in [14, 15], especially 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; [16] addresses a Loosely-Coupled architecture [17] and its components, but without giving further details.
- Firefox meanwhile has two implementations: monoprocess and multiprocess/modular. Electrolysis is the name of the modular architecture being implemented, but it has not yet been fully completed.

2.10 Related Patterns

- The Web Content Renderer pattern, which is under development, represents the subsystem that allows the parsing of a resource obtained through a request. If the obtained resource is a web page, this subsystem will be in charge of obtaining a graphical representation for the user, a bitmap that with host's resources will be displayed in the screen.
- The Browser Kernel pattern, a pattern we are developing, represents the subsystem that represents the Web browser central component. Operations like caching, saving resources to the host, using a host's Networking API and other functions are done in this subsystem.
- The Reified Reference Monitor [5], which describes how to enforce authorization rights when a subject requests access to a protected object or service and returns a decision (response).
- The Sandbox is another name for the pattern Controlled Execution Domain [5].
- For adding security to communications between Client and Server, the TLS pattern in [5] complements our pattern.

3 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 variety of attack vectors, to the user using it as well as the system with which interacts. Therefore, it is important to understand its structure and how it interacts with internal and/or external stakeholders.

A part of our Reference Architecture has been built through the abstraction of documentation through the Web Browser Communication pattern. We created our first architectural pattern for the infrastructure of web browser to help others 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. The reference model obtained in [18] express the type of architecture used in the nineties until 2009 (approximately). However, our proposal represents the current implementation used in browsers, usually called a Modular Architecture. The proposed work allows a better understanding of this system called web browser by using our partial Reference Architecture, this is also helpful to understand existing threats. Also, as it is not subject to specific implementations, it is possible to generalize certain results in other browsers.

Future work to do is finishing the Reference Architecture for web browsers. Other patterns related to Web Browser Communication pattern will be obtained in order to complete the Reference Architecture, such as the Web Content Renderer and Browser Kernel pattern.

We plan to build Misuse Patterns for the Web Browser Communication pattern, to continue the study of the possible threats in the Browser, as a way to educate developers and stakeholders. 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. An example of the type of work to be carried out can be seen in [4].

Acknowledgements

We thank our shepherd, Norihiro Yoshida, for his useful comments that significantly improved the quality of the paper. We also thank Woei-Kae Chen for supervising our paper shepherding. This work was partially supported by CONICYT (grant Fondecyt 1140408).

References

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

[2] E. B. Fernandez, M.M. Larrondo-Petrie, T. Sorgente, and M. VanHilst, "A methodology to develop secure systems using patterns", Chapter 5 in "Integrating security and software engineering: Advances and future vision", H. Mouratidis and P. Giorgini (Eds.), IDEA Press, 2006, 107-126.

[3] E. B. Fernandez, N. Yoshioka, H. Washizaki, J. Jurjens, M. VanHilst, and G. Pernul, “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", Requirements Engineering, Jan 2015. [Online]. Available: http://link.springer.com/10.1007/s00766-014-0218-7

[5] E. B. Fernandez, “Security patterns in practice: designing secure architectures using software patterns”. John Wiley & Sons, 2013.

[6] B. McCloskey, “Multiprocess Firefox". [Online]. Available: https://billmccloskey.wordpress.com/2013/12/05/multiprocess-refox/n#ipc

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

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

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

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

[11] World Wide Web Consortium - W3C, “About" [Online]. Available: http://www.w3.org/Consortium/

[12] T. Vrbanec, N. Kirić, M. Varga,: “The evolution of web browser architecture”. SCIECONF 2013, 472–480 (2013).

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

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

[15] “Features/Security/Low rights Firefox - MozillaWiki" [Online]. Available: https://wiki.mozilla.org/Features/Security/Low_rights_Firefox

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

[17] “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

[18] A. Grosskurth, M.W. Godfrey: “A reference architecture for Web browsers” .
In: 21st IEEE International Conference on Software Maintenance (ICSM’05). pp. 661–664. IEEE (2005).

[19] W3C, “Same Origin Policy," W3C, Web page, 2010. [Online]. Available: https://www.w3.org/Security/wiki/Same_Origin_Policy.

[20] C. Reis and S. D. Gribble, “Isolating web programs in modern browser architectures," Proceedings of the fourth ACM European conference on Computer systems EuroSys 09, vol. 25, no. 1, p. 219, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1519065.1519090

[21] C. Jackson and A. Barth, “Beware of finer-grained origins," Web 2.0 Security and Privacy, 2008. [Online]. Available: http://seclab.stanford.edu/websec/origins/fgo.pdf