A pattern for Web Browser Renderer

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
Boca Raton, Fl, USA
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 security education 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 have developed a Browser Infrastructure pattern that describes the infrastructure to allow the communication between a Web Client and a Server in the Internet, and now we are extended it. In this paper we will identify the component in charge of the rendering of a web resource within the web browser, named here as Web Content Renderer.

Keywords

Browser, Web Client, Browser Renderer, 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 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.

We started to build a Reference Architecture in a student work, and now we are trying to improve it by creating new architectural patterns for our RA, misuse patterns and finding security patterns to build a Security Reference Architecture (SRA). We are currently building a SRA, which is a Reference Architecture where security services have been added in appropriate places to provide some degree of security for a specific system. The basic approach we will use to build a Security Reference Architecture is the application of a systematic methodology from [2, 3, 4], which can be used as a guideline to build secure web browser systems and/or evaluate their security levels. 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 Web Content Renderer Pattern is presented as a part of the process of developing a Reference Architecture (RA) and Security Reference Architecture (SRA) for the Web Browser. We already built a Browser Infrastructure Pattern as our foundation and now we are extending it. The audience to which our paper is focused are Browser developers, Web Application developers and IT students. 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.

2. WEB CONTENT RENDERER PATTERN

Intent

The Web Content Renderer describes the architecture for the processing of web resources obtained by a browser, into bitmaps for displaying them on a peripheral device to the user.

EuroPLoP 2016 July 6-10, Bavaria, Germany.
ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Example

If a user needs to make a bank transaction, such as deposit money to another party, the browser's user will type an URL in the browser to access the online banking service of the bank the user belongs to. But the most important part is the need for displaying the resource obtained to the user.

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 it to the corresponding IP address. It also receives the answers that the users want. Then the web browser needs to parse and interpret the resource to obtain a graphic representation of the resource for the user.

Problem

Internet users need resources from providers/servers and they may be in a specific format, the challenge is to find a way to make them visible on the computer. In this case, if appropiate tools are not available, the resource could not be helpful and it cannot be seen correctly. How can the host provide these functions? The solution to this problem must resolve the following forces:

- *Transparency*: the user should not be concern about how a request is manipulated to get it on the screen.
- *Stability*: The browser must be capable of displaying the resource even if some syntax issues appears, like invalid HTML.
- *Isolation*: Resources can interact with each other while the policies allows them; policies enforce isolation between resources of different domains or **origins**.
- *Heterogeneity*: It does not matter the type of provider with which the browser communicates, it should be possible to interact with whatever type, and it should be capable of showing the content of the obtained resource adequately.
- *Timely*: The obtained resource should be displayed within a reasonable time frame; otherwise a user of the browser could have a bad experience.

Solution

Provide a device, the Web Browser, with the functions needed to understand user requests and send them to the provider. The Browser should be capable of understanding the resource obtained and display it appropriately.

Structure

In figure 1 shows a **Frame Constructor** in charge of building a **Rendered View** of the resource requested by the user and communicating with the **Browser Kernel** The **HTML/XML Parser** is in charge of parsing a web resource and obtain an object representation of it, called a **DOM Tree**. A **CSS Parser** does the same job as the **HTML/XML Parser**, but creates **CSS Style Rules** to change visual aspects of the resource to display. The **DOM Tree** is a dynamic structure that can change anytime while the resource is needed, by using scripting language that is interpreted and executed by the **Javascript Engine**; **CSS Style Rules** can also change in the same way. A **Rendered View** is the visual representation of a resource obtained from the **Browser Kernel**, where a **Render Tree** is the first step to obtain it. The **Render Tree** is a a new dynamic structure that links the **DOM Tree** and **CSS Style Rules** together.

Dynamics

Some use cases are the following:

- Load Resource (actor: Browser Kernel)
- Execute User action (actor: Browser Kernel)

We show in detail Load Resource below. (Figure 2):

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 in charge of sending an obtained resource to the Controlled Process, which is instanced as a Web Content Renderer for displaying the resource appropriately.

Actor

Browser Kernel

Preconditions

The Host must have one or more Browser Kernel 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. The Web Content Renderer must consider that a web resource can be in many different formats, so it should be capable of displaying them accordanly.

Description

1. First, the Browser Kernel sends a resource to the Controlled Process, which is instanced as a Web Content Renderer. The Frame Controlled receives each request for rendering a new resource.

2. The Frame Constructor uses its HTML/XML Parser to and understand the resource. During the parsing process new resources will be needed such as images, plugins, content, etc. where each one of them will do a Load Resource use case. In parallel the tokenization process creates a dynamic structure called as the DOM Tree, which is returned to the Frame Constructor. The same process is done by a CSS Parser to obtain CSS Style Rules.

3. If scripts are found on the resource while parsing it, the Javascript Engine has the job of parsing, interpreting and executing them. This action can change the DOM Tree and CSS Rules Styles already built.

4. Finally, the Frame Constructor is in position for building a visual representation of the resource to be renderer. The DOM Tree and CSS Styles Rules are linked together into a Render Tree, and then a process called Layout and Painting is called to get the bitmap for the Browser Kernel.

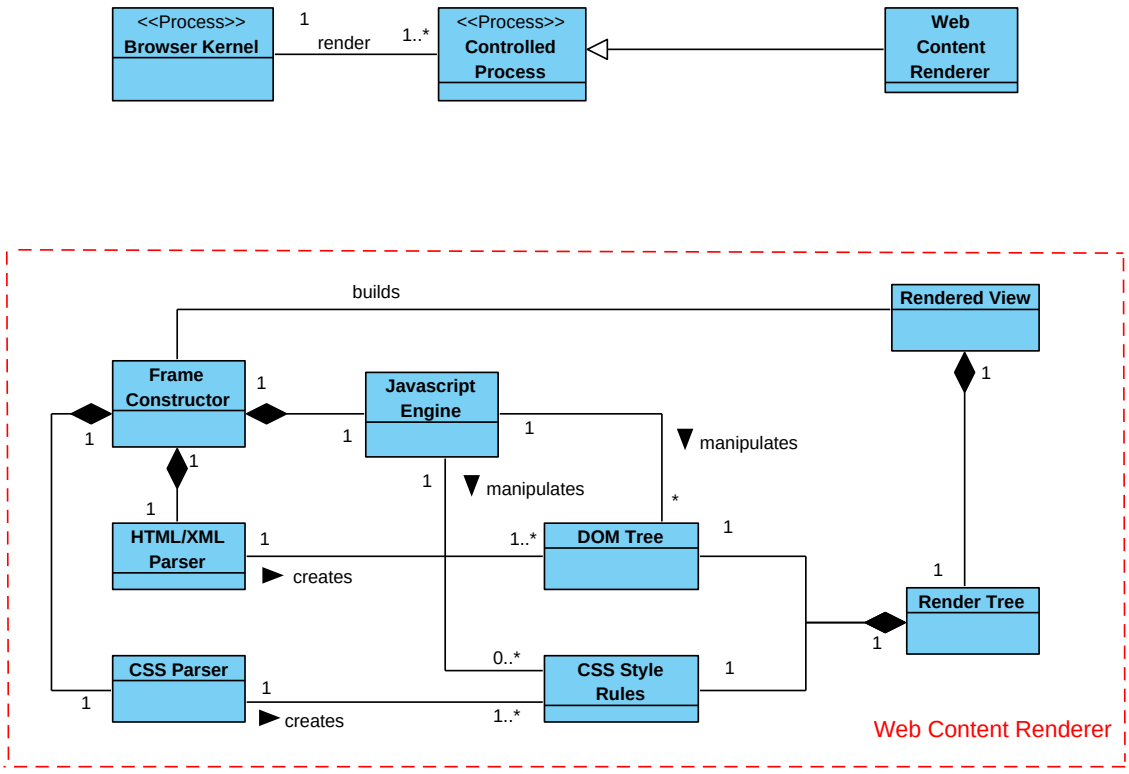

Figure 1: High-level Components of the Browser Infrastructure.

Alternative flow

- The HTML/XML Parser fails in getting the DOM Tree from the resource.
- The resource does not exist, so a **Error page** should be created.
- The execution of Javascript or the rendering process is cancelled.

Postconditions

The Browser Kernel obtains a bitmap representation of the resource.

Implementation

- The Javascript Engine on the Web Content Renderer is normally composed by standard classes, a Javascript Parser that interprets scripting code, a JIT or Just-in-Time Compiler, an Interpreter and a Garbage Collector. Other implementations could have other components, but the must importants are those before [5, 6, 7].
- To enforce the Same Origin Policy, Google Chrome, Firefox and Internet Explorer use different schemes; for example Google Chrome leaves pages/resources isolated with the help of the Renderer (Web Content Renderer in this case).
- Gecko [8] is the name for the Renderer in the mono-process Firefox, it is written in c++ and can be used to render the user interface of other applications such as Thunderbird.
- Trident is the old Renderer used on Internet Explorer [9]. A new Renderer called EdgeHTML, a fork of Trident, is integrated on Microsft Edge, the new browser of Microsoft [10].
- Blink is the implementation of the Renderer done by Google [11].

Consequences

The Web Content Renderer pattern provides the following benefits:

- *Transparency*: All the process for rendering a resource is done without the user interference.
- *Stability*: Even if a resource (HTML for example) is invalid, the rendering process will try to get a visual representation. Other type of resource could be handled the same way or by requesting again for the resource.
- *Isolation*: Resources can interact only if the Same Origin Policy allows it. A Reference Monitor for the DOM Tree and Capabilities for the Javascript Engine are used to enforce this policy.
- *Heterogeneity*: Because each web browser tries to follow the standards of the W3C [12], every page that follows these guidelines can be viewed, as well as other resources.
- *Timely*: Every part of the rendering process will be bounded by time restrictions that allow a nice user experience.

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

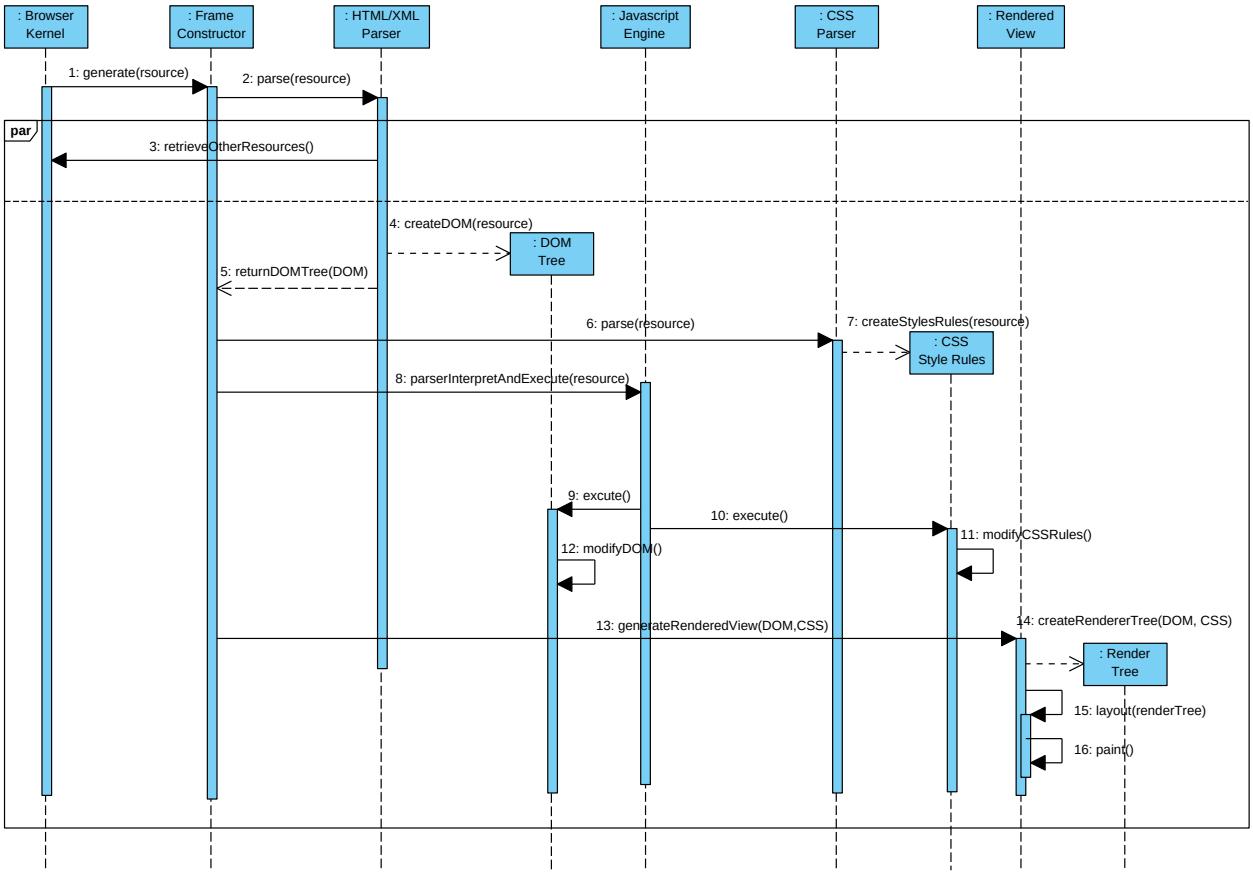

Figure 2: Sequence Diagram: Load Resource.

- Since independent processes are used to create a Web Content Renderer for rendering a resource (depending on the scheme using the browser), it is possible that a lot of the host's resources are used to keep everything open.
- Resources from Providers which do not comply the specifications of the W3C, will be displayed incorrectly by the Web Browser.

Example Resolved

With the given pattern it is now possible to display and navigate smoothly to all resources on the Internet we want. Independent on the type of web resource, it will be possible to display it appropriately to the user it needs.

Known Uses

- Currently, the separation of the components of the *Browser* in various processes, with different levels of access, is called as Modular Architecture [13]. 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 [14].
- Internet Explorer, a proprietary browser, does not give much information about its structure or details of its implementation; [9] addresses Loosely-Coupled architecture [15] and its components, like Trident its Rendering Engine, but without giving further details.

Related Patterns

- The Browser Kernel pattern represents the subsystem that represents the Web browser central component [16].
- The Reified Reference Monitor [17], which describes how to enforce authorization rights when a subject requests access to a protected object or service and returns a decision (response).

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 variaty 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 Browser Infrastructure pattern. We created our first architectural pattern for the infrastructure of *Web Browser* to help others un-

derstand, 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 partially 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 Browser Infrastructure pattern will be obtained in order to complete the Reference Architecture, such as the Web Content Renderer (this paper) and Browser Kernel pattern.

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. 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].

4. 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. 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"*, pp. 107–126, 2006.

[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] Google Chrome, "V8 Javascript Engine." [Online]. Available: https://github.com/v8/v8/wiki{\#}what-is-v8

[6] Microsoft, "Chakra Javascript Engine." [Online]. Available: https://github.com/Microsoft/ChakraCore/wiki/Architecture-Overview

[7] Mozilla, "SpiderMonkey Javascript Engine." [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey

[8] "Gecko:Overview - MozillaWiki." [Online]. Available: https://wiki.mozilla.org/Gecko:Overview

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

[10] "Introducing EdgeHTML 13, our first platform update for Microsoft Edge | Microsoft Edge Dev Blog." [Online]. Available: https://blogs.windows.com/msedgedev/2015/11/16/introducing-edgehtml-13-our-first-platform-update-for-microsoft-edg

[11] "Blink - The Chromium Projects." [Online]. Available: http://dev.chromium.org/blink

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

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

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

[15] "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

[16] P. Silva, R. Monge, and E. Fernandez B., "A pattern for Web Browser Infrastructure," *Proceedings of the 5th Asian Conference on Pattern Languages of Programs (AsianPLoP) 2016, Taipei, Taiwan, February 2016*.

[17] E. B. Fernandez, *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.