A Reference Architecture for web browsers: Part III, A pattern for a Web Browser Kernel.

Paulina Silva and Raúl Monge, Universidad Técnica Federico Santa María
Eduardo B. Fernandez, Florida Atlantic University

A web client like the Web Browser facilitates software development since is quite robust to all possible actions a user may do when interacting with a web application. Since 2000 a number of research proposals for new browser designs had appeared, since the monolithic architecture used before had not considered some important design decisions, like security, stability and others. New Web Browser architecture designs are likely to be based on operating systems properties were several cooperative processes use the same process structure usede ny operating systems to implement system services; the same logic can be applied for the Web Browser. A **Web Browser Kernel** is the representation of the main process, or parent process, in charge of controlling what the browser does, like an operating system. It communicates with other processes, its child processes, and sends instruction to them; of course, according to the **Browser User**'s will. A **Web Browser Kernel** is a piece of software that could be found whenever we are using a **modular architecture** in a Web Browser. The pattern we are developing on this paper has abstracted a part of the internals of the Web Browser, and we show what happen when the user asks for content to be display into the screen of a Web Browser host.

Categories and Subject Descriptors: []: —; []: —; []: —

Additional Key Words and Phrases: Browser, Web Client, Browser Renderer, Reference Architecture, Pattern

ACM Reference Format:

Paulina Silva, Raul Monge, and Eduardo B. Fernandez. A Reference Architecture for web browsers: Part III, A pattern for a Web Browser Kernel HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. X (Month Y), 9 pages.

1. INTRODUCTION

We are currently developing a serie of architectural patterns to build a Reference Architecture (RA) and Security Reference Architecture (SRA) for a Web Browser [Silva et al. 2016b; Silva et al. 2016a; Silva et al. 2016c]. An RA is an abstract architecture that describes functionality without getting into implementation details. Its aim is to provide a guide to build architectures for concrete versions of some system or to extend such system [Avgeriou 2003; Galster and Avgeriou 2011; Ang 2012]. A SRA on the other hand, provides information about possible threats and defense mechanisms of the system. Our intent in describing the Web Browser as a RA and SRA is to understand the symbiosis between the client and the Web Server to which the user asks for resources, and the related security implications. Because we know that a client would not exist without its counterpart, the server, we think that security must be seen and understood from both sides and not only on the server side like most developers do [Alcorn et al. 2014]. We have used patterns in our work because they encapsulate solutions to recurrent problems and define a way to concisely express requirements and solutions, as well as providing a communication vocabulary for designers [Gamma et al. 1994; Buschman et al. 1996]. 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. Based on work in [Fernández and Larrondo 2006], they proposed an approach for building SRAs using patterns, where they defined a precise and semiformal security cloud computing architecture for the complete cloud environment [Fernandez et al. 2016]. They showed that SRAs are useful to apply security to cloud systems and for a variety of other purposes.

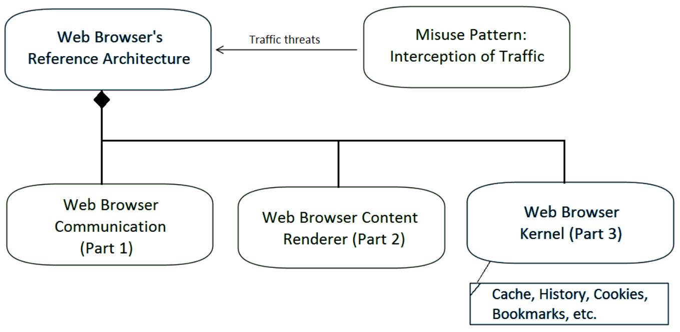
Web Browser's Reference Architecture
Traffic threats
Misuse Pattern: Interception of Traffic
Web Browser Communication (Part 1)
Web Browser Content Renderer (Part 2)
Web Browser Kernel (Part 3)
Cache, History, Cookies, Bookmarks, etc.

Fig. 1: Pattern Diagram of our current work on the Reference Architecture for the Web Browser.

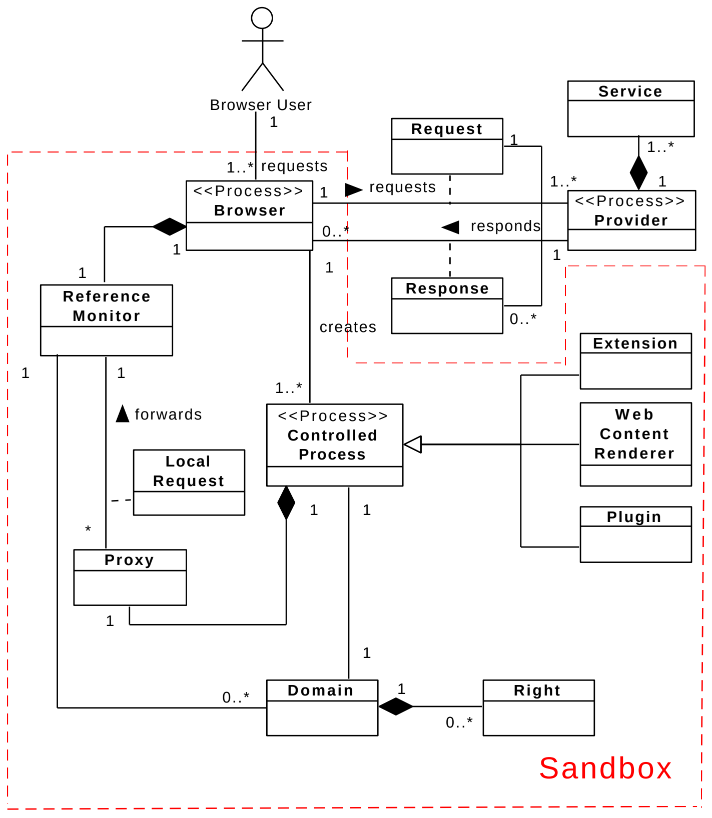

Fig. 2: Overview of the high component for a Web Browser

In previous work [Silva et al. 2016b; Silva et al. 2016c], we use a pattern diagram [Fernandez et al. 2008] (Figure 1) to show the relationships between our patterns and we can see how they relate to each other (rounded rectangles represent patterns and the arcs indicate dependencies between patterns). Figure 2 [Silva et al. 2016b] shows an overview of the high level components of a Web Browser and how they are related with the current pattern, **Web Browser Kernel**.

2. WEB BROWSER KERNEL

2.1 Intent

In its modular architecture, the Web Browser Kernel is the main control point of a browser. It establishes and orchestrates the control and communications channels among the other components of the browser to satisfy the needs of a Browser User (Figure 4).

2.2 Example

When a user launches a browser, a user interface must be displayed and the components subsystems of a browser need to be instantiated and wired together so they are ready to respond to user input and provide the functionality of web browsing.

2.3 Context

A Web Browser is a client which searches for resources on any Web Server connected to the Internet. The Web Browser main control point organizes a set of functional subsystems that collectively implements browsing features. Each Web Browser component communicates via various communication channels that can be configurated as needed. To display requested resources, Web Browser components must have enough allocated systems resources; faults in components can also occur and should be deactivated.

2.4 Problem

A Web Browser is constantly being asked by the Browser User for resources on the Internet. If the Web Browser has a monolithic architecture, it may not be able to satisfy all the requests and sometimes they could fail, making the Web Browser crash or become unusable. How can a Web Browser be prepared for these events?

The solution to this problem must resolve the following forces:

—**Compatibility:** a Web browser should comply with web standards for a correct implementation and display of the content.

—**Compartmentability/Isolation:** A Web Browser should be able to show all the content, even when a page could be broken and have errors. The Web Browser should keep on browsing even after problems arise on a web page, were it should not be reflected on other rendered web pages.

—**Speed:** A web page should be displayed to the Browser User as fast as possible and respond to user's interaction as quickly as possible to ensure a better browsing experience.

—**Security:** The user's and server's information should be kept secure under any circumstances (privacy of data also is included).

2.5 Solution

Since a monolithic architecture concentrates all the work a Web Browser does on a single process, a better solution is to have a modular architecture [Vrbanec 2013; Wu 2014; Barth et al. 2008; Reis et al. 2009], where a process is only in charge of requesting resources and delegates the rendering work to its children (or renderer) processes. The Web Browser Kernel acts as the parent process of the browser and will work with the Web Browser Content Renderer [Silva et al. 2016b] instances to display accordingly the content being asked. Based on the same principles a operating system has like, separation of concerns, compartmentalization, least privileges, the

Web Browser with a modular architecture will be able to contain the propagation of rendering errors and the consequences of unexpected or malicious behaviour.

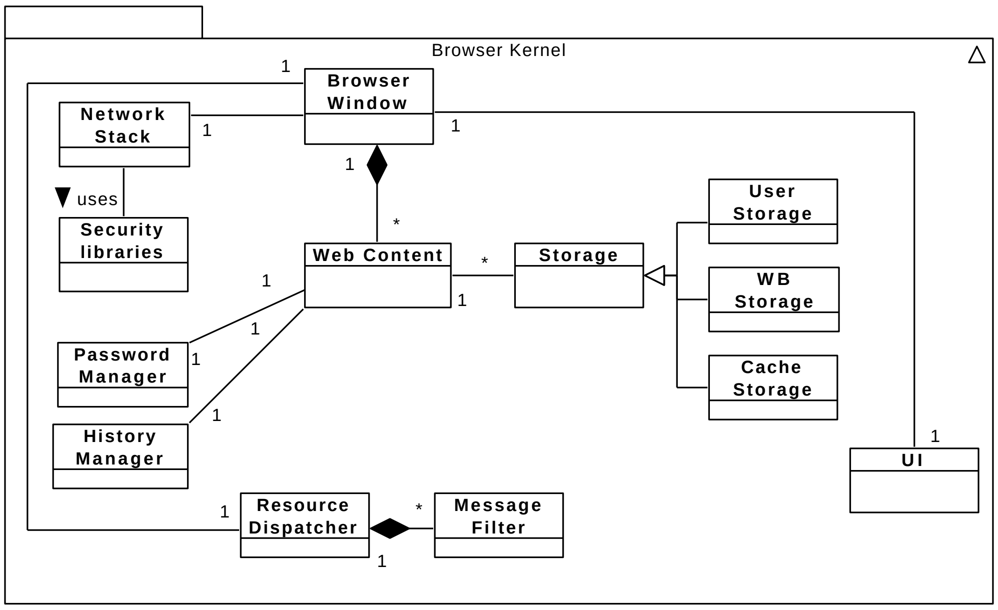

Fig. 3: Pattern for our Web Browser Kernel

2.5.1 *Structure.* Figure 2 shows a class diagram for the **Web Browser Kernel** pattern. The browser host machine receives user's interactions from input devices like mouse or keyboard. When a url is prompted from the **Browser User**, the windows manager of the host machine delegates this action to the **UI** of the Web Browser. The **Browser Window** searches in the **Browser's Storage** for an old version of the page, on the cache. If no old copy of the page from the url being ask is found, an implementation of the **Network Stack** is used to make system calls for the url being asked. **User Storage** is used to complete with user's information in the outgoing request, like adding cookies, etc. **Web Browser Storage** is in charge of storing binaries or files downloaded by the Web Browser. **Security libraries** are implemented on the browser and are used to encapsulate requests to make sure they become secured to the server hosting the resource the **Browser User** wants. The **Password Manager** is in charge of securing old passwords from the **Browser User** and use them in case there are needed. **History Manager** saves all past requests from the **Browser User**, and lets its users modify it at will. When the request is finally encapsulated to make the request to a server, sockets from the operating system will be used to send the request to the Web Server of interest. Once a response to the sent request comes back to the host machine, the **Browser Kernel** will receive it and the **Browser Window** will use the **Resource Dispatcher** to send it it to a new or old **Web Browser Content Renderer** [Silva et al. 2016b; Silva et al. 2016c], thanks to the **Message Filter**.

2.5.2 *Dynamics.* Some use cases are the following:

—Content Retrieval

—Save to storage: cache, Web Browser data or user data

—Manage bookmarks

—Monitor Web Browser

—Receive user input

—Send content to render

We show in detail Content Retrieval below, since is the most important use case for the Web Browser.

2.5.3 *Summary.* The Browser User asks for the content indicated by the URL typed on the keyboard and pushes enter to access the resource, or interacts with an already loaded resource on the Web Browser.

2.5.4 *Actor.* Browser User

2.5.5 *Preconditions.* The host machine must have one or more **Web Browser Kernels** for the **Browser User**, and should be connected to a network or the Internet. The provider the user wants to contact must also be available.

2.5.6 *Description.* The **Web Browser Kernel** is the representation of the main process, or parent process, for a Web Browser with modular architecture. When a **Browser User** surfs through the Internet, the **Web Browser Kernel** receives through the Web Browser's host machine the necessary input, like user interface interaction through the keyboard or the mouse.

(1) A **Browser User** that wishes to retrieve content, interacts with the Web Browser and this interaction is received by the interface provided by the **Web Browser Kernel's UI**, which receives in this case the url of the page the **Browser User** wishes to see.
(2) The **Browser Window** is in charge of coordinating the **Web Browser Kernel** process and receives input from users with the help of its UI.
    (a) Before searching the content on the Internet, the Web Browser will search on its cache if a recent data on the requested url has been saved. If the content is available and up to date, the Web Browser will render it. Go to step 3.
    (b) If the Web Browser did not find anything for the asked url.
        i. The **Browser Window** will ask the **Browser Storage** for the **Browser User**'s credentials if needed and will ask the **Network Stack** for a DNS search for the url.
        ii. If a record is found for the url asked, a TCP connection is made to the **Network Stack**; including the necessary credentials for the request . Depending on the request, **Security Libraries** could be used on the request, to encapsulate data and send it securely.
        iii. When the **Network Stack** sends the request, if a response is received, this is given to the **Browser Window**.
(3) Wherever the response came from, let be from the cache or a up-to date server's response, the **Browser Window** will send the response to the **Resource Dispatcher**. The latter is in charge of sending the soon-to-be rendered page, but before doing it the dispatcher uses its **Message Filter** to send it to the right **Controlled Process** [Silva et al. 2016b].
(4) After the Controlled Process, instantiated as a Web Browser Content Renderer, sends the bitmap for the resource obtained by the request from the Browser User, the Browser Window will save the rendered web page.
(5) Finally, the Browser Window sends to the User Interface the bitmap to be shown by the display of the Browser User's machine.

2.5.7 *Alternative Flows*

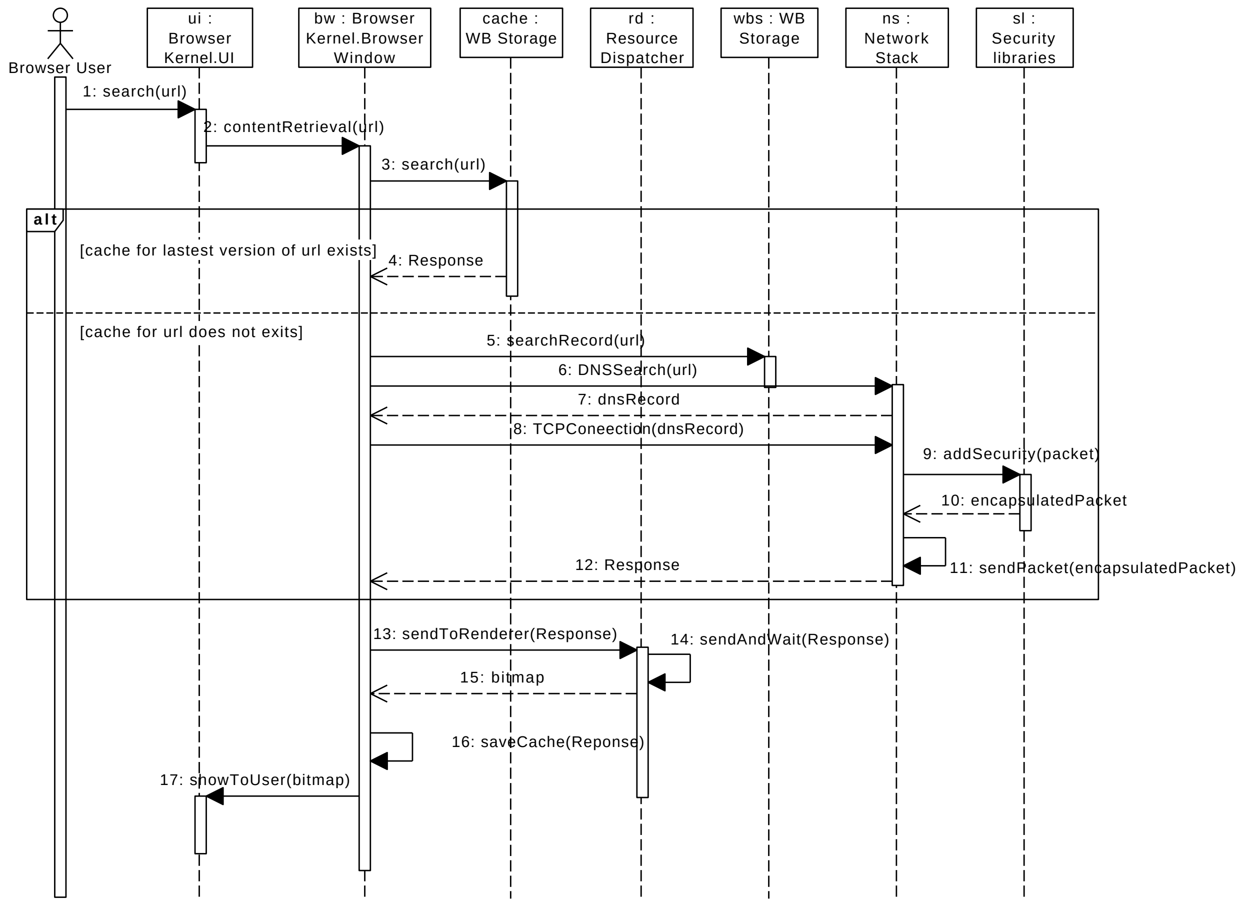

Fig. 4: Dynamics for: Content Retrieval

—The resource obtained is not a web page, it could be a binary or a file. In this case, instead of rendering the response, the Web Browser only downloads it.

—The resource pointed by the URL does not exists.

—The request is cancelled.

2.5.8 *Postconditions.* 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

—The Same Origin Policy (SOP) [Zalewsk 2008] 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 kinds of contents so they cannot interfere with each other. To enforce the Same Origin Policy, Google Chrome, Firefox and Internet Explorer use different schemes [Barth et al.

2010; Grier et al. 2008] [add more references];for example, Google Chrome leaves pages/resources isolated with the help of the Renderer. The Resource Dispatcher with the help of its Message Filter does the job of receiving correctly different requests from the Controlled Processes instantiations.

—The SSL/TLS protocol complements this pattern while providing security (using the Security Library class) for communication channels between the browser and the provider. The Security Library class in our pattern makes this happen.

2.7 Consequences

The Browser Kernel pattern provides the following benefits:

—**Compatibility:** By using Web Standards a web developer assures that the content is shown as she expected.

—**Compartmentability/Isolation:** Since a modular architecture is used, web pages are rendered through children **Controlled Processes** which are **Web Browser Content Renderer** instantiations [Silva et al. 2016c]. Even thought child process can interact through the **Web Browser Kernel**, all the rendering is isolated in the child and will not interfere with other children and the main processes.

—**Speed:** Since the work is distributed through the **Web Browser Kernel** and the **Web Browser Content Renderer** instantiations, speed is increased if compared with a monolithic architecture.

—**Security:** Browser User's Information remain only accessible to the **Web Browser Kernel** only. If needed to surf on the Internet, it will be only used by the main process. In a way that the main process is isolated from its children processes or **Web Browser Content Renderer** instantiations, through the calls that are allowed to enter to the main process; the **Message Filter** is in charge of this.

This pattern has the following liabilities:

—Resources from Providers which do not comply the specifications of the W3C, will be displayed incorrectly by the Web Browser.

—Session Cookies that will be stored on the Browser Storage will be the Achilles heel for the Web Browser. Even if they provide us with a stateful communication, they can affect tremendously the Browser User and the Provider on the Internet by becoming attack vectors for XSS or CSRF attacks, whenever a Web Browser has exploitable vulnerabilities or bad implemented functions.

2.8 Example Resolved

With the pattern of the modular architecture for the Web Browser here presented, the browser will posses a more stable, fast and secure architecture, which will let show all the Browser User's requested web pages. This is possible because the Web Browser will not break if one page could not be displayed as instructed or will not be paused completely if a Web Server is not answering. As we have defined before, the Browser Kernel is essentially the controller of the browser, and delegates the hard work - rendering - to its children processes.

2.9 Known Uses

—Google Chrome is based on a modular architecture, where a Browser Kernel acts as the main process of its Web Browser Chrome and Chromium (Open Source) [Barth et al. 2008].

—Internet Explorer and Edge are proprietary browsers, which do not give much information about their structure or implementation. [Crowley 2010; IE8 2008] address the Loosely-Coupled architecture of Internet Explorer which is also a modular architecture.

—Firefox [Brinkmann 2015]

2.10 Related Patterns

—The Web Browser Communication pattern presents the components of the Web Browser and how they communicate with each other when a resource is requested [Silva et al. 2016b].

—The Web Browser Content Renderer pattern describes at a high-level the components of a web renderer of a Web Browser. It is in charge of composing and obtaining bitmaps of the requested web resources [Silva et al. 2016c].

—The Reified Reference Monitor [Fernandez 2013], describes how to enforce authorization rights when a subject requests access to a protected object or service and returns a decision (response).

2.11 Conclusion and Future Work

When building web applications that connects users to services by using web browsers, some developers do not understand the symbiosis between the web client or browser and the web servers to which resources are being requested. A Web Browser is a complex software and web application developers need to understand the basics of how a Web Browser works, what components make this web client and the interactions between them inside the Web Browser, and the mechanism involved in: (1) the communication with the Web Browser and Web Server and (2) how a web page is rendered. Our aim with this and our previous work is to make understandable the internals of the browser and the already said mechanisms by using architectural patterns to construct a Reference Architecture (RA) for web browsers. Our RA has been formulated with the **Web Browser Communication** pattern, the **Web Browser Content Renderer** and now the **Web Browser Kernel**. These three patterns abstract the infrastructure of a Web Browser to help others understand holistically the components, interactions and relationships of this system called Web Browser. We plan to build more patterns for the Web Browser, as a way to educate developers and stakeholders of web applications. For basic Web Browser-to-server communication, the browser needs the basic tools to: obtain, translate and paint the contents a browser user wants to see, otherwise is not what we would call a "Web Browser". The Web Browser Kernel pattern summarizes the basic components a Web Browser has and use when trying to comply requests, as well as is introduced as the main control point for all the Web Browser user's actions.

REFERENCES

2008. IE8 and Loosely-Coupled IE (LCIE) - IEBlog - Site Home. (2008).

2012. A framework for analysis and design of software reference architectures. *Information and Software Technology* 54, 4 (April 2012), 417–431. DOI:http://dx.doi.org/10.1016/j.infsof.2011.11.009

Wade Alcorn, Christian Frichot, and Michele Orrù. 2014. *The Browser Hacker's Handbook*. John Wiley & Sons.

Paris Avgeriou. 2003. Describing, Instantiating and Evaluating a Reference Architecture: A Case Study. *Enterprise Architect Journal* 342, 1 (2003), 347. DOI:http://dx.doi.org/10.1097/MAJ.0b013e3182314ba8

Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. 2010. Protecting Browsers from Extension Vulnerabilities. *Ndss* 147 (2010), 1315–1329. DOI:http://dx.doi.org/10.1111/j.1365-2486.2006.01169.x

Adam Barth, Collin Jackson, Charles Reis, TGC Team, and others. 2008. The Security Architecture of the Chromium browser. (2008).

Martin Brinkmann. 2015. The state of multi-process architecture in Firefox. (2015). https://www.ghacks.net/2015/05/02/the-state-of-multi-process-architecture-in-firefox/

Frank Buschman, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. A system of patterns: pattern-oriented software architecture. (1996).

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

Eduardo Fernández and María Larrondo. 2006. Security Patterns and Secure Systems Design. June (2006), 1–12.

Eduardo B. Fernandez. 2013. *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

Eduardo B. Fernandez, Raul Monge, and Keiko Hashizume. 2016. Building a Security Reference Architecture for Cloud Systems. *Requir. Eng.* 21, 2 (June 2016), 225–249. DOI:http://dx.doi.org/10.1007/s00766-014-0218-7

Eduardo B. Fernandez, G?nther Pernul, and Maria M. Larrondo-Petrie. 2008. Patterns and Pattern Diagrams for Access Control. In *Trust, Privacy and Security in Digital Business*. Springer Berlin Heidelberg, Berlin, Heidelberg, 38–47. DOI:http://dx.doi.org/10.1007/978-3-540-85735-8_5

Matthias Galster and Paris Avgeriou. 2011. Empirically-grounded Reference Architectures: A Proposal. (2011), 153–157.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design patterns: elements of reusable object-oriented software*. Pearson Education.

C. Grier, Shuo Tang Shuo Tang, and S.T. T King. 2008. Secure Web Browsing with the OP Web Browser. *2008 IEEE Symposium on Security and Privacy (sp 2008)* Sp (2008), 402–416. DOI:http://dx.doi.org/10.1109/SP.2008.19

Charles Reis, Adam Barth, and Carlos Pizano. 2009. Browser Security: Lessons from Google Chrome. *Commun. ACM* 52, 8 (2009), 45–49. DOI:http://dx.doi.org/10.1145/1536616.1536634

Paulina Silva, Raúl Monge, and Eduardo B. Fernandez. 2016a. A Misuse Pattern for Web Browsers: Interception of traffic. *Proceedings of the 5th Asian Conference on Pattern Languages of Programs (AsianPLoP) 2016, Taipei, Taiwan* (2016).

Paulina Silva, Raúl Monge, and Eduardo B. Fernandez. 2016b. A Reference Architecture for web browsers: Part I, A pattern for Web Browser Communication. *Proceedings of the 5th Asian Conference on Pattern Languages of Programs (AsianPLoP) 2016, Taipei, Taiwan* (2016).

Paulina Silva, Raúl Monge, and Eduardo B. Fernandez. 2016c. A Reference Architecture for Web Browsers: Part II, a Pattern for Web Browser Content Renderer. In *Proceedings of the 21st European Conference on Pattern Languages of Programs (EuroPlop '16)*. ACM, New York, NY, USA, Article 27, 10 pages. DOI:http://dx.doi.org/10.1145/3011784.3011813

Tedo Vrbanec. 2013. The evolution of web browser architecture. (2013), 472–480.

Xin Wu. 2014. Secure browser architecture based on hardware virtualization. *International Conference on Advanced Communication Technology, ICACT* (2014), 489–495. DOI:http://dx.doi.org/10.1109/ICACT.2014.6779009

Michal Zalewsk. 2008. Browser Security Handbook, part 2. (2008).