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 browsers 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 independent of 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 Browsers.

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 developers 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 Secure Reference Architecture for the Web Browser. Threat analysis and security patterns were done in our previous student work (graduation report), and we will improve the architecture in the constrution of the SRA.

Browser Infrastructure Pattern

Intent

The Browser Infrastructure Pattern allows the request of a web resource in the Internet to 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 request is made.

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

Example

Within the Host it is possible a lack of resources that a Host user may need. 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 o they should be presented in the screen of the computer to be visualized. In this case, if an appropiate tools is not used, 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 worried 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 rquests, a Sandbox hosting each Web Content Renderer (WBR) instance created will allow the browsing and visualization of the resource obtained. 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 (Figura 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 not specify in great detail, because what matters in this work will be the origin and destination of the message (is not within the scope to see synchronization).

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

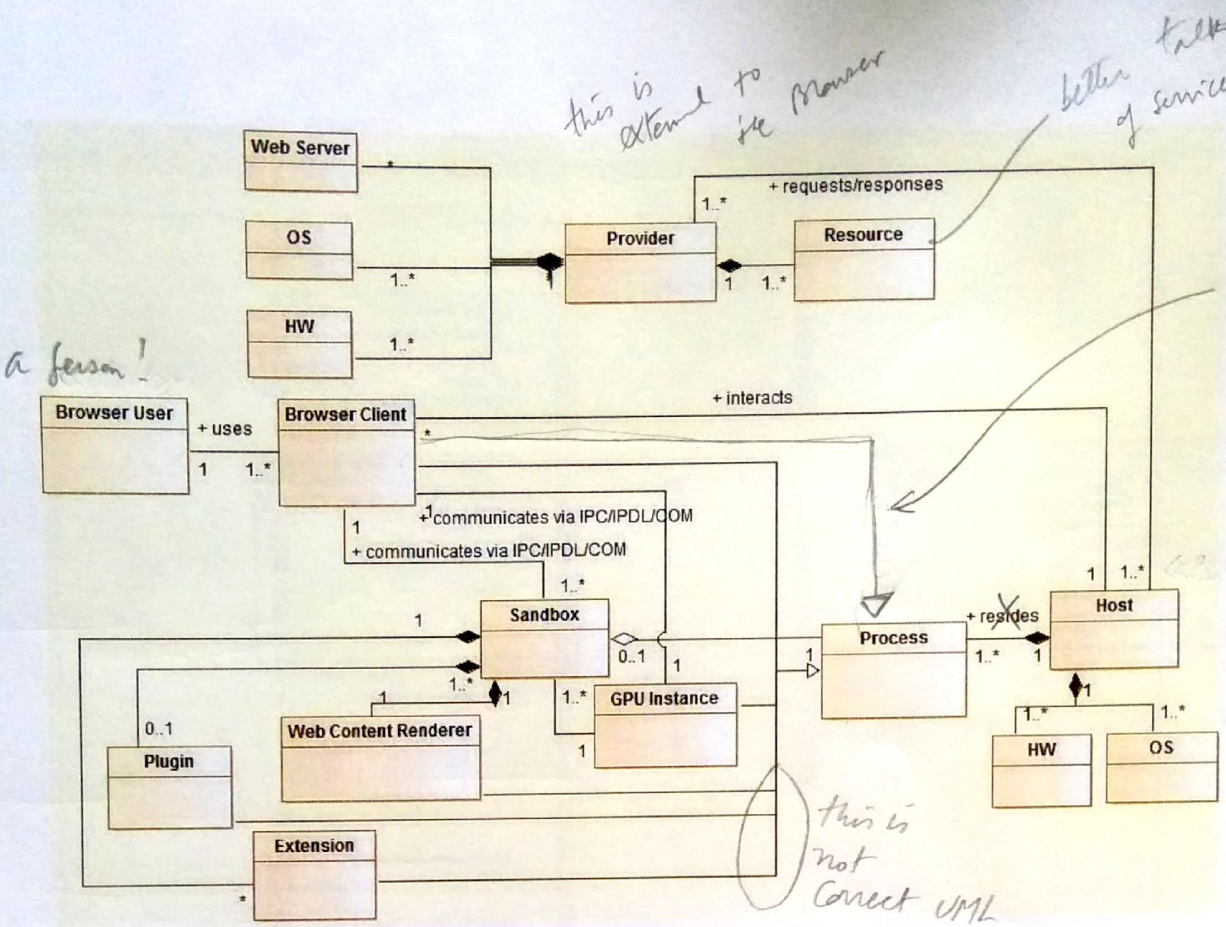

Figure 1: High-level Components of the *Browser*

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 directed to do other way so.

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 website 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, H 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 is displayed by the peripheral device output to the Host user.

Implementation

- The sandbox may be implemented in various ways. Google Chrome [7] is based on not reinventing the