

# Rich client web applications: the future so near

**Abstract:** The come of new web development patterns has raised new possibilities for web applications. AJAX, the enabling technology for these new patterns, is a consolidated approach for the web development, and new tools and frameworks to simplify its use are developed continually. With these new techniques it is possible to develop a new generation of web applications, that are near to the Software as a Service pattern. An example are dynamic web pages that are quite similar to complex client application, that enables the use of a server, or even of multiple servers, only to fetch data, that are then shown and elaborated according to the application logic embedded into the web page. This text examines these new possibilities, the technologies to realize them exploring also technical limits and security issues and, finally, presents a sample application based on them.

## Preface

This text is the result of a work of research, design and implementation aimed to expose the possibilities given by the new strategies in web development. The final purpose is to show a way that promise to grow fast and that will change the actual vision of web applications and of whole computer applications. The work is not exhaustive nor complete, but could be a good starting point for a more deep analysis.

Author information: Roberto Bifulco, [oltremago@gmail.com](mailto:oltremago@gmail.com) [www.robortobifulco.it](http://www.robortobifulco.it)

This work and its author are made in Italy.

## Table of Contents

Preface.....	2
Introduction.....	3
Chapter I	
The evolution of the web applications.....	4
Chapter II	
AJAX.....	6
II.1Data serialization.....	7
II.2AJAX tools.....	9
Chapter III	
Security.....	10
III.1Attack Types.....	10
III.1.1Cross Site Scripting (XSS).....	10
III.1.2Cross Site Request Forgeries (CSRF).....	11
III.1.3JSON Hijacking.....	12
Chapter IV	
The need for RPC.....	13
IV.1RPC requirements.....	13
IV.2GJPR.....	14
Chapter V	
A simple chat application.....	15
V.1Architecture.....	15
V.2The polling problem.....	16
Bibliography.....	18

## Introduction

Web applications changed a lot in the last decade. The starting point was a simple static page, written in HTML, to show texts and images. A fast evolution brought us to dynamically generated pages (through CGI or server side scripting languages) that permits to create personalized content for user's needs, and so, the creation of e-commerce applications and so on.

The problems arised in these cases are linked to the too simple interface that a web page can offer, that makes simple desktop-like operations a far mirage for on-line web applications. Moreover the user interaction with the web application is a simple click-wait-reload action, that causes long waiting periods. Some tries were made during the years to solve this problems, the more of them uses browsers' plugins that need to be downloaded and added to the browser in order to make possible some enhancements in the interface. This solution has a big problem, for a strange reason, when a user tries to use a web site that envelop an object handled by one of this plugins, his browser fires one of the next possible advices:

- You don't have a plugin to run the page's content
- Your plugin is not up-to-date, please download the last version

The strange is in the fact that these advices appear more frequently than expected. The reality is quite simple, user downloads one time the plugin, but developers use, during months, different versions of that plugins for their applications, this makes the user's version not good. For expert users this is not a big problem, but for the others this could bring to not use that web site.

This work presents solutions to avoid these problems and techniques to bring a new evolution in web development, following a trend already started with the great work of some organizations, like Google, that makes applications (software) much more similar to services than to products.

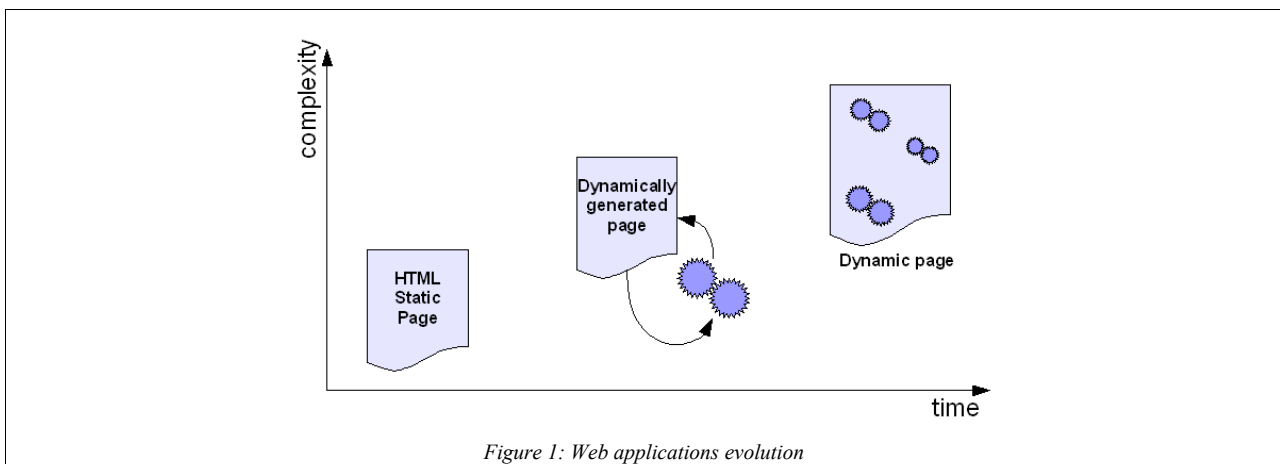
This work is organized in 5 chapters:

- **Chapter I:** A brief introduction to the web technologies evolution and on the future trends and research topics;
- **Chapter II:** The description of the technology that has revolutionized the web development, AJAX, and of linked issues;
- **Chapter III:** In this chapter are presented some of the most known security threats for web applications and their possible solutions;
- **Chapter IV:** To provide complex web applications there is the need for an RPC framework, here are analyzed the requirements for such framework and possible implementations;
- **Chapter V:** For sample purpose, here is presented a simple chat application realized through the presented technologies.

## Chapter I

### *The evolution of the web applications*

As already said in the introduction, web applications had a large evolution in the past years. *Figure 1* presents a schematic view of this evolution. During years, web applications passed from static pages, to dynamic ones. This evolution brings growing functionalities into web applications but also growing complexity for their realization.



The growing in complexity is the result of the different developing patterns: static pages are simple text, formatted through tags; dynamically generated pages are the output of a server side running program, that handles the HTTP requests and generates the appropriate responses. Dynamic pages are something more, they act as an active part in the delivery of the service, something like a complex client-server application. This approach requires that the design phase takes into account both the client side and the server side, making them the components that must interact to delivery the service. Just to clean any doubt, the key difference between dynamically generated pages and dynamic pages is that the first are a simple text-like interface to call functions on the server-side program, the latter contain itself part of the program logic, modifying its content without interaction with server and making requests to server only to fetch data.

With dynamic pages an entire new field of applications can be prospected. The old development patterns, delegating all the program logic at server side, were strictly linked to work with only one server, that was responsible for the execution of the application logic and for the fetching of the data. Dynamic pages, relying on their “embedded logic”, can use different servers to fetch data and combine them into a new rich application that enhances the user experience. This new approach, called “*mash up*” seems to be the next evolution of the web development.

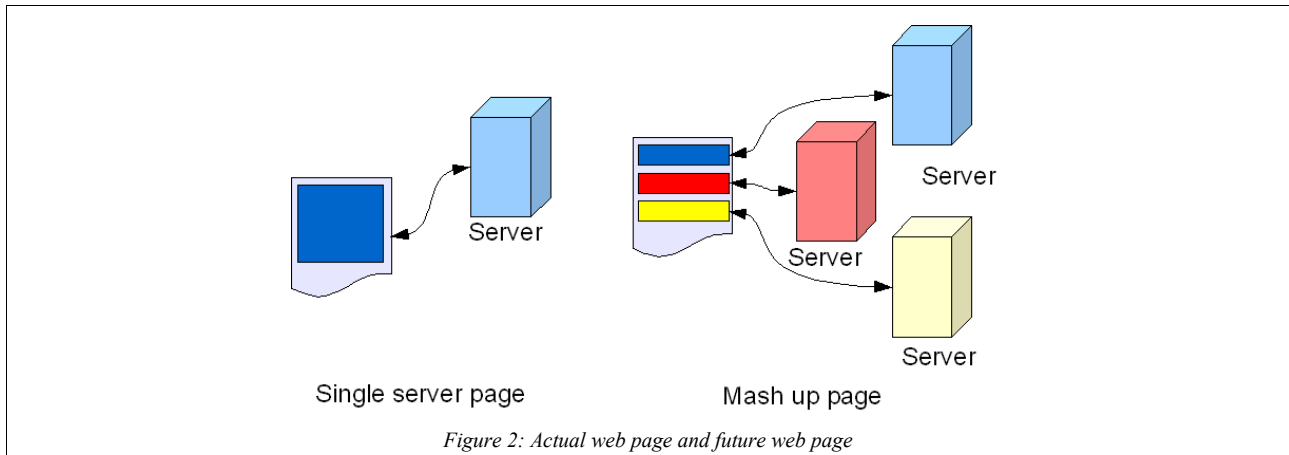


Figure 2 shows this new approach in opposite to the old one: a page is built up using data from different servers, that are mashed through the page active logic. Actually it is not so simple to implement a page like this, some technical issues limit the possibilities of what a web page can do. While it is actually possible that the page makes asynchronous multiple requests for data fetching, there is the bigger limitation of the *Same Origin Policy*, that force a web page to make requests only to the server from which it was loaded [6]. This limitation was introduced for security reasons, in order to avoid unexpected malicious use of web page's active content (We will return on this topic later). How to avoid or modify this browsers' limitation is an open research topic.

Just to give an example of the new possible applications, we can imagine a travel reservation application that build, starting from information from hotels and travel obtained form specific companies web servers, a set of reservation for the travel and for the hotels, according to user needs.

This new technologies seem to give added value to the new ideas about the software deployment. The actual trend is to imagine the software as a service (*SaaS*). With dynamic pages *SaaS* becomes a really near reality, because dynamic pages are software that is deployed through the net, with user transparency. With new web applications software becomes a product given on demand, something like what happened in the first decades of the 20<sup>th</sup> century with electric power for factories.

Instead of having the need to buy and deploy software to get a service, with *SaaS* users buy a service for whom the software is automatically deployed in a transparent way.

## Chapter II

### AJAX

The technology that permits the new development patterns presented in the previous chapter is AJAX. AJAX which stands for Asynchronous JavaScript and XML, is, using the *Wikipedia* definition:

*“a group of inter-related Web Development techniques used for creating interactive Web Applications”*

Javascript is a browser script language, that was born to give some simple dynamic functionalities to HTML pages. This language uses the *Document Object Model* to represent web documents and an object model to interact with the execution environment (the browser)[9]. Some years ago Microsoft introduced a new object handled by the browser, that now is known as XMLHttpRequest, to make asynchronous HTTP request to handle XML format file, in order to update page's content (And so the name AJAX).

Google, through its big work on on-line software, gave a great example of using AJAX with Gmail, Docs&Spreadsheet, etc. From that moment, AJAX gets great attentions and now almost all browsers support the XMLHttpRequest object.

The bigger AJAX advantage is the possibility to fetch data to update only a part of a web document, Before AJAX web applications were based on the *click-wait-refresh* model, where to update a small page content (for example a form button) a complete page reload was needed. AJAX, on the other hand, grants the possibility to make a partial reload of a page in an asynchronous way, fetching only the page's part that needs an update. So, a single page is loaded just one time and then updated through a sequence of asynchronous requests, reducing drastically the user's waiting time.

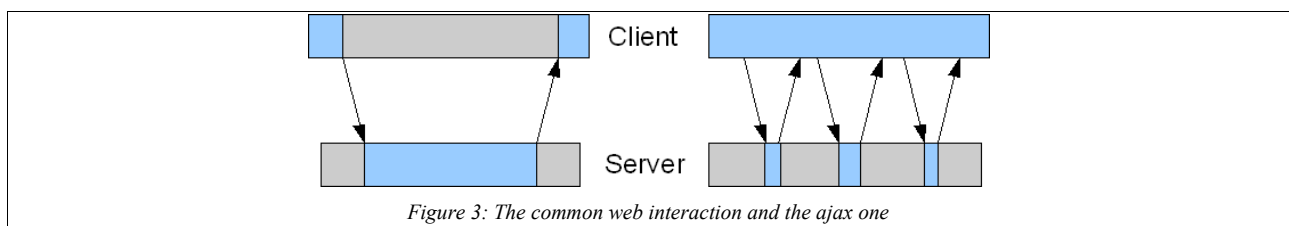


Figure 3: The common web interaction and the ajax one

In *Figure 3* is shown the common web client-server interaction and the ajax one: in the first a user's request is followed by a long waiting period due to the whole page reload; in the latter the user is always active because he is using an interactive dynamic interface, where requests to server are sent behind the scene.

A first look may raise a question about the overhead that would bring a sequence of requests with small data in them, but if we consider that the same application in the common web paradigm needs the resend of the entire page at each request, we can persuade ourselves of the advantages also on

the net traffic side.

### II.1 Data serialization

The biggest feature of AJAX is the possibility to fetch simple data that page's scripts organize for the presentation, cutting all the overhead linked to the formatting information that are embedded in HTML texts. Moreover, if the future is the building of rich client applications, the requested informations are not text, but a mix of different formats like numbers, strings, dates, and so on.

For this reason a serialization language is needed. In the beginning the serialization language was XML, and it is widely used yet. While XML is a great meta-language, it has two big downsides:

- ➔ the great number of meta-information characters that in a small request could overcome the real information characters;
- ➔ the need to implement an XML parser through Javascript, that as every interpreted language, pay a huge price in terms of elaboration time and resource usage.

These downsides are enough to search a new serialization language that actually is the JSON language.

“JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others”[10].

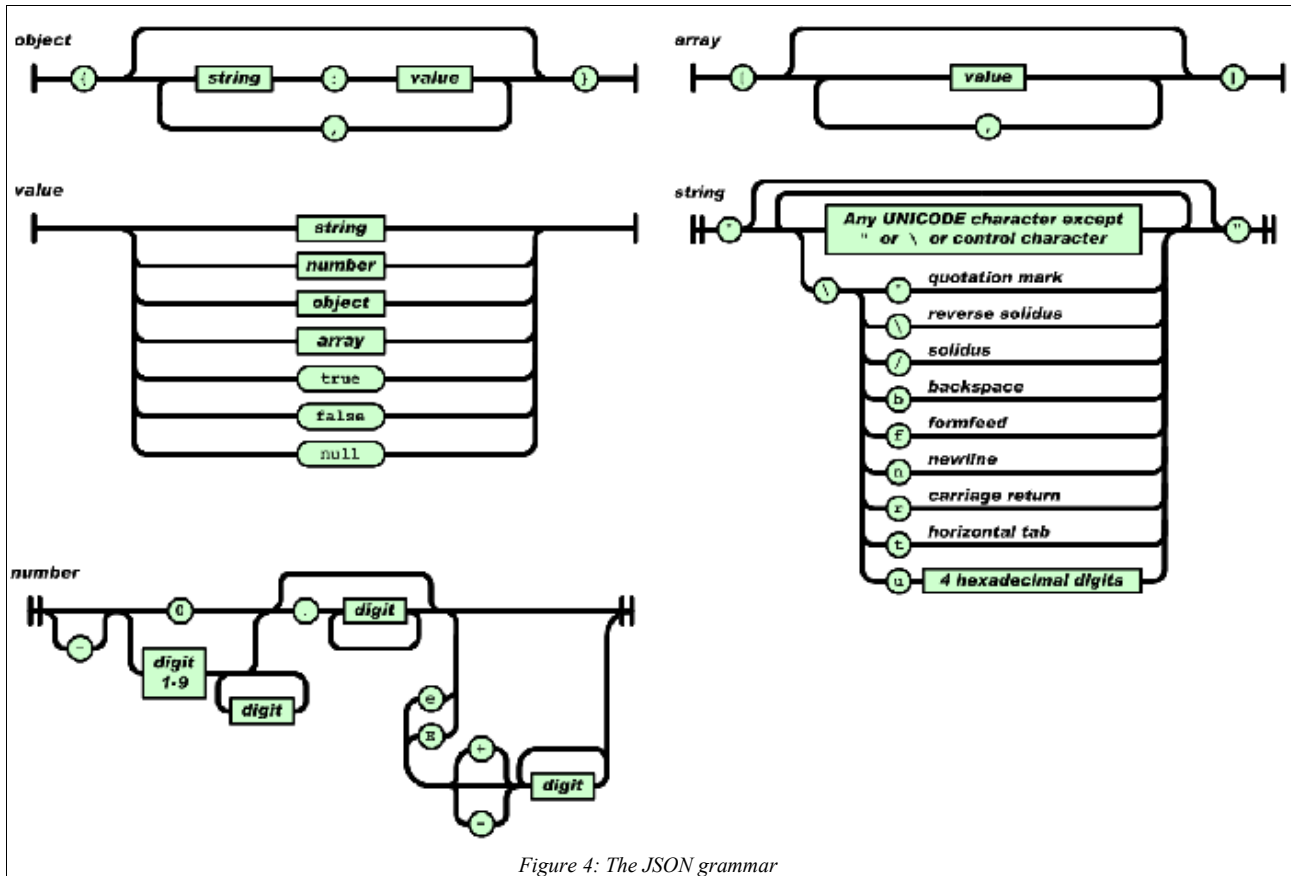


Figure 4: The JSON grammar

The great advantage of JSON is that uses a subset of the Javascript language, in particular, JSON share the Javascript data structure representation. This means that the parsing of a JSON string can be done through the native-language written Javascript browser's parser, with a really high performance.

Some of the most famous AJAX applications use JSON for data serialization, an example are the already mentioned Gmail and Documents&Spreadsheets, but also Yahoo groups and others.

An example of JSON encoded text is the next:

```
{ "widget": {
  "debug": "on",
  "window": {
    "title": "Sample Konfabulator Widget",
    "name": "main_window",
    "width": 500,
    "height": 500
  },
  "image": {
    "src": "Images/Sun.png",
    "name": "sun1",
    "hOffset": 250,
    "vOffset": 250,
    "alignment": "center"
  },
  "text": {
    "data": "Click Here",
    "size": 36,
    "style": "bold",
    "name": "text1",
    "hOffset": 250,
    "vOffset": 100,
    "alignment": "center",
    "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
  }
}
}}
```

The same text expressed as XML:

```
<widget>
  <debug>on</debug>
  <window title="Sample Konfabulator Widget">
    <name>main_window</name>
    <width>500</width>
    <height>500</height>
  </window>
  <image src="Images/Sun.png" name="sun1">
    <hOffset>250</hOffset>
    <vOffset>250</vOffset>
    <alignment>center</alignment>
  </image>
  <text data="Click Here" size="36" style="bold">
    <name>text1</name>
    <hOffset>250</hOffset>
    <vOffset>100</vOffset>
    <alignment>center</alignment>
    <onMouseUp>
      sun1.opacity = (sun1.opacity / 100) * 90;
    </onMouseUp>
  </text>
</widget>
```

Other examples are in <http://www.json.org/example.html> .

## II.2 AJAX tools

The growing interest in AJAX development brings to the flourish of tools and framework to simplify the developers work and augment their productivity. One of the most interesting of this tools is the *Google Web Toolkit* (GWT).

The *Google Web Toolkit* tries a visionary approach to AJAX development: write Java code and compile it in Javascript code for browser execution. This approach grants to Java developers to use their preferred tools and techniques to develop the application that will be converted in Javascript through the GWT compiler. Moreover, GWT provides a set of widgets (Panel, Menus, Buttons, Trees, etc.) to build quickly an interactive web page. There is another value in GWT, Google provides with it a simple RMI framework, based on JSON serialization, to make remote requests to Java servlets. The development cycle, as described on GWT site, is the next:

1. *Use your favorite Java IDE to write and debug an application in the Java language, using as many (or as few) GWT libraries as you find useful.*
2. *Use GWT's Java-to-JavaScript compiler to distill your application into a set of JavaScript and HTML files that you can serve with any web server.*
3. *Confirm that your application works in each browser that you want to support, which usually takes no additional work.*

The key value of a tool like GWT is that it gives the possibility to reduce the number of development errors that are common in Javascript development, simply because the code is written in an highly typed language like Java.

More information about GWT can be found in [11].

## Chapter III

### *Security*

When web applications become rich client applications, that provide advanced and even critical service, security becomes a key aspect. Active page content, the engine of new web applications, can be also a source of attack from malicious parties. Today there are few developers well trained on this aspect, because web applications never before used such mechanics to delivery their services.

This topic is really big, so, in this chapter, we will cover the key aspects linked to the Javascript security and so we will talk on the client-side security and on some client-server interaction issues.

#### **III.1 Attack Types**

We talked about the *Same Origin Policy* in the first chapter, and we said that it was introduced for security reason. It is not sufficient to guarantee the security of a rich client web application. A lot of techniques were developed to break the web applications security. Next we will describe the most popular, giving some example.

The next paragraphs are quite technical, so a small knowledge in Javascript is needed to fully understood the text, moreover, we suggest to fully understood the *Same Origin Policy*, for example reading [8].

##### **III.1.1 Cross Site Scripting (XSS)**

In this attack an attacker injects malicious piece of code into an otherwise benign site. There are two types of XSS attack:

- ◆ Reflected XSS
- ◆ Stored XSS

The first is used when the attacker exploits a vulnerable Web Application that display back to the browser input parameters. Usually this happens publishing URLs that contains the attack string. Next is shown an example[8]:

```
http://www.trusted.com/search?keyword=<script>document.images[0].src="http://evil.com/steal?cookie=" + document.cookie; </script>
```

When someone clicks on this URL, he is forwarded to the trusted.com page, but, because of the vulnerability of this web application, the code contained in the URL will be executed. Let's read this code:

```
<script>
document.images[0].src="http://evil.com/steal?cookie=" + document.cookie;
</script>
```

This simple script makes a request to the attacker's web application, sending the user cookie as information. This way, the attacker collects personal cookies that could contain reserved informations.

Stored XSS uses the same strategy, but, instead of URLs, it uses the possibility to save content that will be shown in future by the web application, example of these applications are a wiki or a forum.

**SOLUTION:** XSS is an attack that can be easy avoided if the user input is sanitized. However this is not so simple to take care of all different sources of threat, and so, this is not so simple to know how to sanitize input. An interesting XSS cheat list is in [14].

### III.1.2 Cross Site Request Forgeries (CSRF)

This attack exploits server's session management system vulnerabilities. It is based on the assumption that the mainly used mechanism to trace sessions is the use of cookies.

When you log in to a web application, this web application sets a cookie on your browser and knows that no one can access that cookie (via browser) due to the *Same Origin Policy*. Every time the browser make a request to that web site, it appends the cookie, so the user session is identified.

These session cookies are deleted after a period of time or when the browser is closed (or never if the web application is done very badly!). The time the cookies are not deleted is a vulnerability time if the server uses only the cookie attached to the request to identify the session.

An example: The site [www.good.com](http://www.good.com) provide a service to paying registered users. Roberto is a registered user and he is visiting [www.good.com](http://www.good.com) and in the same time he has a browser window on [www.dummy.com](http://www.dummy.com). The latter site is done by the attacker, Antonio, to use the service of [www.good.com](http://www.good.com) without paying. [www.dummy.com](http://www.dummy.com) contains a script like:

```
<script language="javascript" src="www.good.com">
</script>
```

This way, [www.dummy.com](http://www.dummy.com) makes a request to [www.good.com](http://www.good.com) and, because Roberto is already authenticated there, the [www.good.com](http://www.good.com)'s cookie is attached to the request that starts from [www.dummy.com](http://www.dummy.com). If Antonio knows well the service offered by [www.good.com](http://www.good.com) he can build up a request that modify Roberto's data or much more.

**SOLUTION:** the solution in this case is based on a modification of the server side application, in order to make the session authentication system based on added informations and not only on the cookie. Usually the same cookie is passed as an explicit argument to the web application. This is a good solution because the cookie value can be read only from the page that sets the cookie.

Moreover, a best practice is to accept request for web application only via HTTP POST. This because even if GET is faster, a POST request is not possible by a third party page, because violates the *Same Origin Policy*.

### III.1.3 JSON Hijacking

JSON hijacking (sometimes called Javascript hijacking[15]) is an attack based on CSRF techniques. This attack uses the fact that JSON is a subset of JavaScript, so, if a service returns JSON strings as response, it is possible to assign this string to an object and to use this object from a third party page (the source page for the CSRF) to stole informations.

If a service returns a JSON object, this object is described the same as a Javascript object, and so, it is included in the page environment. If the attacker override the javascript's object constructor function, he can stole the informations included into the object.

This attack was used in the past to show a Gmail vulnerability, more detailed informations are in [17].

**SOLUTION:** because the attack is based on CSRF, avoiding CSRF corresponds to avoid JSON hijacking.

Anyway, including the JSON response into javascript code that makes the returned string unreadable if not elaborated is another way. For example, including the JSON response into “/\* \*/”, that are comments symbols for JavaScript. The attacker's page, using CSRF, can't modify the strings, and so, because this string for javascript is comment, it is ignored and the data inside it can't be read by the attacker's code.

## Chapter IV

### *The need for RPC*

With the coming of new web development patterns there is the need for new development techniques. As we said in the previous chapters, future web applications are composed by complex clients and complex servers. This approach can't rely on simple HTTP request that are manually handled by programmers, due to the complexity of this process, for the client-server interaction.

The scenario depicted by new web applications is composed by servers that offer a service in the form of an API and clients that consume that service using that interface. This makes clear the need for a *Remote Procedure Call* framework that makes the use of a server's service easy and fast for clients programmers.

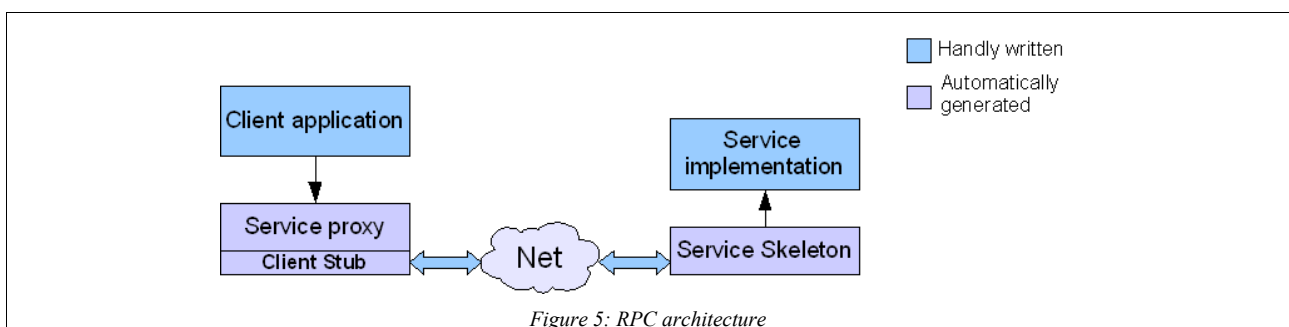
This way is well known, and, as we saw, tools like GWT already provide RPC mechanisms.

The problem is that GWT (that is our reference tool in this text), provide only a Java Servlet backhand. Our goal is to build a simple RPC framework for a more cheaply PHP backhand. This effort is done to provide a framework to build a simple application (In the next chapter!), to give a real sample of the possibilities of this new development approach.

### IV.1 RPC requirements

The RPC framework we want to build will provide a tool to automatically generate a set of Java and PHP5 classes to handle the service's calls between client and server. The approach is the classic with stub classes:

- ◆ **Client stub:** provide a service proxy, in order to give to the programmer the abstraction that he is making a local call. These classes are generated in Java and then translated in Javascript by the GWT compiler;
- ◆ **Server stub:** provide a service skeleton that the programmer completes giving the service implementation. These classes are generated in PHP5.



The framework uses a custom RPC protocol to makes calls. We will not give deep informations on this protocol because it is designed only for sample purpose and is not a really good protocol. Anyway, the main goals of an RPC protocol are the next:

- ◆ to provide a way to identify the service to call;
- ◆ to provide a way to make a service call;
- ◆ to provide a way to identify the request and so to associate the responses to their requests.

While the first two requirements are quite simple to satisfy and there is not much more to say than what was already said (for example in [25]), the third requirement is slightly more interesting.

An RPC protocol based on HTTP is basically asynchronous, so we can't use a simple request serialization to identify requests and their responses. The approach we used for our sample protocol is to provide a separate request channels abstraction. In other words, every request is done on a dedicated channel, where only the request and its response are exchanged. This way the identification is simplified also for the programmer, that will use a simple callback pattern to handle the response to a request.

## ***IV.2 GJPR***

The framework that we developed can be found in [24], it is made by a really simple PHP tool that parses a service interface written in a simple custom language and generates PHP and Java classes.

Actually this tool supports only primitive Java types and their mono-dimensional arrays as functions arguments and return values.

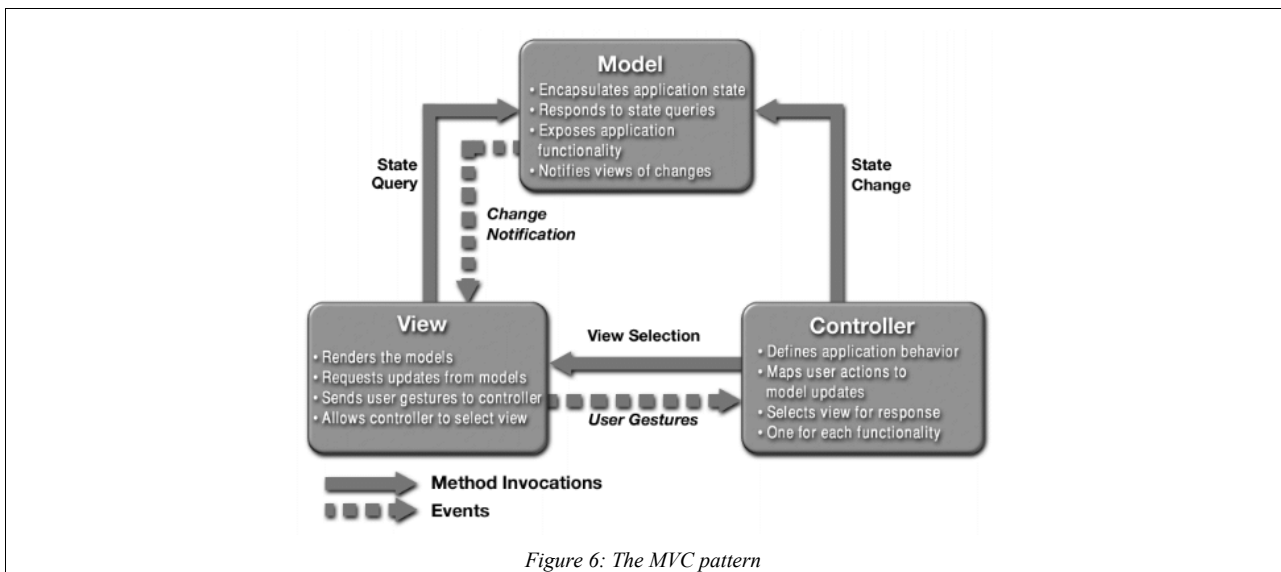
# Chapter V

## *A simple chat application*

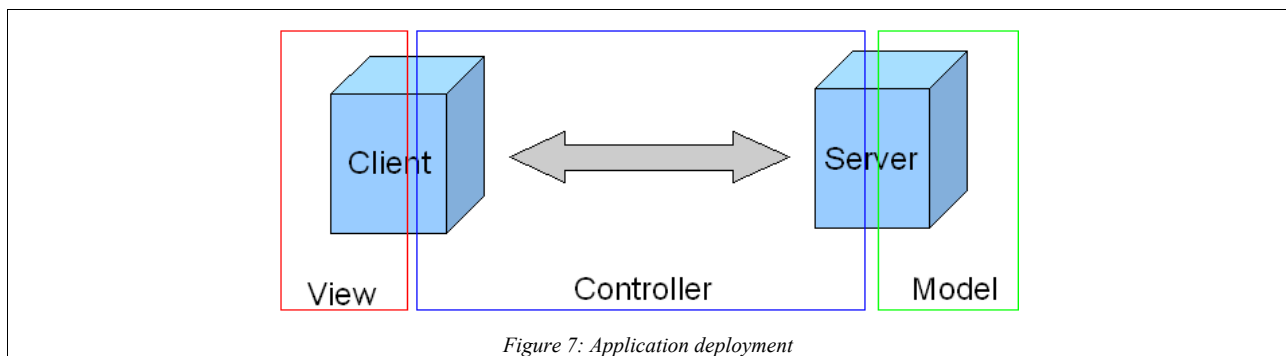
We used technologies shown in previous chapters to develop a sample application. The application is a simple chat, that uses GWT for the client development, GJPR for the RPC and PHP for the server development.

### V.1 Architecture

The application architecture follows the classic *Model View Controller* pattern.



The three model parts are deployed on the client and on the server, as shown in *Figure 7*.



The client is entirely responsible for the view, the same for the server as regard as the model. The controller is shared between the two entities because elaborations and validations are needed on both sides.

The server, using RPC, offers a service interface, defined through GJPR. The resulting Java Interface is the next:

```
package it.robortobifulco.web.simplechat.client.service;

public interface ChatRoom{

    public boolean addMessage(String from, String date, String message);

    public String[] getMessages(int since);

    public boolean addMember(String username);

    public boolean removeMember(String username);

    public String[] getMessageFrom(String username, String from, int since);

    public boolean sendMessageTo(String from, String to, String date, String message);

    public String[] getRoomMembers();

    public int[] getStatus(String username);

    public String[] getPrivateCommunications(String username);

}
```

Calling these methods it is possible to add an user to the chat or to remove it, add a message and manage private messages between users.

## V.2 The polling problem

HTTP uses the request-response model, where only the client can start a communication. This is a problem for a chat service that needs to fire messages to the chat user every time a message is added, which is an asynchronous event. So, to get an update a client must request it.

This is a problem for two reasons:

1. The client doesn't know when a new message is on chat;
2. Requests for update can be not useful because there is no new messages on chat.

The solution in these case is the using of a polling algorithm, that must resolve a trade off between the time to update the user's chat, in other words the waiting time perceived by the user, and the net and server load, to resolve update requests. If the polling algorithm is not well setted, there is the possibility that a user receives updates late, having the feel that the chat is “slow”, on the other side, there is the possibility that to much not useful update requests overload the server and the net.

The decision about the polling algorithm is made taking in consideration this:

*“If there isn't new messages, it's probable that no new messages will come and conversely”*

Using this consideration, the time interval between two update requests is adapted at every new request response, increasing if the request response is a “miss”, that means that no new messages are on chat, decreasing if new messages are on chat (“hit”).

There are five parameters to set the algorithm, we will call simply interval the time interval between two subsequent update requests:

1. *Starting interval dimension*: this is not an important parameter, it has an effect only in the starting phase of the algorithm;
2. *Interval minimum dimension*: the minimum time interval between two subsequent requests;
3. *Interval maximum dimension*: the maximum time interval between two subsequent requests;
4. *Interval increasing factor*: the time increase that the interval obtain after a *miss*;
5. *Interval decreasing factor*: the time decrease that the interval obtain after an *hit*.

These parameters influence the algorithm in various modes. For example, setting a very low interval minimum dimension will make really fast the chat from the user perspective. On the other side this will overload the server.

The increasing and decreasing factors are really important parameters, they represent the capacity of the algorithm to follow the changing status of the chat.

After these considerations, we set these parameters as follow (values expressed in seconds):

1. *Starting interval dimension*: 3;
2. *Interval minimum dimension*: 0,5;
3. *Interval maximum dimension*: 5;
4. *Interval increasing factor*: 0,5;
5. *Interval decreasing factor*: 2.

These settings gives a slow increase to the interval if no messages are on chat, and a fast decreasing if new messages are found. This because if a new message is on chat it is probable that someone will response to that message. The curve that these settings paints is the next (based on a given *hit – miss* sequence):

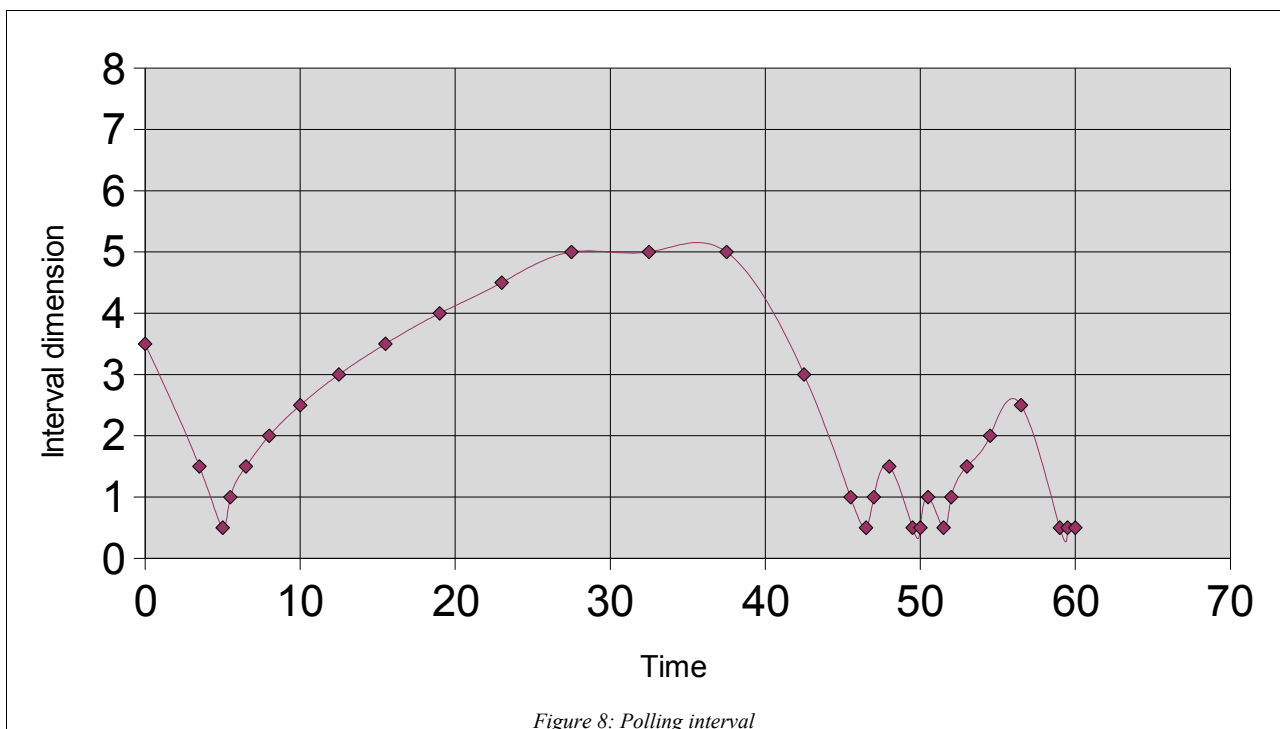


Figure 8: Polling interval

## Bibliography

- [1] Programmazione Web Lato Server – Della Mea, Di Gaspero, Scagnetto, 2007
- [2] AJAX per applicazioni web – Romagnoli, Salerno, Guidi, 2007
- [3] Guida PHP di base – Gianluca Gillini, HTML.it
- [4] Guida PHP pratica – Gabriele Farina, HTML.it
- [5] Open Web Application Security Project – <http://www.owasp.org>
- [6] The Same Origin Policy – <http://www.mozilla.org/projects/security/components/same-origin.html>
- [7] Ajax and Mash-up Security – <http://www.openajax.org/whitepapers/Ajax%20and%20Mashup%20Security.html>
- [8] Overcome security threats for Ajax applications – <http://www.ibm.com/developerworks/library/x-ajaxsecurity.html>
- [9] Javascript description – <http://www.w3schools.com/js/default.asp>
- [10]JSON – <http://www.json.org/>
- [11]GWT – <http://code.google.com/webtoolkit/>
- [12]PHP Manual – <http://www.php.net>
- [13]OpenAjax whitepapers – <http://www.openajax.org/>
- [14]XSS cheat sheet – <http://hackers.org/xss.html>
- [15]JavaScript Hijacking – Chess, O'Neil, West, Fortify Software, 2007
- [16]Shaping the future of secure AJAX mashups – <http://www-128.ibm.com/developerworks/library/x-securemashups/>
- [17]Safe JSON – Yates, 2007
- [18]Remote JSON - JSONP – <http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/>
- [19]Security for GWT Applications – <http://groups.google.com/group/Google-Web-Toolkit/web/security-for-gwt-applications>
- [20]JSON is not as safe as people think it is – [http://getahead.org/blog/joe/2007/03/05/json\\_is\\_not\\_as\\_safe\\_as\\_people\\_think\\_it\\_is.html](http://getahead.org/blog/joe/2007/03/05/json_is_not_as_safe_as_people_think_it_is.html)
- [21]Using GWT for JSON Mashups – Morril, 2007
- [22]AJAX and Mashup security – <http://ajax.sys-con.com/read/436300.htm>
- [23]Web Security – Roberto Bifulco, <http://www.robertobifulco.it/PV/projects/Web%20Security/articles/Web%20Security.html>
- [24]GJPR – Roberto Bifulco, <http://www.robertobifulco.it/PV/projects/GJPR/articles/GJPR.html>
- [25]Computer Networks: a system approach – Peterson, Davie, 2007
- [26]GWT-Ext Widget Library – <http://code.google.com/p/gwt-ext/>
- [27]Gwt Window Manager – <http://www.gwtwindowmanager.org/>
- [28]Software + Services – The Architecture Journal 13, Microsoft, 2007