WIRTSCHAFTSUNIVERSITÄT WIEN SEMINARARBEIT

Titel der Seminararbeit:

Vergleich der Einbindungsmöglichkeiten von AJAX-Komponenten mittels Apache MyFaces und Ruby on Rails

Englischer Titel der Seminararbeit:

Comparison of the possibilities to implement AJAX-Components with Apache MyFaces and Ruby on Rails

Verfasser:	Clemens Friedrich
Matrikel-Nr.:	0450510
Studienrichtung:	J033 526 Bakkalaureat Wirtschaftsinformatik
Kurs:	1224 IT-Spez: Vertiefungskurs VI - Electronic Commerce
Textsprache:	Englisch
Betreuerin/Betreuer:	ao. Univ. Prof. Dr. Rony G. Flatscher

Table of Contents

Table of Figures

Index of Tables

1 Abstract

This term paper deals with the question which possibilities the two web-application frameworks Apache MyFaces and Ruby on Rails provide to implement AJAX components.

The first part of the paper gives an introduction to the term AJAX and explains what is defined by it. As AJAX offers a lot of advantages for interactive web-applications there are also a lot of drawbacks that have to be considered which are also covered in this chapter.

The next two chapters discuss the two mentioned web-application frameworks. Both are introduced and the basics are explained. Later on the focus is set on the possibilities of implementing AJAX components with the help of these frameworks. In both frameworks the JavaScript Libraries that stand behind the frameworks are explained and code examples will be given.

After analyzing the frameworks a comparison identifies both the advantages and disadvantages of the two frameworks.

Keywords:

AJAX, Ruby on Rails; Apache MyFaces; JavaServer Faces; XMLHttpRequest; Prototype; script.aculo.us; Dojo Toolkit

2 Introduction to AJAX

"Ajax is an acronym for Asynchronous Javascript And XML. Ajax is not a new programming language, it's an umbrella term which describes a group of features and enhancements to improve appearance and functionality of traditional web sites." [PaFe06]

As mentioned above Ajax is not a new programming language, but a so-called "umbrella term", it is necessary to take a closer look what is under the umbrella. The core of Ajax embraces the following techniques [Garr05]:

- **XHTML** and **CSS** for presentation
- **Document Object Model** for dynamic presentation
- **XML** and **XSLT** for data interchange and manipulation
- **XMLHttpRequest** for asynchronous data retrieval
- **JavaScript** that binds together the components

As it is getting clearer, Ajax has something to do with asynchronous transfer of data. Actually it is the transfer of data between a client which is a web-browser and a server. To understand this concept it is essential to have a look on both the classic and the Ajax web model.

Foremost it is crucial to mention that the techniques behind AJAX are not really new. That means that the idea of updating web-pages incrementally using the eXtensible Markup Language and the HyperText Transmission Protocol rose several years ago but gained its popularity when the popular name AJAX was mentioned first by Jesse James Garrett in February 2005 and used by some important internet companies like Amazon or Google [Garr05] [Gibs06].

2.1 The classic web model

In the classic web model, a user-action, like clicking on a hyperlink on a web-page, results in a request to the server. This server receives the request and performs actions to fulfill the request of the user. These actions could be for example a database query accessing other systems or simply calculating something. After that the server is able to deliver the requested file to the user which is mostly a HTML file. Figure 1 demonstrates this model [Garr05].

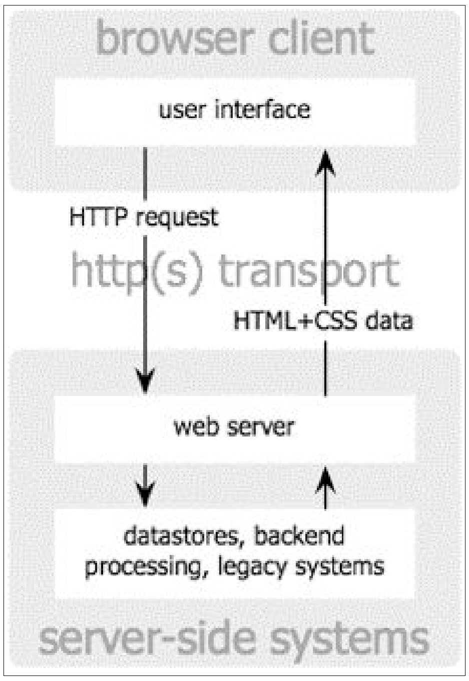


Figure 1: The classic web application model [Garr05]

The main problem of this this model is, that the user has to wait while the server is processing the request. So the user has to consider a break in his workflow every time the application needs information from the server. Figure 2 shows this problem in the context of the time.

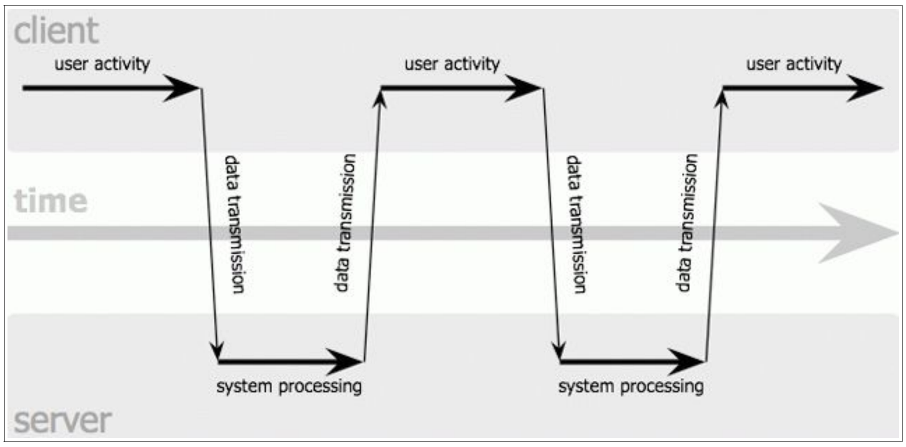


Figure 2: The classic web application model including the time axis [Garr05]

To bypass this problem the Ajax technique comes into play.

2.2 The Ajax web model

The Ajax model gives an application the possibility to send HTTP-requests to a server without refreshing a whole page, not even giving the user any notice of the request. Ajax is then able to present changes to the user without reloading the whole page by using the Document Object Model, DOM [PaFe06].

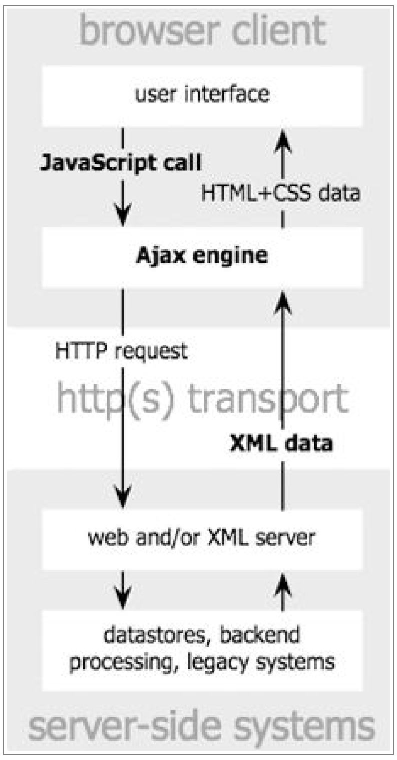


Figure 3: The AJAX web application Model [Garr05]

In the basic web model, each user interaction would lead to a HTTP-request. Within the

Ajax web model this user interaction leads to a JavaScript call to the Ajax engine. Simple actions, like validating a form, can be done by the Ajax engine itself without requesting data from the server. But if the engine needs data that are not loaded yet it requests the data asynchronously using XML without interrupting the users work [Garr05]. Figure 3 shows this model in the technical way while Figure 4 shows the model in the context of the time.

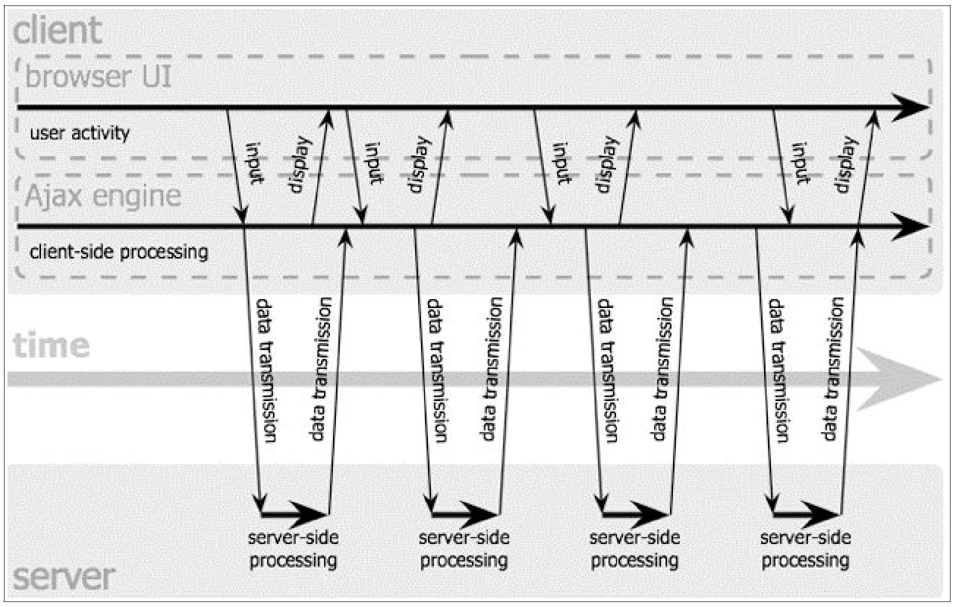


Figure 4: The AJAX web application model including the time axis [Garr05]

2.3 AJAX analyzed

As the previous section showed, AJAX stands for a simple communication between server and client at a sub-page level. Some time ago there have been several approaches to accomplish this goal. One of those were Java Applets, which did not really succeed because of slow Java implementations and suffering cross-browser compatibility [Raym07].

Nowadays the XMLHttpRequest Object is the widely used solution for building AJAX applications. Its original implementation was by Microsoft in the Internet Explorer 5 called *XMLHTTP* using ActiveX and was cloned by other browsers and called *XMLHttpRequest* [Raym07].

As AJAX stands for Asynchronous JavaScript and XML these terms are now analyzed and reasons why AJAX is more a term than a definition of the used technologies are shown.

Asynchronous means that all the calls back to the server, the XMLHttpRequest, are nonblocking which implies that the client is able to execute further code while waiting for the response from the server. If there were no nonblocking characteristic, users would have to wait for the response without doing anything, because the browser is not able to work. So the browser would seem to freeze while the server responds to the request [Raym07].

JavaScript, which was originally called LiveScript, is a powerful scripting language that is supported more-or-less in every modern web-browser. Especially with the use of the numerous libraries that support JavaScript and development support tools, JavaScript is a very agreable platform. But as Internet Explorer supports Visual Basic scripting as well and *Adobe Flash* is widely deployed and both technologies are capable of calls to the server, it is not a necessity to use JavaScript [Raym07].

The *Extensible Markup Language (XML),* is the last term defined under the umbrella term AJAX and is the easiest technique to substitute. Actually the XMLHttpRequest Object is able to transfer every type of content, HTML code as well as for example images. Ruby on Rails for example transfers HTML and JavaScript rather than XML data [Raym07].

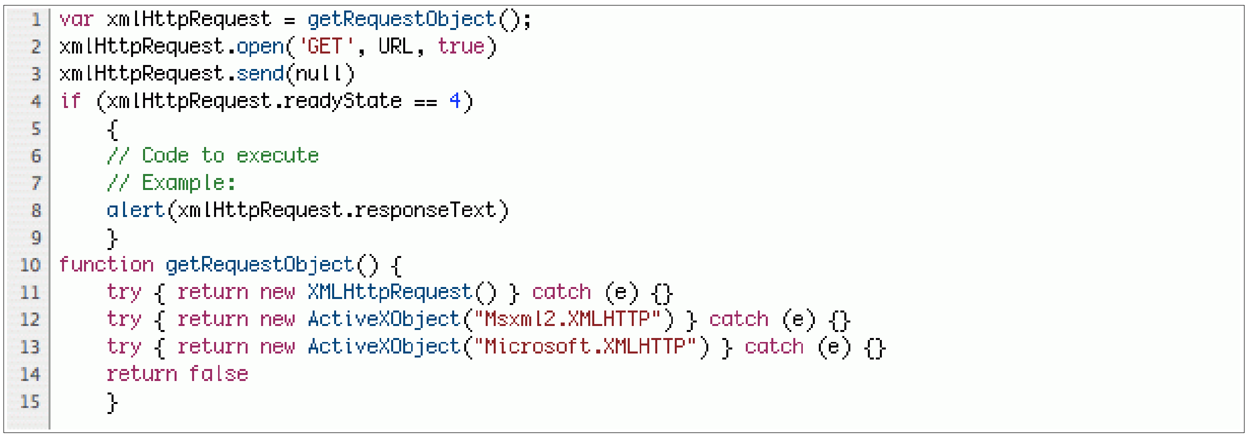
In the personal opinion of the author the most important piece of AJAX is the XMLHttpRequest Object, which is the heart of every AJAX application. As for frameworks like Ruby on Rails it is not necessary to code JavaScript code or understand XML to create a simple applications, the following section will cover just the XMLHttpRequest Object because the author is convinced to understand the basics of the object is crucial to understand the examples that are presented later.

2.4 The XMLHttpRequest Object

The Object XMLHttpRequest is part of most web-browsers and responsible for sending requests to the server and receiving the responses. In most open-source web-browsers like Mozilla Firefox the regular XMLHttpRequest Object is implemented. Microsofts

Internet Explorer 5 and 6 contain a XMLHTTP Object which is part of the ActiveX component while Internet Explorer 7 implements the regular XMLHttpRequest Object [Lubk07].

The actual cycle of using the object is to use the method `open` to create a new object, `send` to transmit the request. For defining what is going to happen after the response was received the `onreadystatechange` is used [Lubk07]. Figure 5 shows this cycle in a simplified way.



```
 1 var xmlHttpRequest = getRequestObject();
 2 xmlHttpRequest.open('GET', URL, true)
 3 xmlHttpRequest.send(null)
 4 if (xmlHttpRequest.readyState == 4)
 5     {
 6     // Code to execute
 7     // Example:
 8     alert(xmlHttpRequest.responseText)
 9     }
10 function getRequestObject() {
11     try { return new XMLHttpRequest() } catch (e) {}
12     try { return new ActiveXObject("Msxml2.XMLHTTP") } catch (e) {}
13     try { return new ActiveXObject("Microsoft.XMLHTTP") } catch (e) {}
14     return false
15     }
```

Figure 5: Code of a XMLHttpRequest Object cycle

The first line creates a new `xmlHttpRequest` object by calling the function `getRequestObject()` which checks the type of the browser. The second line opens a new request with some parameters. The first one is `'GET'` and defines that information is submitted in the link that is called. It would also be able to use `'POST'` which sends information in the data part or `'HEAD'` which does not send the content of the document with it. The second parameter is the `URL` that will be called and the third one is a boolean parameter that defines whether the request is asynchronous or not. Line 3 finally sends the request to the server. Line 4 checks the value of `xmlHttpRequest.readyState` which defines the state of the query and is changed several times during the lifetime of a request. There are five different states [Lubk07]:

Value	**Meaning**
0	`open` was not called yet
1	`open` was called but `send` was not
2	request was sent
3	data transfer from the server is in progress
4	data transfer is completed

Table 1: States of a XMLHttpRequest

As Table 1 shows the last status is 4, so if the value of `xmlHttpRequest.readyState` is 4 the transfer is completed and the received data is ready to use as it is done in line 8.

2.5 AJAX on the Network Layer

To provide a closer look at AJAX and what it actually does it makes sense to have a look what happens on the network while AJAX is working. Therefore a network protocol analyzer was used combined with a web-browser that executed AJAX commands.

The author of this paper decided to choose the web-browser *Mozilla Firefox*. Firefox is an Open-Source product of the Mozilla Foundation which provides all its products on Microsoft Windows, Mac OS X and Linux [Mozi07]. Furthermore Firefox is the worlds second most widely used web-browser with a market share of 15.6% measured in November 2007 [Mark07]. The data that is going to be analyzed derive from the Google Project called *Google Suggest* which is a typical Google search page that provides suggestions in real time while typing the search term [Goog07]. Figure 6 shows the start-page of Google Suggest.

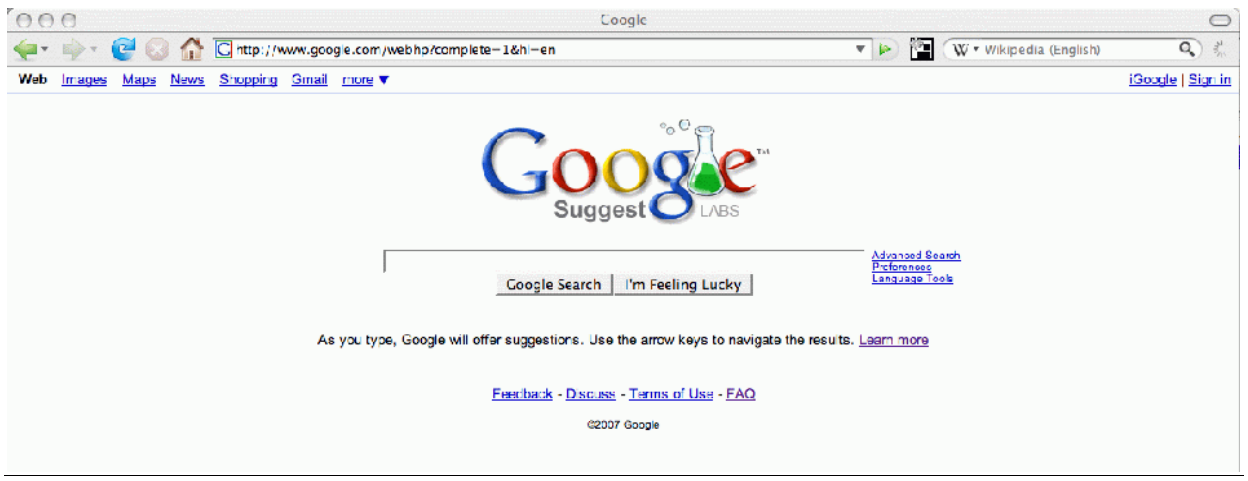
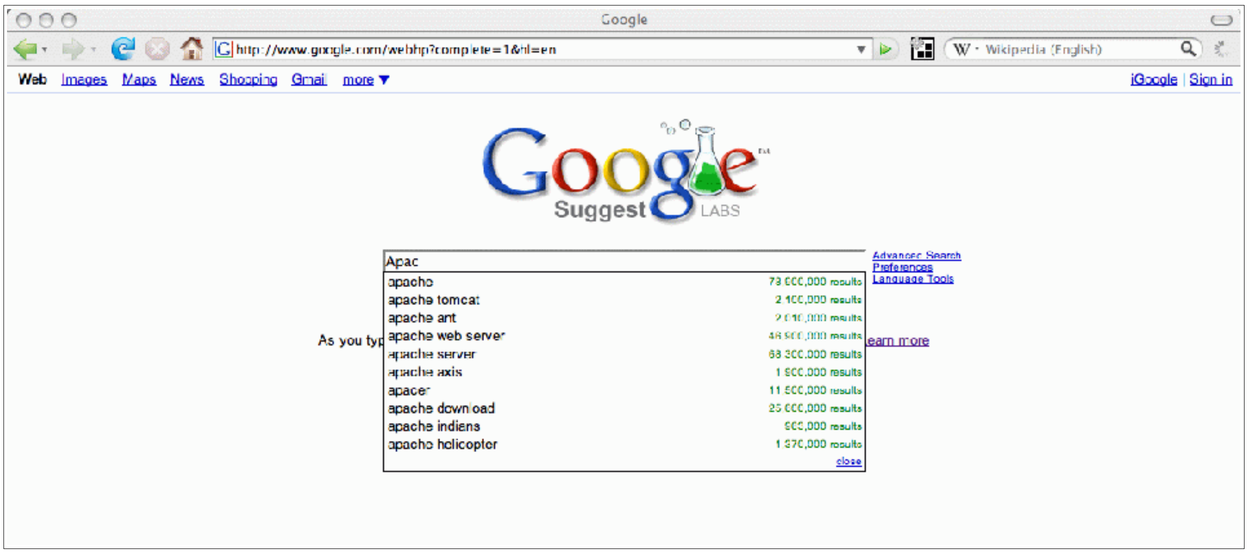

Google
http://www.google.com/webhp?complete=1&hl=en
Wikipedia (English)
Web Images Maps News Shopping Gmail more ▼
iGoogle | Sign in
Google Suggest LABS
Google Search
I'm Feeling Lucky
Advanced Search
Preferences
Language Tools
As you type, Google will offer suggestions. Use the arrow keys to navigate the results. Learn more
Feedback - Discuss - Terms of Use - FAQ
©2007 Google

Figure 6: Start Page of Google Suggest

To analyze the activities on the network while performing Google Suggest the network analyzer *Wireshark* which was formerly known as *Ethereal was* used. It supports the inspection of numerous protocols, such as HTTP, which is very important for this purpose [Wire07]. To demonstrate how Google Suggest works Figure 7 and Figure 8 show what happens when you start typing and searching for the term *“Apache MyFaces”.*

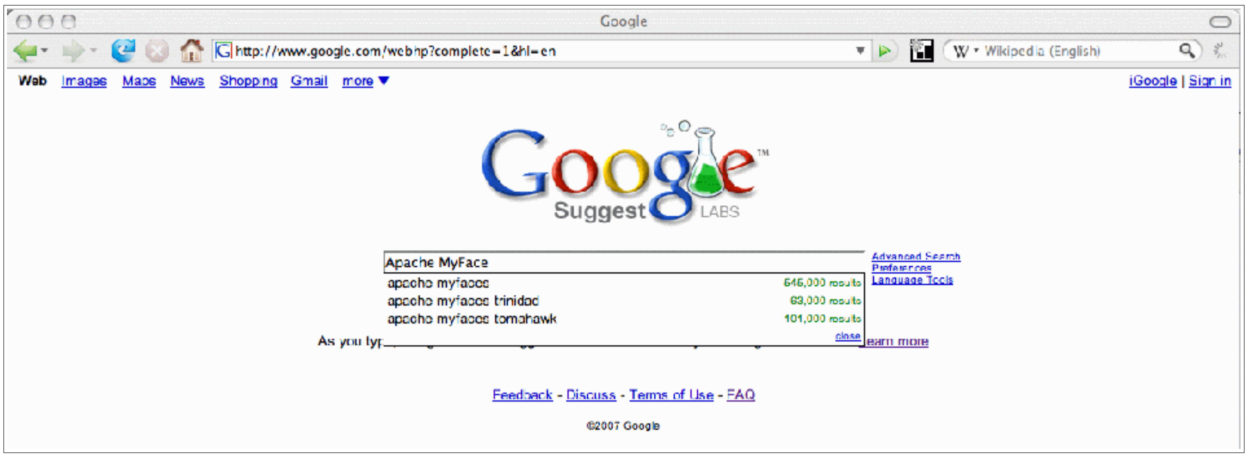


Apac

apache	78,000,000 results
apache tomcat	2,100,000 results
apache ant	2,010,000 results
apache web server	46,900,000 results
apache server	68,300,000 results
apache axis	1,900,000 results
apacer	11,500,000 results
apache download	25,000,000 results
apache indians	900,000 results
apache helicopter	1,370,000 results

Figure 7: Google Suggest: Searching Status 1

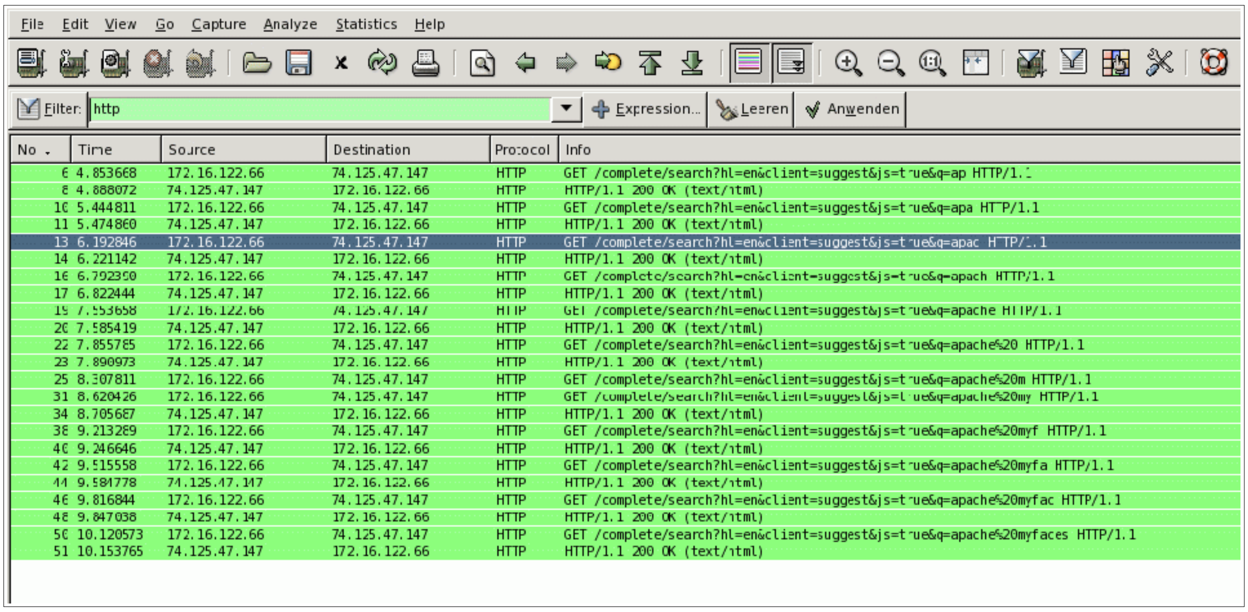
While typing Google Suggest is trying to find matching suggestions and presents them in a list below the textbox. The more information the user types the narrower the suggestions are.


Google
http://www.google.com/webhp?complete=1&hl=en
Wikipedia (English)
Web Images Maps News Shopping Gmail more
iGoogle | Sign in
Google Suggest LABS
Advanced Search
Preferences
Language Tools
learn more
Feedback - Discuss - Terms of Use - FAQ
©2007 Google

```
Apache MyFace
apache myfaces                    646,000 results
apache myfaces trinidad            63,000 results
apache myfaces tomahawk           101,000 results
                                            close
As you type
```

Figure 8: Google Suggest: Searching Status 2

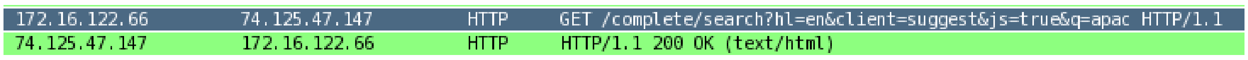
As the search term *"Apache MyFaces"* is typed, the network analyzer Wireshark monitors the network traffic all the time. After recording the result is filtered to provide just HTTP traffic. Figure 9 shows the result of the monitoring.



```
No .  Time       Source          Destination     Protocol  Info
   6  4.853668   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=ap HTTP/1.1
   8  4.888072   74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
  10  5.444811   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apa HTTP/1.1
  11  5.474860   74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
  13  6.192846   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apac HTTP/1.1
  14  6.221142   74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
  16  6.792350   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apach HTTP/1.1
  17  6.822444   74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
  19  7.553658   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apache HTTP/1.1
  20  7.585419   74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
  22  7.855785   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apache%20 HTTP/1.1
  23  7.890973   74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
  25  8.307811   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apache%20m HTTP/1.1
  31  8.620426   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apache%20my HTTP/1.1
  34  8.705687   74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
  38  9.213289   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apache%20myf HTTP/1.1
  40  9.246646   74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
  42  9.515558   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apache%20myfa HTTP/1.1
  44  9.584778   74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
  46  9.816844   172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apache%20myfac HTTP/1.1
  48  9.847038   74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
  50  10.120573  172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apache%20myfaces HTTP/1.1
  51  10.153765  74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
```

Figure 9: Wireshark: Result of the network analysis

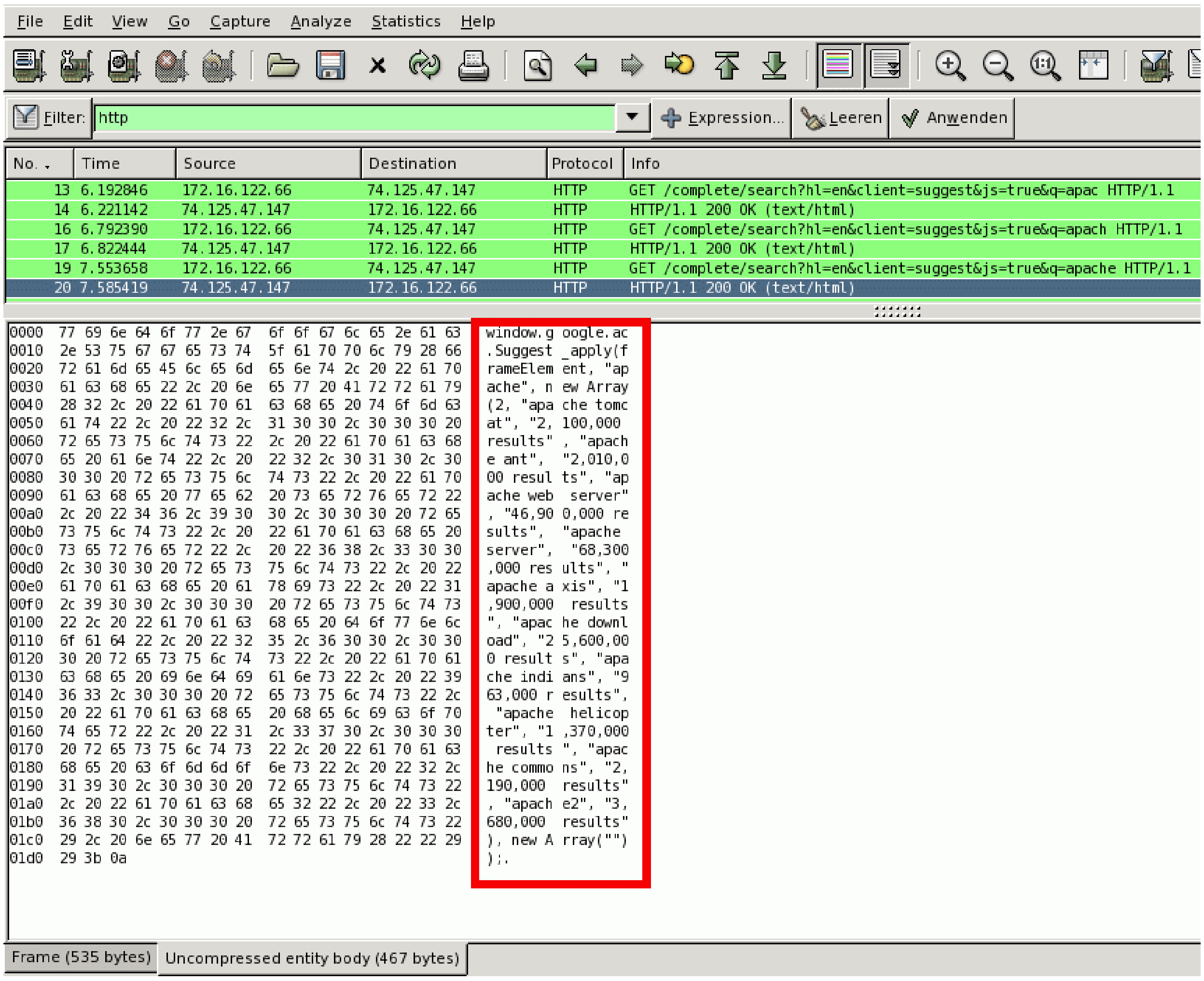
A closer look on the "Info" column shows that every keystroke leads to a HTTP request to Google that transfers the current input of the text-box, where the search terms can be entered. The specific command that is executed for example after typing "Apac" is "*GET /complete/search?hl=en&client=suggest&js=true&q=apac HTTP/1.1".* Obviously a search function of Google is called and the *"q"* attribute defines the term that should be looked up, which in this case is *"Apac". Figure 10* shows this specific request.



```
172.16.122.66    74.125.47.147    HTTP   GET /complete/search?hl=en&client=suggest&js=true&q=apac HTTP/1.1
74.125.47.147    172.16.122.66    HTTP   HTTP/1.1 200 OK (text/html)
```

Figure 10: Wireshark: HTTP GET request to Google

The response of the server is right below the GET request in Figure 10. A closer look into the body of this package shows the results that are provided in the suggestion list. Figure 12 shows the results in Wireshark which are, as expected, the same as in Figure 7. Examples are “Apache Tomcat” with 2,100.000 results or the “Apache Helicopter” with 1,370.000 results.


File Edit View Go Capture Analyze Statistics Help
Filter: http
Expression... Leeren Anwenden

```
No. .  Time      Source          Destination     Protocol  Info
   13 6.192846  172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apac HTTP/1.1
   14 6.221142  74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
   16 6.792390  172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apach HTTP/1.1
   17 6.822444  74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
   19 7.553658  172.16.122.66   74.125.47.147   HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=apache HTTP/1.1
   20 7.585419  74.125.47.147   172.16.122.66   HTTP      HTTP/1.1 200 OK (text/html)
```

```
0000  77 69 6e 64 6f 77 2e 67  6f 6f 67 6c 65 2e 61 63   window.g oogle.ac
0010  2e 53 75 67 67 65 73 74  5f 61 70 70 6c 79 28 66   .Suggest _apply(f
0020  72 61 6d 65 45 6c 65 6d  65 6e 74 2c 20 22 61 70   rameElem ent, "ap
0030  61 63 68 65 22 2c 20 6e  65 77 20 41 72 72 61 79   ache", n ew Array
0040  28 32 2c 20 22 61 70 61  63 68 65 20 74 6f 6d 63   (2, "apa che tomc
0050  61 74 22 2c 20 22 32 2c  31 30 30 2c 30 30 30 20   at", "2, 100,000 
0060  72 65 73 75 6c 74 73 22  2c 20 22 61 70 61 63 68   results" , "apach
0070  65 20 61 6e 74 22 2c 20  22 32 2c 30 31 30 2c 30   e ant",  "2,010,0
0080  30 30 20 72 65 73 75 6c  74 73 22 2c 20 22 61 70   00 resul ts", "ap
0090  61 63 68 65 20 77 65 62  20 73 65 72 76 65 72 22   ache web  server"
00a0  2c 20 22 34 36 2c 39 30  30 2c 30 30 30 20 72 65   , "46,90 0,000 re
00b0  73 75 6c 74 73 22 2c 20  22 61 70 61 63 68 65 20   sults",  "apache 
00c0  73 65 72 76 65 72 22 2c  20 22 36 38 2c 33 30 30   server",  "68,300
00d0  2c 30 30 30 20 72 65 73  75 6c 74 73 22 2c 20 22   ,000 res ults", "
00e0  61 70 61 63 68 65 20 61  78 69 73 22 2c 20 22 31   apache a xis", "1
00f0  2c 39 30 30 2c 30 30 30  20 72 65 73 75 6c 74 73   ,900,000  results
0100  22 2c 20 22 61 70 61 63  68 65 20 64 6f 77 6e 6c   ", "apac he downl
0110  6f 61 64 22 2c 20 22 32  35 2c 36 30 30 2c 30 30   oad", "2 5,600,00
0120  30 20 72 65 73 75 6c 74  73 22 2c 20 22 61 70 61   0 result s", "apa
0130  63 68 65 20 69 6e 64 69  61 6e 73 22 2c 20 22 39   che indi ans", "9
0140  36 33 2c 30 30 30 20 72  65 73 75 6c 74 73 22 2c   63,000 r esults",
0150  20 22 61 70 61 63 68 65  20 68 65 6c 69 63 6f 70    "apache  helicop
0160  74 65 72 22 2c 20 22 31  2c 33 37 30 2c 30 30 30   ter", "1 ,370,000
0170  20 72 65 73 75 6c 74 73  22 2c 20 22 61 70 61 63    results ", "apac
0180  68 65 20 63 6f 6d 6d 6f  6e 73 22 2c 20 22 32 2c   he commo ns", "2,
0190  31 39 30 2c 30 30 30 20  72 65 73 75 6c 74 73 22   190,000  results"
01a0  2c 20 22 61 70 61 63 68  65 32 22 2c 20 22 33 2c   , "apach e2", "3,
01b0  36 38 30 2c 30 30 30 20  72 65 73 75 6c 74 73 22   680,000  results"
01c0  29 2c 20 6e 65 77 20 41  72 72 61 79 28 22 22 29   ), new A rray("")
01d0  29 3b 0a                                           );.
```

Figure 11: Wireshark: Suggestions received from the Google Server

2.6 Advantages of AJAX-Applications

With AJAX developers are able to avoid some typical problems of web-applications. One of these is that users have to wait until a new page has loaded. AJAX makes it possible to load further data in the background so that the user does not recognize. A very famous example is the application Google Maps. It pre-fetches the map information that lies right next to the part of the map viewed by the user. So there is no time of waiting until the new part of the map is loaded if the user drags the map. This advantage relates to the A in AJAX, which stands for Asynchronous and implies that data can be exchanged between browser and server without an interaction of the user [Zuck07].

Another advantage of AJAX is the improved user interactivity with the application. AJAX applications try to act more like desktop applications. This means no refreshing of a whole site or no long waiting times while the application gets the necessary information from the server, but still retaining its benefits like no need of downloading an application, no updates that have to be installed and a wide compatibility over various Operating Systems an web-browsers. A good example for the improved user interactivity is GMail, an email service provided by Google. In its web-interface it is not necessary to press the refresh button of the web-browser to reload the whole page to see if there are new mails. Instead, only the new mail is added to the inbox without reloading the whole page [Zuck07].

Those are the two major benefits that developers and users can earn when using AJAX for dynamic web-pages. But as AJAX is more a bunch of technologies than a standalone technique, it has to face a lot of problems and developers have to consider a lot of wattles to use AJAX appropriate and not to penalize the user by providing low usability applications.

Therefore the next part of this paper covers the problems and risks of AJAX applications.

2.7 Disadvantages of AJAX-Applications

So far Ajax sounds to be a good way to bring more dynamic life into the static web model. But there are several drawbacks and issues that have to be considered when talking about Ajax. This part is subdivided into an usability and a security part, to get a perspective of both issues.

First there is a general drawback: AJAX is widely based on JavaScript, which implies that Web-Browsers have to support JavaScript to show the contents of the page. This might be a problem for mobile and text-only browsers. Furthermore the World Wide Web Consortium Web Content Accessibility Guidelines (WCAG) 1.0 requires to run web-sites with turned off JavaScript support. Even if version 2.0 of the guidelines, which is not recommended yet, removes this requirement, this issue has to be considered at the moment [Gibs06].

2.7.1 Usability problems

The experience of looking at AJAX-driven web-pages is new for many internet users. Of course the new possibilities, provided by AJAX, are a good and easy way to implement more dynamically acting web-pages, but some users will face several problems or worries.

User related problem

Web-pages that contain AJAX components are basically designed to update a web-page dynamically. Users may not expect changes to the current page, because the classic way of surfing through the internet is to click on a hyperlink and waiting for a new page to load. To give an example: Just imagine a shopping cart of an online-store that updates the selected goods, prices, shipping costs and taxes automatically without asking for it. Some users may be unaware of these changes which could lead to confusion of the customers [Gibs06].

Another problem of some AJAX applications is that they do not provide a distinct Unified Resource Identifier which the user could simply add to the bookmarks [Gibs06].

Search engine problem

Search engines and their crawlers comb through web-pages looking for link tags like *HREF* or *SRC*. So these crawlers are not looking for any AJAX or JavaScript parts. Furthermore, the search engines do not know which values they have to pass to the server to call the right state of the application. To make AJAX web-pages reachable from search engines developers have to write two versions of their web-pages: One with the actual AJAX code and one with the HTML code for the web-crawlers [Schw06].

Back button problem

As the XMLHttpRequest circumvents the web-browsers history function, pushing the “back” button mostly does not lead to the desired result [Inma06]. Users normally want to go back one step when pushing the “back” button. In an AJAX application this could be revoking the last action as like deleting an item that was put in the shopping-cart accidently. The web-browser would instead call the last-visited web-page. Despite several workarounds of this problem like working with IFrames, this issue is very important regarding the usability of web-pages.

URL problem

As the communication of AJAX applications happens in the background and is working asynchronously, the user requests the URL only once when entering the web-page and might change the state of the web-page by interacting with the AJAX application. If the user wants somebody else to see the same thing, usually he would copy the URL of the web-page and send it to the person he wants to share the page with. Since the communication happens in the background, the URL does not change while navigating through the AJAX application. Thus the URL does not contain the information of the users actual status in the application. Figure 12 shows an example of this problem. The web-site http://www.us.map24.com is loaded and a route between two US-Cities has been searched. As the result appears on the screen the URL stays the same, so it is not possible to send this link and open it in another web-browser.

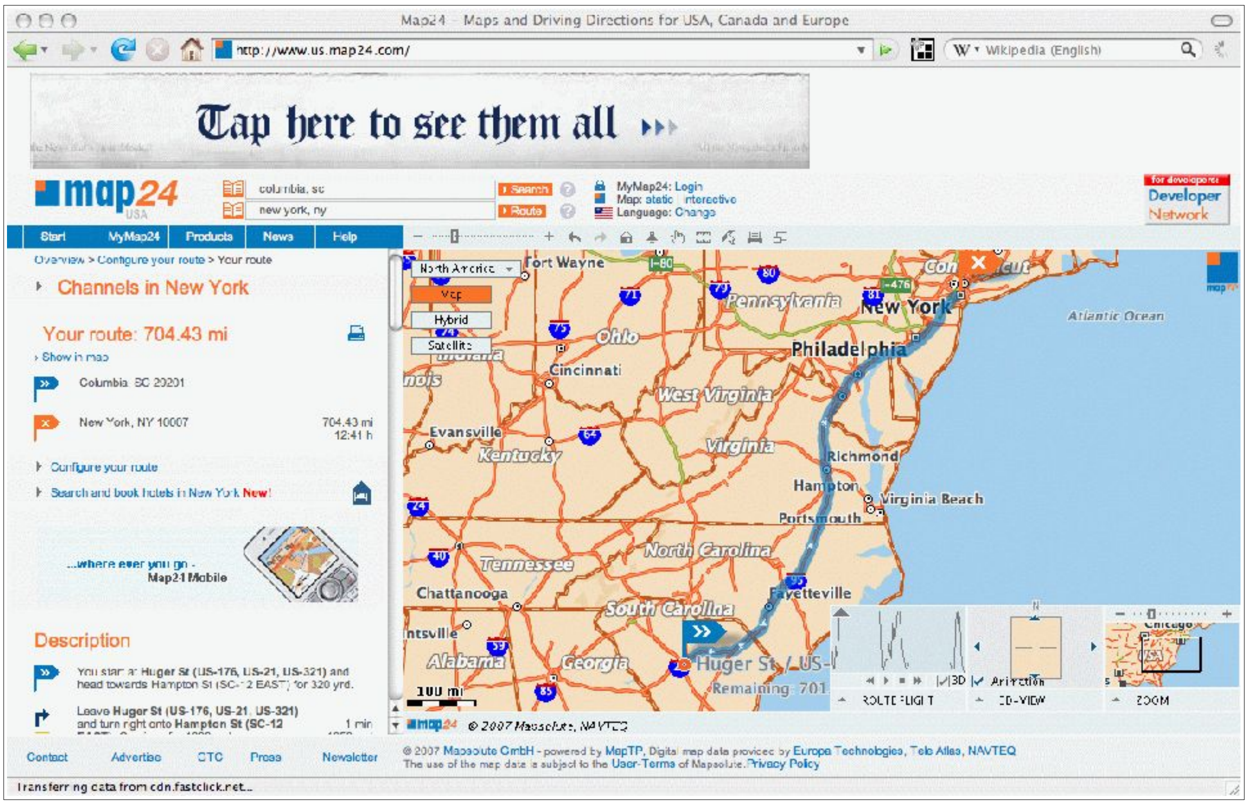


Figure 12: URL-problem example on map24.com

Assistive technologies problem

Another problem may occur to people that use assistive technologies. Normally screen readers “read” the page line by line, so it is likely that the screen reader never recognizes a change that has been made in a part of the web-page it has already read [Weba07]. Applications that always focus on changed data could confuse the reader of an article for example by jumping to different points every time something is updated on the page [Gibs06].

2.7.2 Security Risks in AJAX-Applications

As the flexibility of web-applications is rising by using the AJAX technology, some security problems are rising as well. Compared to classic web-applications that base upon the classic web-model, there are three major security issues [Ritc07]:

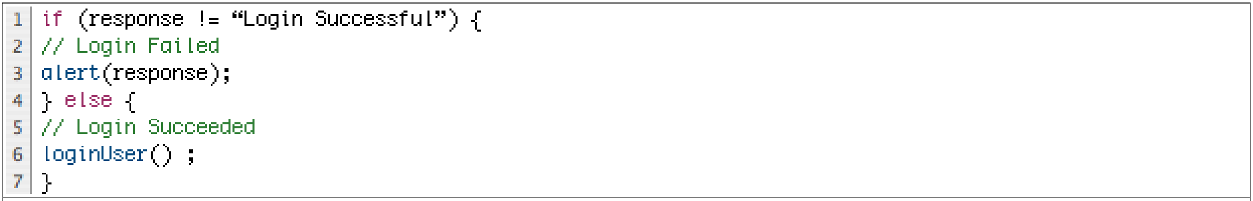
- client-side security controls
- increased attack surfaces

- new possibilities for Cross-Site Scripting

These issues are now going to be discussed.

Client-side security controls

As AJAX relies on a lot of client-side controls, many developers act frivolously and put security relevant code into the control of the client. The problem of this fact is that as code is executed on the client-side, every user has access to the code and is able to manipulate it [Hayr06]. To mention an example it is just necessary to imagine a simple login page. After entering username and password the site calls a JavaScript function instead directing all the information directly to the server. The client-side function that is called could for example validate the format of the input before the login is requested using the XMLHttpRequest object. When the server responds and the login information was correct, another JavaScript function is called that executes the application behind the login. The security risk of this procedure is that the user could simply call the JavaScript function after the login instead of going through the whole login process [Ritc07]. Figure 13 shows the critical JavaScript code where the *loginUser()* function could be called anytime without validating the login data first.



```
if (response != "Login Successful") {
// Login Failed
alert(response);
} else {
// Login Succeeded
loginUser() ;
}
```

Figure 13: Critical JavaScript Login Code

Increased attack surfaces

Through the use of AJAX most web-applications consist of many small applications, such as looking up the city of a customer when entering the ZIP-Code. Each of these small applications is a security risk of its own thus it is necessary to consider the security aspects of every of those small applications. This is leading from a single point of entry to a multiple point of entry concept which offers more vulnerabilities [Hayr06].

New possibilities for Cross-Site Scripting

Cross-Site Scripting, also called XSS, includes all the efforts to infiltrate malicious code into a web-application that is later on executed on the client side. This is typically possible where HTML or JavaScript Code is accepted as an input to a web-page that later returns the values to the browser. For example a clickable URL containing malicious JavaScript Code could be created and sent by email to somebody. As the recipient clicks on the URL the malicious code is executed and the attacker could steal the session, create a fake login, log the keystrokes or execute any other Script Code [Ritc07].

For several reasons, the risks of Cross-Site Scripting are rising through the usage of AJAX components. Usually XSS lasts as long as a page is loaded. One characteristic of AJAX applications is that there is mostly only one page loaded so that a permanent XSS could be created. In this scenario an attacker for example would be able to log all the keystrokes made within the application [Ritc07].

3 The Model-View-Controller pattern

As both Ruby on Rails as well as Apache MyFaces are based on the Model-View-pattern it is crucial to introduce this concept briefly.

“The Model-View-Controller (MVC) pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes” [Micr07]

The duties of the three classes are as follows [Micr07]:

- **Model**: The model contains the data of the application and is responsible to respond to requests usually from the View and to respond to changes in the state usually from the Controller.
- **View**: The presentation of the information.
- **Controller**: The Controller receives the input from the user and informs the Model respectively the View of the changes.

Figure 14 illustrates the MVC pattern and the interactions of the three parts among each other.

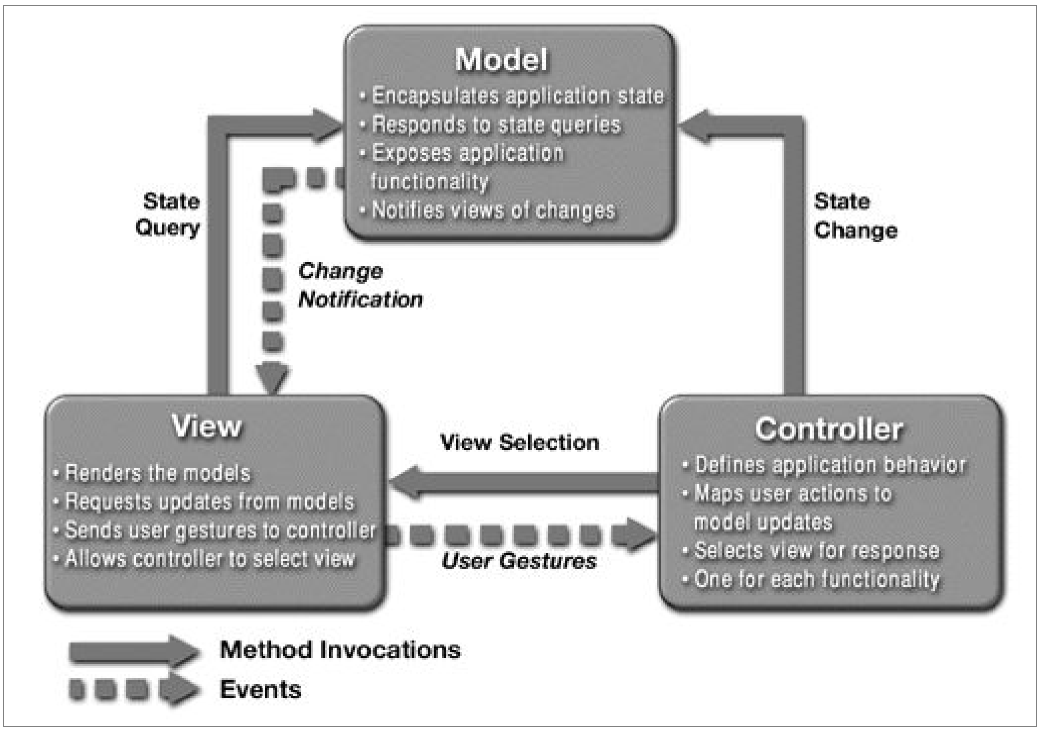

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
State Change
Change Notification
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Figure 14: The Model-View-Controller pattern [SunM02]

One important fact of the MVC is that the View as well as the Controller depend on the Model but the Model depends on neither the Controller nor the View. This allows the building and testing of the Model without the View or the Controller [Micr07].

Advantages resulting from this pattern are a reduction of duplicate code as well as code that is easier to maintain. As the business logic is kept separately, adding new data sources as well as changes in the presentation of data is easier [SunM02].

4 Ruby on Rails

“Ruby on Rails (or more commonly, just Rails) is a full-stack MVC web development framework for the Ruby language.” [Raym07, page 8]

The term full-stack stands for the fact, that the framework includes nearly everything that is necessary to create a finished product. At least this is true on the application layer, because mostly a database is needed as well as a web server [Raym07].

This chapter will give a few historical aspects of Ruby and Ruby on Rails as well as the capabilities to implement AJAX components and will provide some code examples to give a closer look what code looks like in Ruby on Rails applications. Ruby on Rails is released under the MIT license while Ruby itself is licensed under the Ruby License [Ruby07]. The official link to the Ruby on Rails web page is http://www.rubyonrails.org/.

The history behind Ruby on Rails is not very extensive as Ruby on Rails was first developed and released in 2004 by David Heinemeier Hansson who was born in Copenhagen, Denmark in 1979 and nowadays works at 37signals, a web-application company. The first stable release was in 2005 [Hein07].

Ruby itself, which is the basis for Ruby on Rails, is an object-oriented programming language with roots in List, Perl and Smalltalk. Ruby is developed by Yukihiro Matsumoto and was first released in 1995 [Raym07].

In this paper the Ruby on Rails Version 1.2.3 was used with Ruby Version 1.8.6. On December 17 2007 the Version 2.0 of Ruby on Rails was released which was too late to analyze all the changes between the different versions.

4.1 AJAX-Components

Ruby on Rails and AJAX correlate in two different ways. On the one hand there are two JavaScript frameworks namely *Prototype* and *script.aculo.us*. Both of them come along with Ruby on Rails and are developed with Ruby on Rails but are also available in other programming languages like PHP or Java. Prototype provides access to the XMLHttpRequest Object as well as methods for manipulations using DOM and JavaScript data structures. script.aculo.us works on the top of Prototype and has its strengths in visual effects and advanced user-interface options, like drag and drop [Raym07].

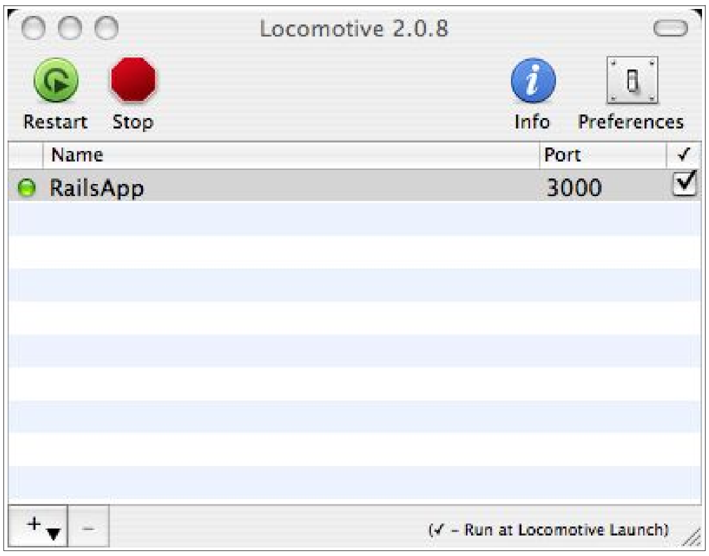
The second way are the Rails helpers which are Ruby methods that are called within the controller and later invoke JavaScript functions in Prototype and script.aculo.us. Those make it possible to create extensive AJAX applications without coding JavaScript itself. Especially important regarding the title of this paper are the so called Rails Helper callbacks which give the possibility to make things happen during the life-cycle of a XMLHttpRequest. In detail there are eight different callbacks that define different states of the life-cycle of the request. An example of these callbacks will be given later [Raym07].

In the following sections some code examples of Ruby on Rails AJAX applications will be provided. Later on the two JavaScript Libraries Prototype and script.aculo.us will be presented and code examples will be given.

4.2 Installation

As the author of this paper uses Mac OS X 10.4 there are two major possibilities to install the Ruby on Rails framework. The latest version of Mac OS X 10.5 also known as Leopard, already ships with Ruby on Rails [Augu07]. The first possibility is to download and compile the components on the machine by following the instructions that can be found on http://hivelogic.com/articles/ruby-rails-mongrel-mysql-osx. The primary steps are to install Ruby first and then install the Ruby package manager called *RubyGems* that will install then Ruby on Rails. The second option to get Ruby on Rails onto a Mac OS X 10.4 is to use the pre-packaged tool called *Locomotive* which contains Ruby, Rails and some other tools [Loco07]. As this papers focus is more on the AJAX capabilities of Ruby on Rails than on Ruby on Rails itself, the author decided to install Locomotive. After installing Locomotive, it provides a simple user interface that allows to control the current projects which is shown in Figure 15.

For Windows there is also a pre-packaged tool called *InstantRails* which comes with Ruby, Ruby on Rails, Apache Web-Server and MySQL [Inst07].

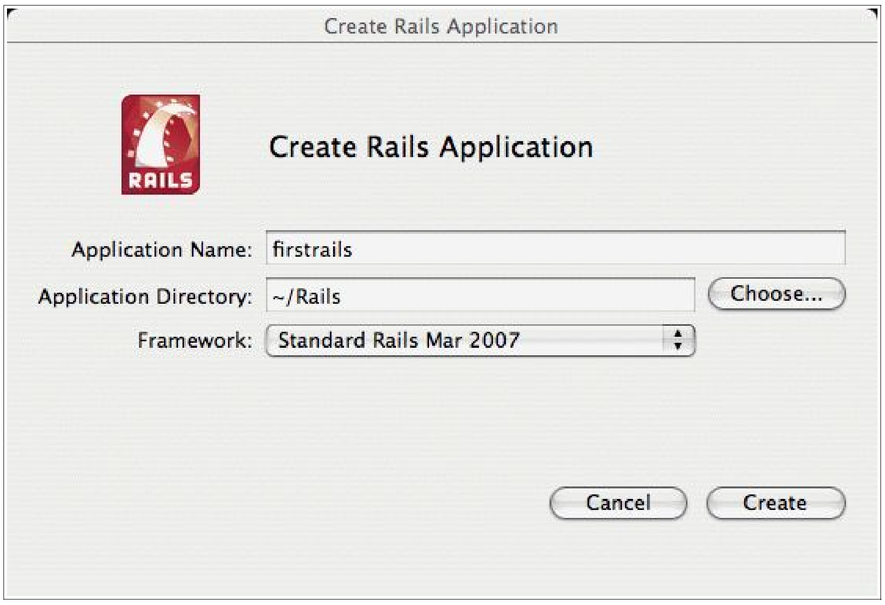


Name	Port	✓
RailsApp	3000	✓

Figure 15: Screenshot of Locomotive

4.2.1 First Ruby on Rails Project

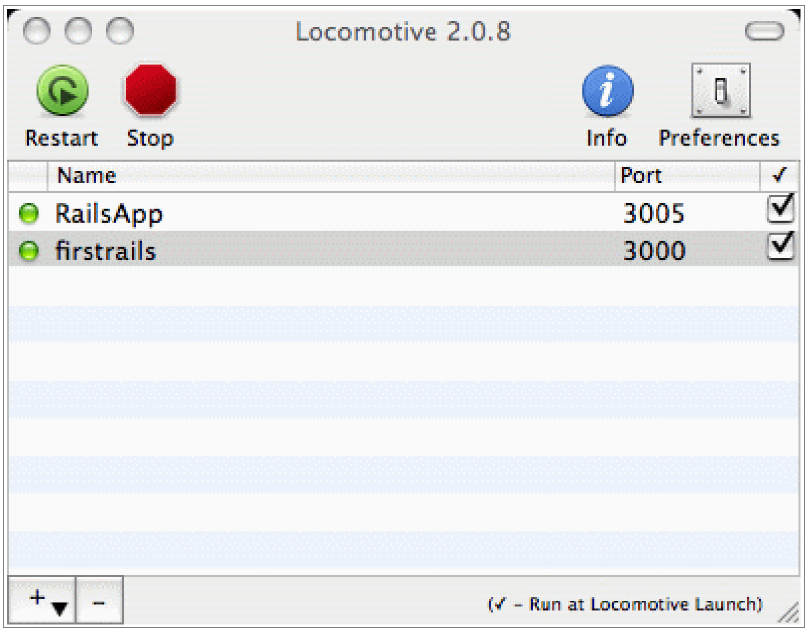
Most of the Ruby on Rails commands are executed in the terminal, so this section will show how to create a project and how to implement simple applications. The first step is to create a skeleton which can be referred as a project. The skeleton is called *firstrails*.



Application Name: firstrails
Application Directory: ~/Rails
Framework: Standard Rails Mar 2007

Figure 16: Screenshot of Locomotive when creating a new application

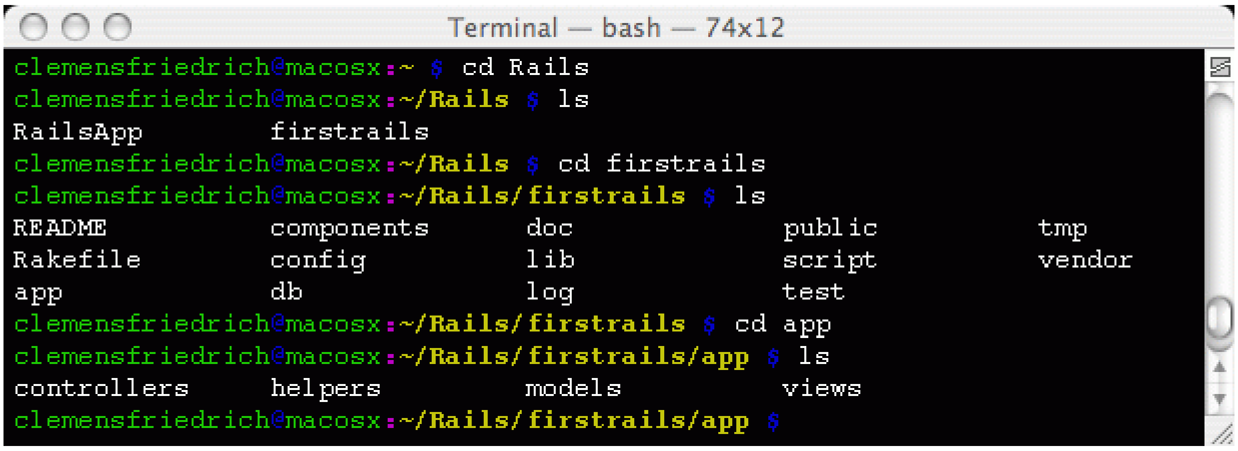
Figure 16 shows how to create an application in Locomotive while Figure 17 shows the status when the application is created and port 3000 is assigned to the application.



Name	Port	✓
● RailsApp	3005	✓
● firstrails	3000	✓

Figure 17: Screenshot of Locomotive containing the new application

Locomotive calls the rails functions that create all the necessary files and folders. Figure 18 shows how the file structure of Ruby on Rails applications look like. The root folder for Ruby on Rails applications is `~/Rails` where the folder `firstrails` was created. The most important folder in the project is the `app` folder, where all the Rails specific code is saved in. As Figure 18 shows at the bottom the `app` folder contains subfolders that are called `controllers`, `models` and `views` which refers to the MVC pattern.

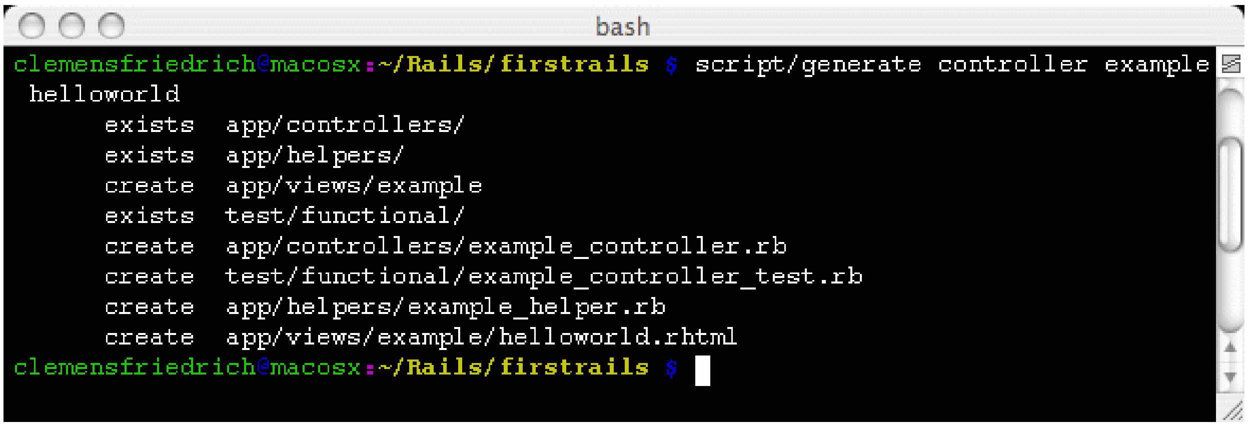


```
clemensfriedrich@macosx:~ $ cd Rails
clemensfriedrich@macosx:~/Rails $ ls
RailsApp        firstrails
clemensfriedrich@macosx:~/Rails $ cd firstrails
clemensfriedrich@macosx:~/Rails/firstrails $ ls
README          components      doc             public          tmp
Rakefile        config          lib             script          vendor
app             db              log             test
clemensfriedrich@macosx:~/Rails/firstrails $ cd app
clemensfriedrich@macosx:~/Rails/firstrails/app $ ls
controllers     helpers         models          views
clemensfriedrich@macosx:~/Rails/firstrails/app $
```

Figure 18: Terminal-Output of the Ruby on Rails structure of files and folders

4.3 Code Examples in Ruby on Rails

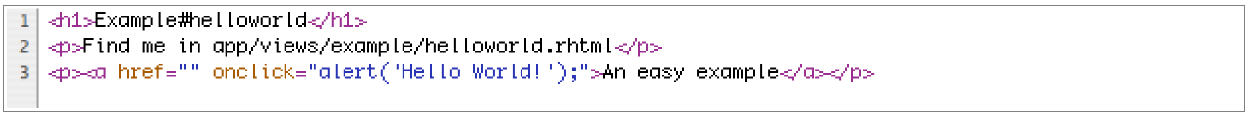
Based on the setup of a new application in the previous section this section will show some code examples in Ruby on Rails. First of all an example is shown how to create the classical “Hello World” application. The first step is to create a new controller referring to the MVC pattern of the previous chapter. Figure 19 shows the shell commands that have to be executed to create the controller with the name `example` and an action with the name `helloworld`.



```
clemensfriedrich@macosx:~/Rails/firstrails $ script/generate controller example
 helloworld
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/example
      exists  test/functional/
      create  app/controllers/example_controller.rb
      create  test/functional/example_controller_test.rb
      create  app/helpers/example_helper.rb
      create  app/views/example/helloworld.rhtml
clemensfriedrich@macosx:~/Rails/firstrails $
```

Figure 19: Terminal commands to create helloworld example

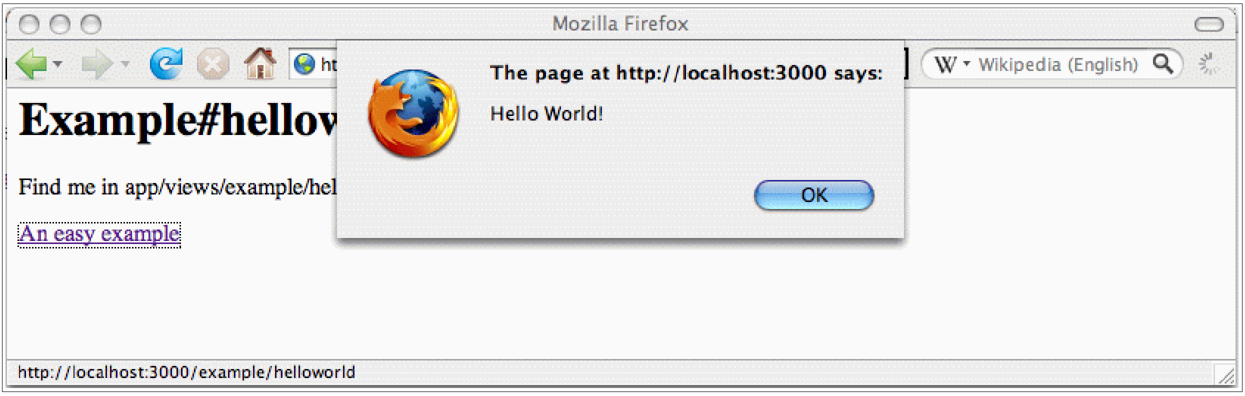
After creating the controller the view which is located in the subfolder `app/views/example/helloworld.rhtml` has to be modified. Figure 20 shows the code that includes a hyperlink which calls a JavaScript alert box.



```
<h1>Example#helloworld</h1>
<p>Find me in app/views/example/helloworld.rhtml</p>
<p><a href="" onclick="alert('Hello World!');">An easy example</a></p>
```

Figure 20: Code in the helloworld.rhtml file

After saving the file, the “Hello World!” page can be opened in a web-browser as Figure 21 shows. The URL to the page consisits of the controller followed by the action, in this case the URL is http://localhost:3000/example/helloworld. Clicking the link that is shown on the page, the JavaScript alert box is opened which shows the “Hello World!”.



Mozilla Firefox

Example#hellov

Find me in app/views/example/hel

An easy example

The page at http://localhost:3000 says:

Hello World!

OK

http://localhost:3000/example/helloworld

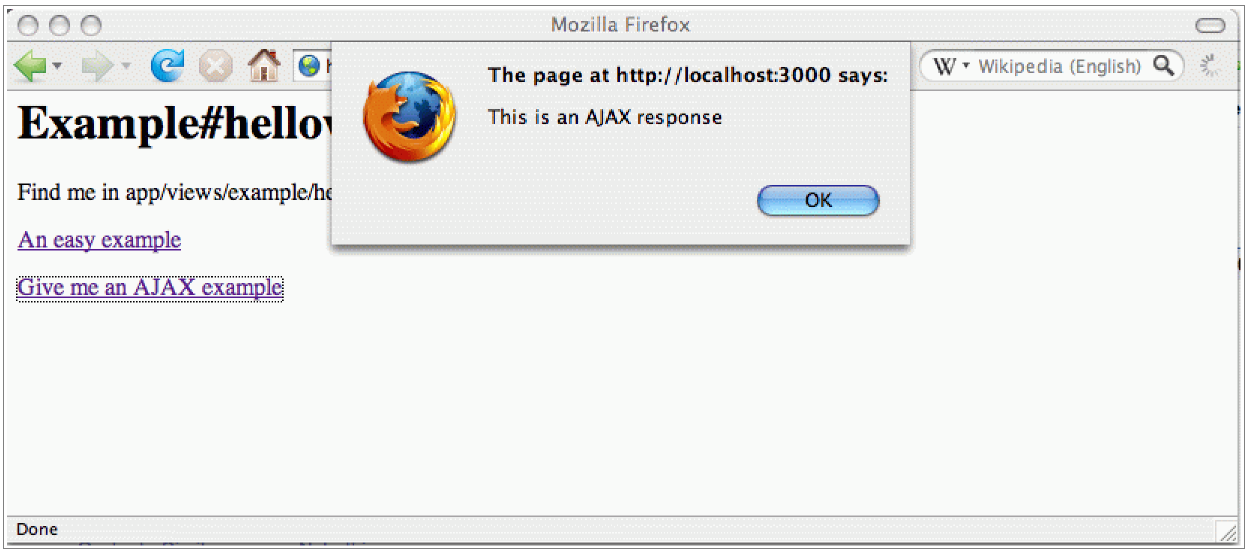
Figure 21: Screenshot of the "Hello World" page

Now it is time to create a simple AJAX application, that means that we are using JavaScript and the XMLHttpRequest Object to communicate with the server. Therefore it is necessary to create a new action named `ajaxresponse` in the controller `example` that is located in the `app/controllers/example_controller.rb file`. Furthermore a new view has to be created in the views folder that is called `ajaxresponse.rhtml`. This file just contains a string that will be “This is an AJAX response”. Figure 22 shows the code of the `ajaxresponse.rhtml` file while Figure 23 shows the web-browser page after clicking on the hyperlink that says “Give me an AJAX example”. The specialty of this example is that the JavaScript Code creates an XMLHttpRequest Object to request the string that appears in the alert box from the server.

```
<h1>Example#helloworld</h1>
<p>Find me in app/views/example/helloworld.rhtml</p>
<p><a href="#" onclick="alert('Hello World!');">An easy example</a></p>
<p><a href="#" onclick="serverSideAlert();">Give me an AJAX example</a></p>
<script type="text/javascript">
    function serverSideAlert() {
        function getRequestObject() {
            try { return new XMLHttpRequest() } catch (e) {}
            try { return new ActiveXObject("Msxml2.XMLHTTP") } catch (e) {}
            try { return new ActiveXObject("Microsoft.XMLHTTP") } catch (e) {}
            return false
        }
        var request = getRequestObject();
        request.open('get','/example/ajaxresponse', true);
        request.onreadystatechange = function() {
            if(request.readyState == 4) alert(request.responseText);
        }
        request.send(null);
    }
</script>
```

Figure 22: Code of the helloworld.rhtml file

The function `getRequestObject()` checks if the web-browser supports the typical XMLHttpRequest object or if it supports the XMLHttpRequest Object as an ActiveX Object as Microsoft's Internet Explorer 6 does [Raym07].



The page at http://localhost:3000 says:

This is an AJAX response

Figure 23: Screenshot of the AJAX response in the web-browser

This example was a very basic one that did not make use of the JavaScript Libraries Prototype and script.aculo.us that Ruby on Rails supports. In the following sections the two named Libraries are introduced and code examples are given how to use them with Ruby on Rails.

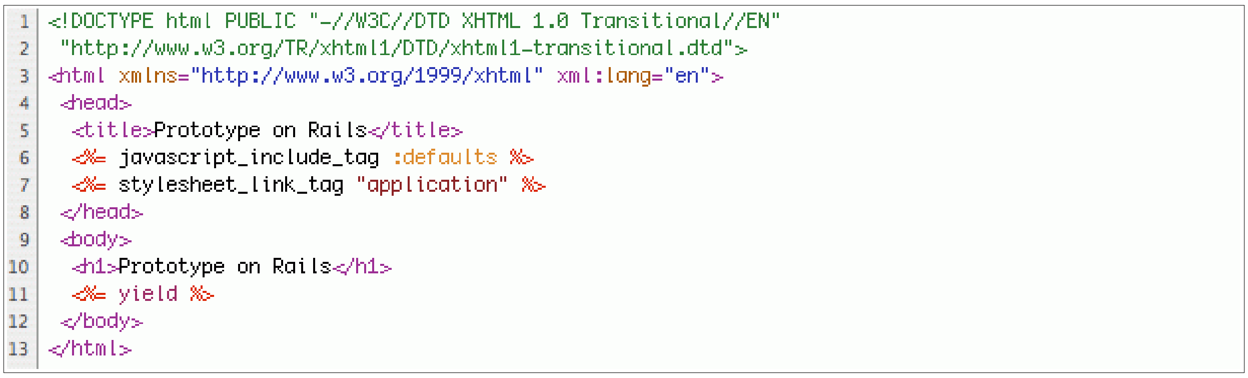
4.4 Prototype

Prototype is a JavaScript Framework developed by Sam Stephenson that extends core JavaScript classes and implements additional ones to provide new features. Its focus is especially on AJAX and DOM manipulations. The possibilities it supports can be divided into four major sections [Raym07]:

- AJAX support (wrappers for the XMLHttpRequest Object)
- DOM manipulation
- form manipulation (DOM manipulations especially for forms)
- core extensions (especially for working with data structures)

As Ruby on Rails ships with full Prototype support, the following section will present some code examples how to implement AJAX components with Prototype in Ruby on Rails using the Rails Helpers that have been mentioned before.

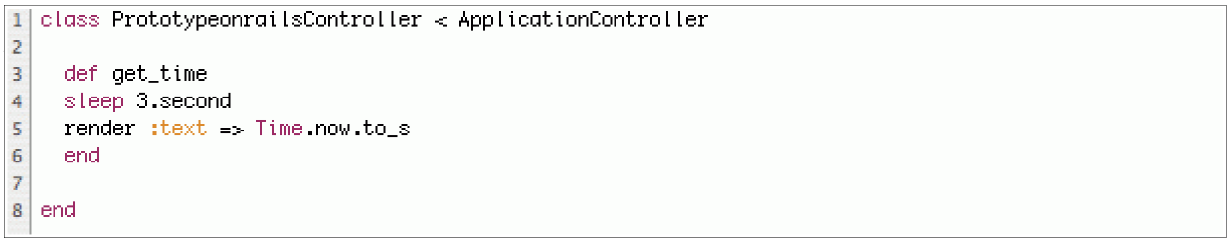
For this section a new controller is created while the old application called *firstrails* will be still used. The new controller will be called *prototypeonrails* with an action called *get_time*. This example is going to show a simple web-page that provides a link which requests the time from the server using AJAX. To provide a nicer look of the web-page, the author adopted the Cascading Style Sheet that is used in the book “AJAX on Rails” from Scott Raymond. The Style Sheets are saved in the `public/stylesheets/` folder of the Rails application. Furthermore a simple layout file was created, which is saved in the `app/views/layouts/` folder and is called `application.rhtml`. Figure 24 shows the layout file which starts with a common XHTML definition. The specialties are definitely in line 6, 7 and 11. The tags `<%= ... %>` define a Ruby Expression. The one in line 6 defines that the JavaScript Libraries Prototype and script.aculo.us will be included, the one in line 11 defines where the dynamic content will be inserted.



```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
4   <head>
5    <title>Prototype on Rails</title>
6    <%= javascript_include_tag :defaults %>
7    <%= stylesheet_link_tag "application" %>
8   </head>
9   <body>
10   <h1>Prototype on Rails</h1>
11   <%= yield %>
12  </body>
13 </html>
```

Figure 24: Code of the layout file appliaction.rhtml

Figure 25 shows the code of the controller that was created previously and defines the method `get_time`. First there is a sleep period of three seconds followed by rendering the actual time of the server.

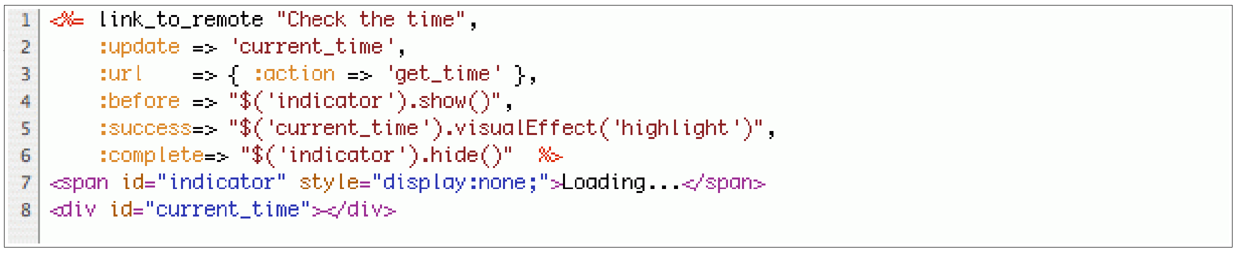


```
class PrototypeonrailsController < ApplicationController

  def get_time
  sleep 3.second
  render :text => Time.now.to_s
  end

end
```

Finally Figure 26 presents the code of the `index.rhtml` file which is located in the `app/views/prototypeonrails/` folder.



```
1 <%= link_to_remote "Check the time",
2     :update => 'current_time',
3     :url    => { :action => 'get_time' },
4     :before => "$('indicator').show()",
5     :success=> "$('current_time').visualEffect('highlight')",
6     :complete=> "$('indicator').hide()"  %>
7 <span id="indicator" style="display:none;">Loading...</span>
8 <div id="current_time"></div>
```

This file defines the JavaScript functionalities. Actually Prototype is used, but the code in the Figure are the Rails helpers, that have been mentioned in the Ruby on Rails introduction section. The code consists of a link that will be created by using the `link_to_remote` command. The commands starting with a colon are options of the Rails helper. The options :`before`, :`success` and :`complete` are the so called Rails Helper callbacks that have been introduced earlier. The `$()` function is a wrapper for the DOM method `document.getElementById` that offers some advantages like the possibility to hand it more than just one argument. But in general it defines the place where the content should be inserted [Raym07]. So line 4 defines that before the XMLHttpRequest Object is created the span element in line 7 should be shown. But now it is time to look at the output in the web-browser.

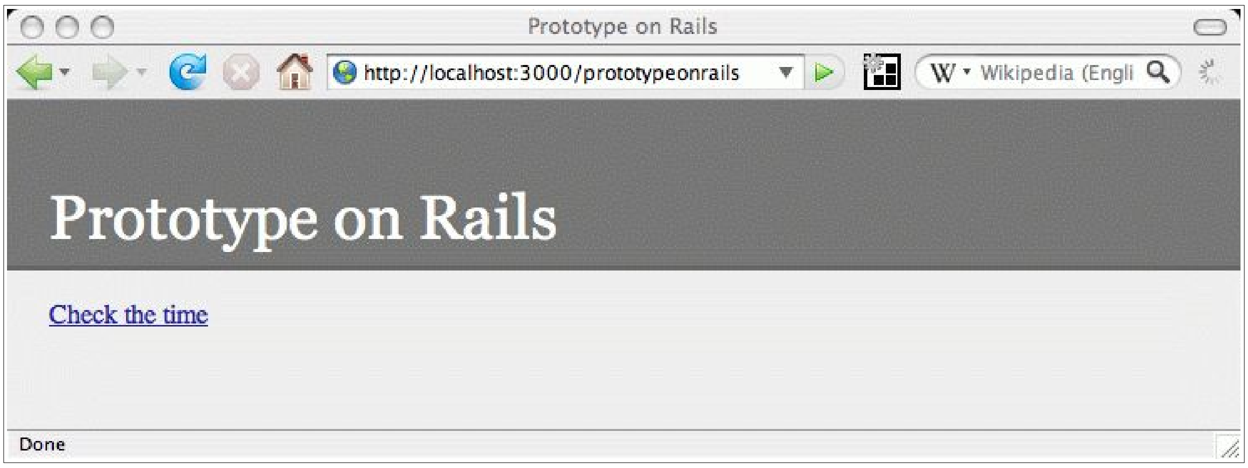
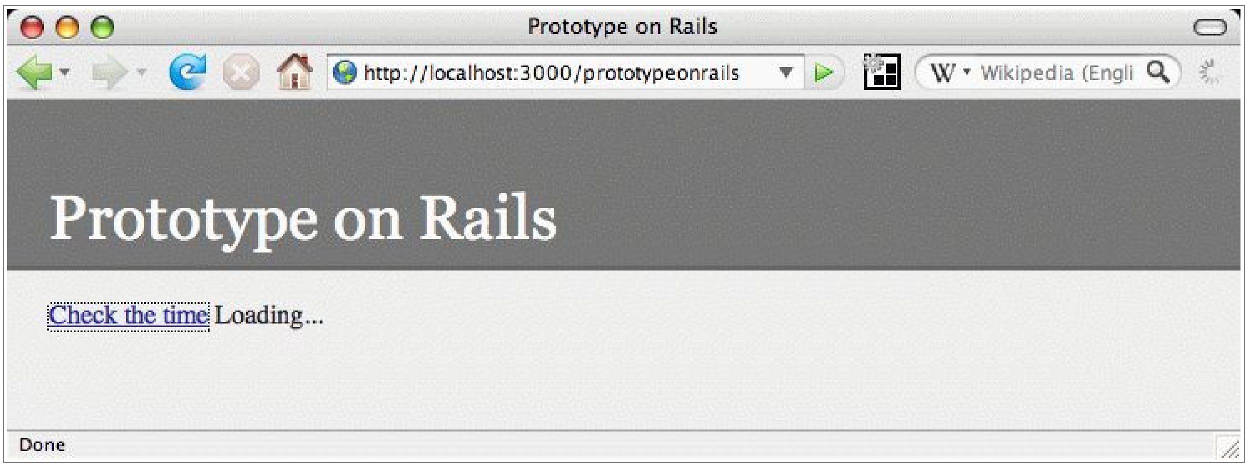


Figure 27: Initial State of prototypeonrails

Figure 27 shows the initial state of the application that presents a link that has been clicked in Figure 28. In this figure it is also possible to see the `span` element that is now visible until the XMLHttpRequest is complete.

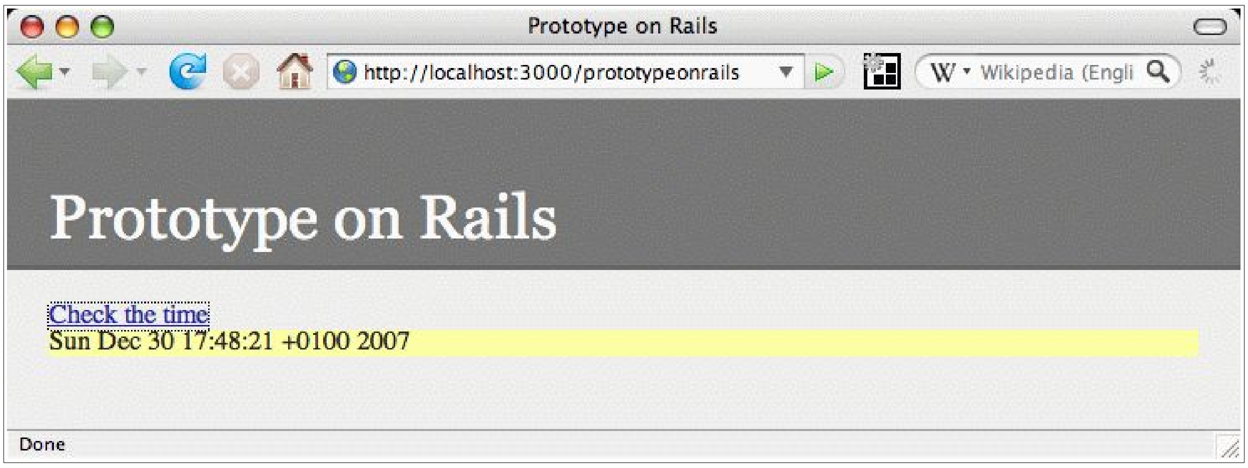


Prototype on Rails

Check the time Loading...

Figure 28: Loading State of prototypeonrails

Figure 29 shows the final state in which the final state is presented. The time of the server is retrieved and presented in the `div` element with the `id="current_time"` is shown.

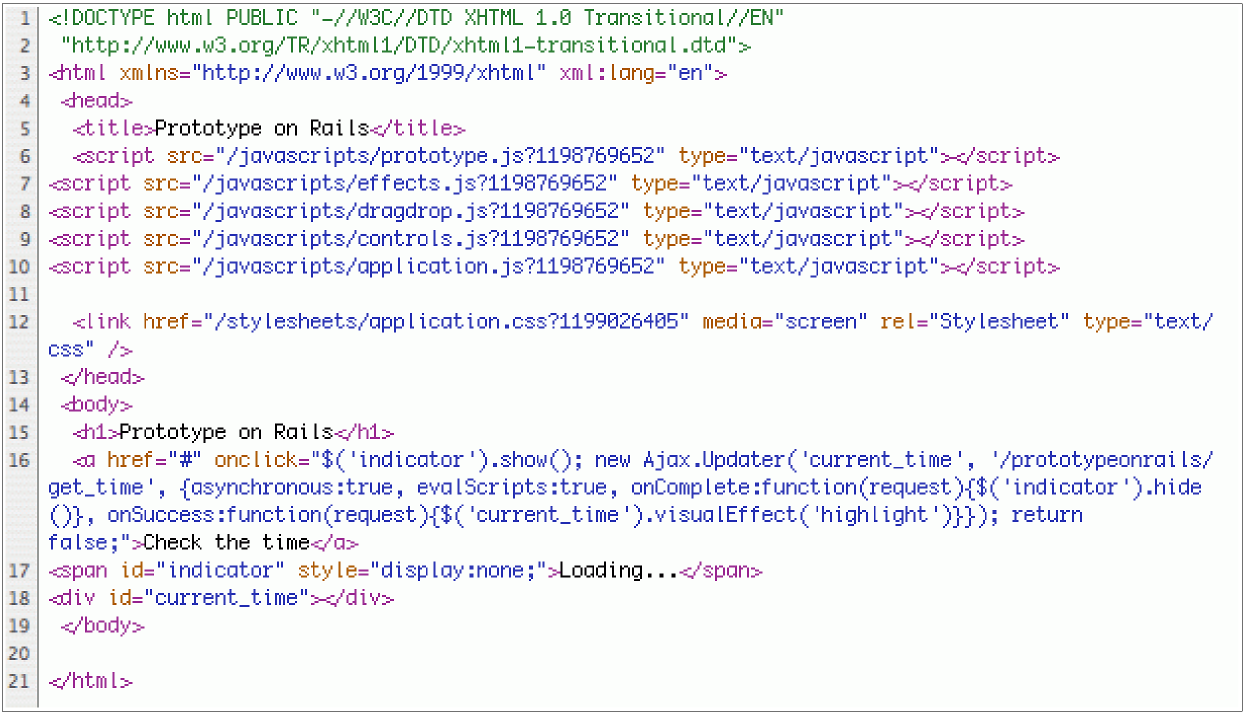


Prototype on Rails

Check the time
Sun Dec 30 17:48:21 +0100 2007

Figure 29: Final State of prototypeonrails

Finally it is interesting to see the HTML Code of the page that comes from the server because in this example Prototype itself was not used directly because the Rails Helpers did their work. Figure 30 shows the HTML file that was generated by the server. In line 6 it can be seen that the JavaScript library Prototype is loaded and used for example in line 16 with the `Ajax.Updater()` function which is a Prototype function.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <title>Prototype on Rails</title>
  <script src="/javascripts/prototype.js?1198769652" type="text/javascript"></script>
<script src="/javascripts/effects.js?1198769652" type="text/javascript"></script>
<script src="/javascripts/dragdrop.js?1198769652" type="text/javascript"></script>
<script src="/javascripts/controls.js?1198769652" type="text/javascript"></script>
<script src="/javascripts/application.js?1198769652" type="text/javascript"></script>

  <link href="/stylesheets/application.css?1199026405" media="screen" rel="Stylesheet" type="text/css" />
 </head>
 <body>
  <h1>Prototype on Rails</h1>
  <a href="#" onclick="$('indicator').show(); new Ajax.Updater('current_time', '/prototypeonrails/get_time', {asynchronous:true, evalScripts:true, onComplete:function(request){$('indicator').hide()}, onSuccess:function(request){$('current_time').visualEffect('highlight')}}); return false;">Check the time</a>
<span id="indicator" style="display:none;">Loading...</span>
<div id="current_time"></div>
 </body>

</html>
```

Figure 30: HTML file generated by the server for prototypeonrails

As this section dealt with the Rails helpers and the JavaScript Library Prototype the next section will cover to the JavaScript Library script.aculo.us and the Rails Helpers that use it.

4.5 script.aculo.us

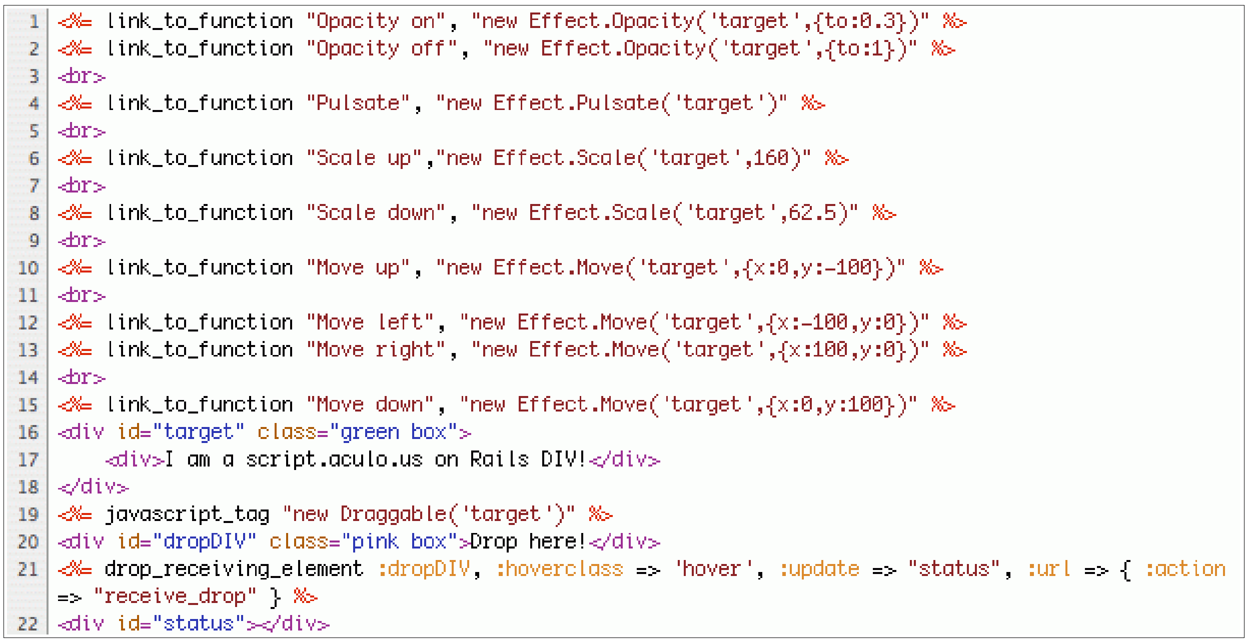
The development of script.aculo.us and Prototype happens in concert with Ruby on Rails and is very close. Some parts that were Prototype parts in the beginning are now parts of script.aculo.us which is built on Prototype. But anyway, script.aculo.us has a different goal, it is designed to provide the following features [Raym07]:

- Visual Effects
- Transitions
- Drag-and-Drop elements

The script.aculo.us Framework works with most modern web-browsers that are Microsoft Internet Explorer 6 and up, Mozilla Firefox, Apple Safari and most Linux web-browsers like Konqueror [Raym07].

As a code example was given for Prototype used with Ruby on Rails an example of script.aculo.us in Ruby on Rails is given as well. It will contain examples for the Visual Effects and for the Drag-and-Drop functionality. script.aculo.us offers five core effects that control the fundamental attributes of an element which are `Opacity`, `Scale`, `Move`, `Highlight` and `Parallel`. Most of them will be used in the code example. Transition is an option to determine the pattern of a specific change. It was left out because the relevance is not too important for simple AJAX applications [Raym07]. The code itself will make use of the Rails helpers as well as script.aculo.us code itself. At the end a look at the HTML file will be given.

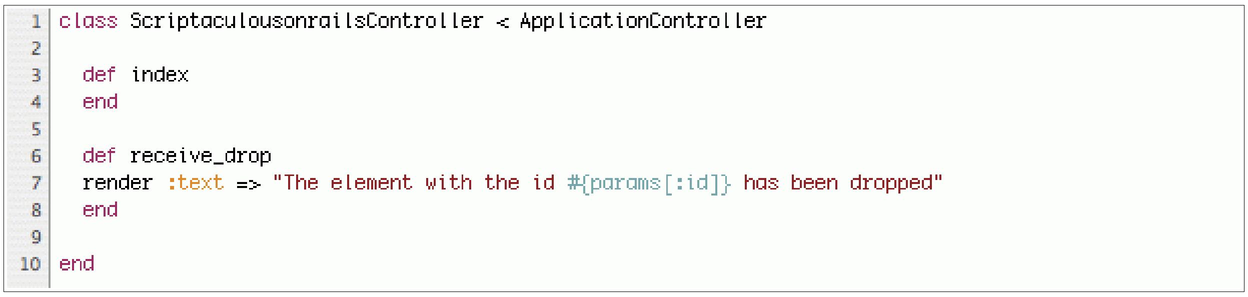
Again the same application called *firstrails* is going to be used but a new controller called *scriptaculousonrails* is going to be created. The layout from the previous example is used again. Figure 31 shows the code of the new `index.rhtml` which contains the HTML, JavaScript and Rails Helper code.



```
<%= link_to_function "Opacity on", "new Effect.Opacity('target',{to:0.3})" %>
<%= link_to_function "Opacity off", "new Effect.Opacity('target',{to:1})" %>
<br>
<%= link_to_function "Pulsate", "new Effect.Pulsate('target')" %>
<br>
<%= link_to_function "Scale up","new Effect.Scale('target',160)" %>
<br>
<%= link_to_function "Scale down", "new Effect.Scale('target',62.5)" %>
<br>
<%= link_to_function "Move up", "new Effect.Move('target',{x:0,y:-100})" %>
<br>
<%= link_to_function "Move left", "new Effect.Move('target',{x:-100,y:0})" %>
<%= link_to_function "Move right", "new Effect.Move('target',{x:100,y:0})" %>
<br>
<%= link_to_function "Move down", "new Effect.Move('target',{x:0,y:100})" %>
<div id="target" class="green box">
    <div>I am a script.aculo.us on Rails DIV!</div>
</div>
<%= javascript_tag "new Draggable('target')" %>
<div id="dropDIV" class="pink box">Drop here!</div>
<%= drop_receiving_element :dropDIV, :hoverclass => 'hover', :update => "status", :url => { :action
=> "receive_drop" } %>
<div id="status"></div>
```

Figure 31: Code of the index.rhtml of scriptaculousonrails

The `link_to_function` Helper that is used very often creates a hyperlink which directly calls JavaScript functions which are for example `Effect.Opacity` or `Effect.Move`. These functions are part of the script.aculo.us Framework. All the `Effect.*` functions refer to the div element *target* and make possible to control the div element like to move it or do scale it up or down. All controls can be seen in BILD. Line 19 defines that the div *target* is a draggable item while in line 21 the Rails Helper `drop_receiving_element` defines that the div with the id *dropDIV* is able to receive a dropped element and sets the action `receive_drop` when an item is dropped on it. Figure 32 shows the `receive_drop` method that was added in the controller of the application that is called `scriptaculousonrails_controller.rb`.



```
 1 class ScriptaculousonrailsController < ApplicationController
 2 
 3   def index
 4   end
 5 
 6   def receive_drop
 7   render :text => "The element with the id #{params[:id]} has been dropped"
 8   end
 9 
10 end
```

Figure 32: Code of the controller scriptaculousonrails_controller.rb

The method defines to return the string that can be seen in line 7. After the `receive_drop` request is complete, which is an AJAX call by the way, the last div in line 22 with the id *status* is updated. But let's have a look on the output in the web-browser. Again there will be three screenshots provided in the different states.

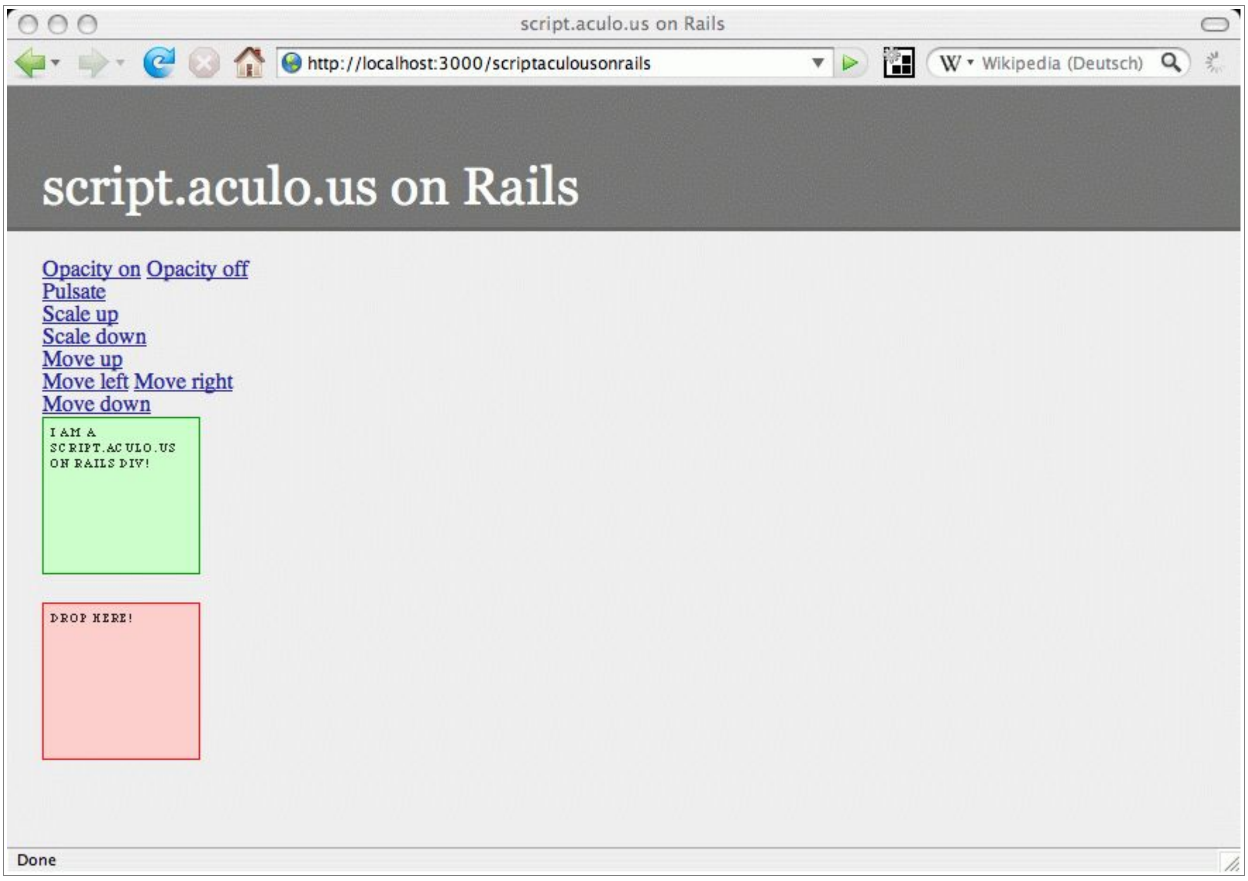


Figure 33: Initial state of scriptaculousonrails

Figure 33 presents the links to the JavaScript functions as well as the green div which is draggable and the pink one that is droppable. Figure 34 presents the green div, which actually has the *target* id, moved and resized.

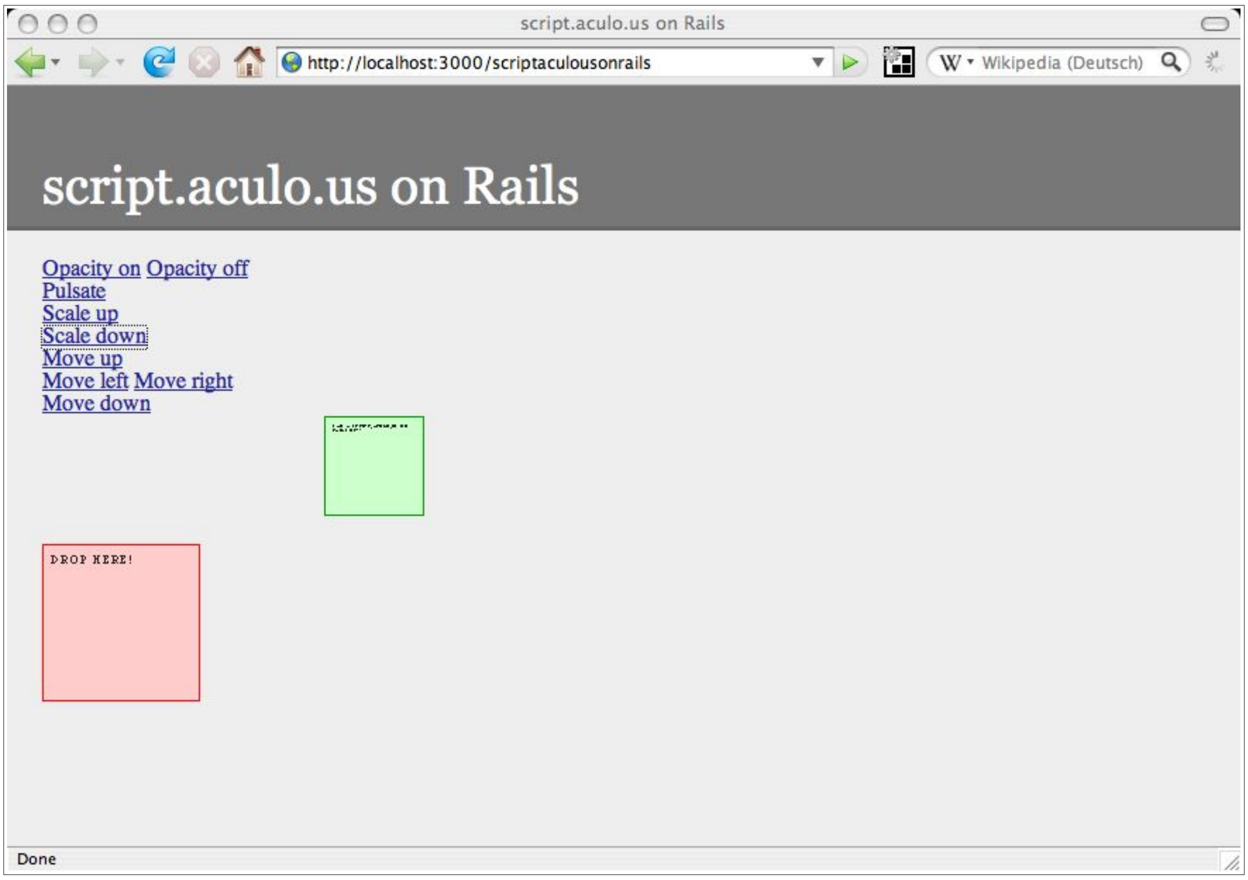
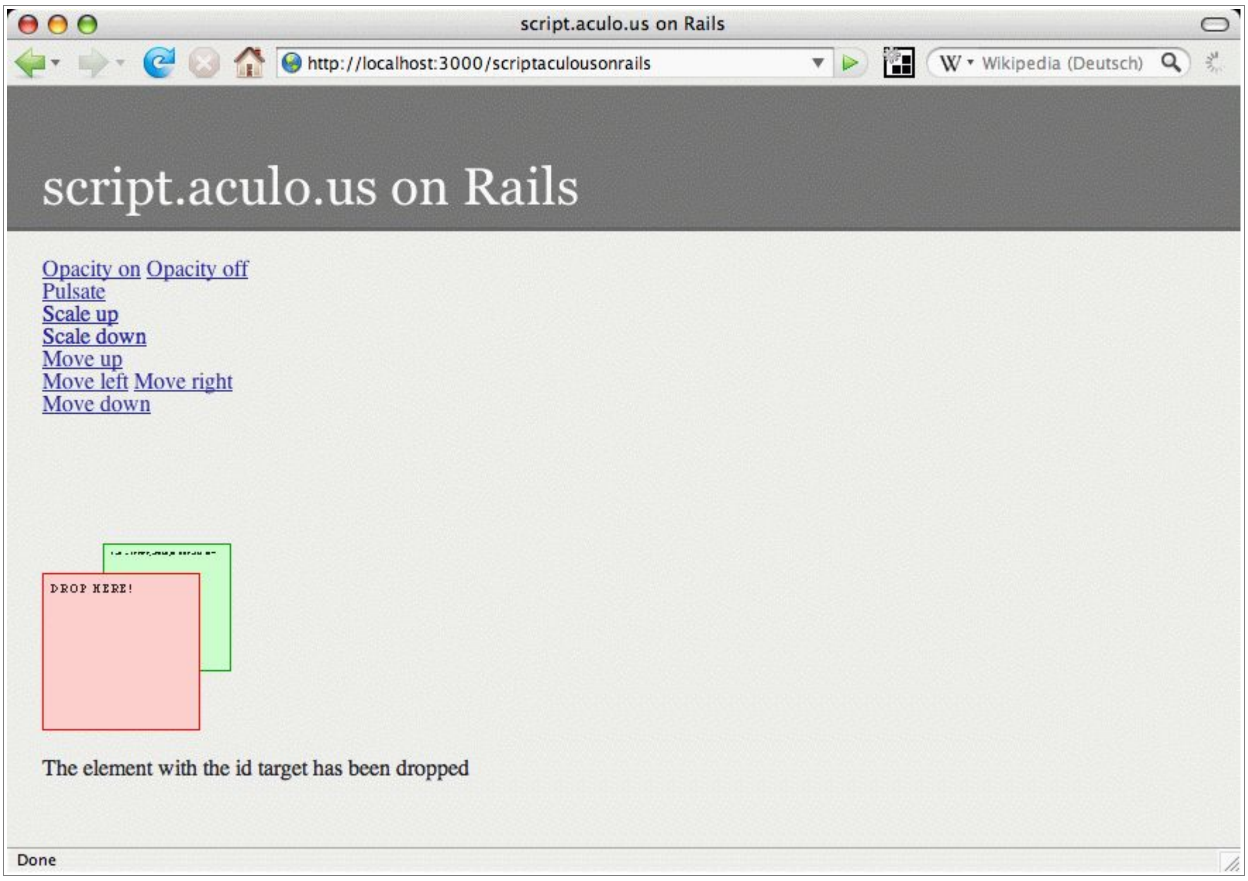


Figure 34: Second State of scriptaculousonrails with a moved "target" div

Figure 35 presents the final state where the green div was dragged and dropped in the pink div. After the drop, a AJAX call has been sent to request the `receive_drop` method. As the response from the server arrived the last div *status* has been updated with the text that was just received. Finally, it is interesting to have a look on the HTML file that is sent by the server, which is presented in Figure 36. Line 6 to 10 show that the JavaScript Frameworks Prototype and script.aculo.us are loaded. Line 15 to 29 show the Visual Effect functions of script.aculo.us which have been called directly instead of using Rails Helpers. Line 49 on the other side shows the use of the Rails Helper, if the output code is compared with the code in the view file in Figure 31 Line 21.



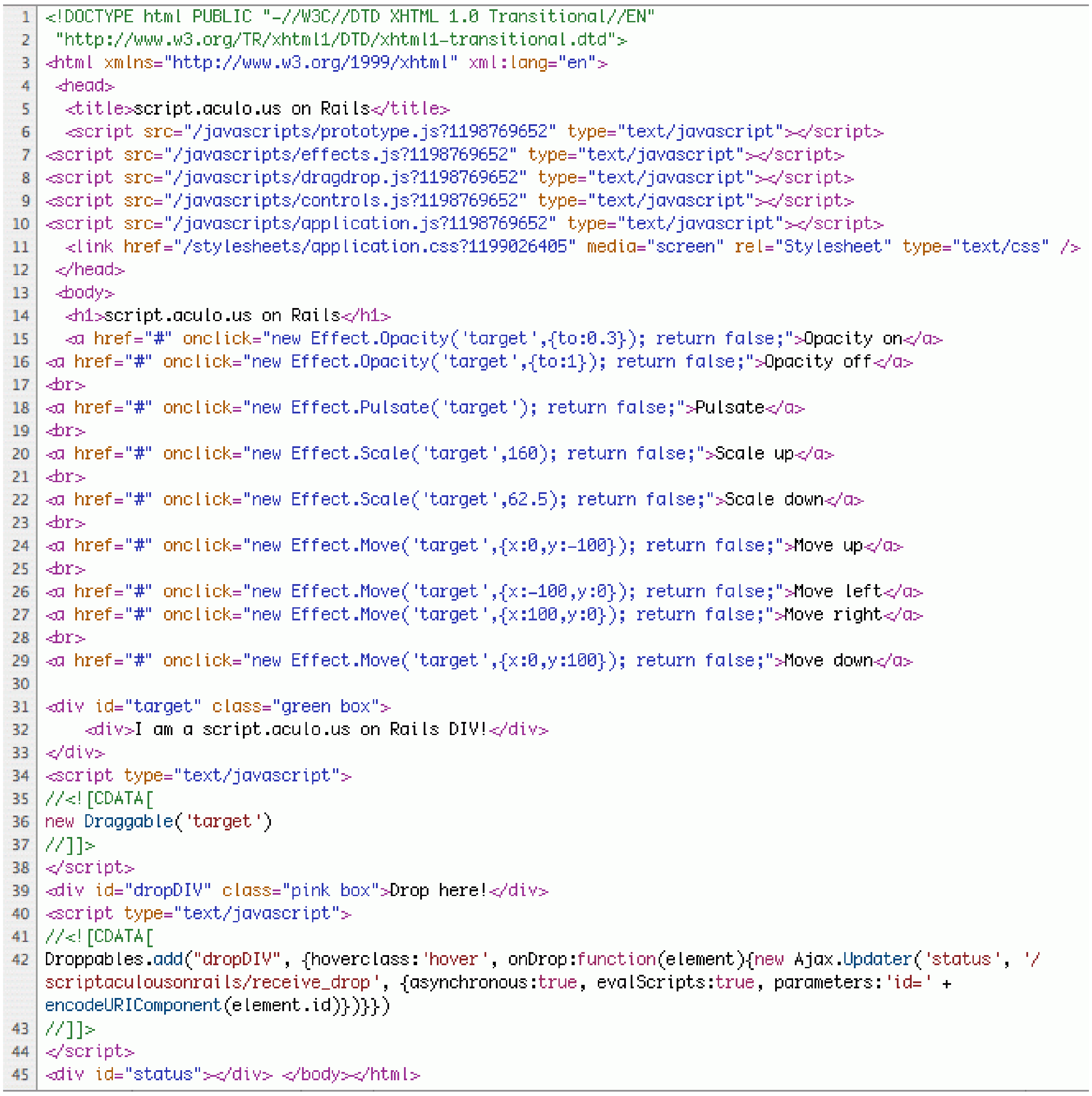
script.aculo.us on Rails

Opacity on Opacity off
Pulsate
Scale up
Scale down
Move up
Move left Move right
Move down

DROP HERE!

The element with the id target has been dropped

Figure 35: Final State of scriptaculousonrails



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <title>script.aculo.us on Rails</title>
  <script src="/javascripts/prototype.js?1198769652" type="text/javascript"></script>
<script src="/javascripts/effects.js?1198769652" type="text/javascript"></script>
<script src="/javascripts/dragdrop.js?1198769652" type="text/javascript"></script>
<script src="/javascripts/controls.js?1198769652" type="text/javascript"></script>
<script src="/javascripts/application.js?1198769652" type="text/javascript"></script>
  <link href="/stylesheets/application.css?1199026405" media="screen" rel="Stylesheet" type="text/css" />
 </head>
 <body>
  <h1>script.aculo.us on Rails</h1>
  <a href="#" onclick="new Effect.Opacity('target',{to:0.3}); return false;">Opacity on</a>
<a href="#" onclick="new Effect.Opacity('target',{to:1}); return false;">Opacity off</a>
<br>
<a href="#" onclick="new Effect.Pulsate('target'); return false;">Pulsate</a>
<br>
<a href="#" onclick="new Effect.Scale('target',160); return false;">Scale up</a>
<br>
<a href="#" onclick="new Effect.Scale('target',62.5); return false;">Scale down</a>
<br>
<a href="#" onclick="new Effect.Move('target',{x:0,y:-100}); return false;">Move up</a>
<br>
<a href="#" onclick="new Effect.Move('target',{x:-100,y:0}); return false;">Move left</a>
<a href="#" onclick="new Effect.Move('target',{x:100,y:0}); return false;">Move right</a>
<br>
<a href="#" onclick="new Effect.Move('target',{x:0,y:100}); return false;">Move down</a>

<div id="target" class="green box">
    <div>I am a script.aculo.us on Rails DIV!</div>
</div>
<script type="text/javascript">
//<![CDATA[
new Draggable('target')
//]]>
</script>
<div id="dropDIV" class="pink box">Drop here!</div>
<script type="text/javascript">
//<![CDATA[
Droppables.add("dropDIV", {hoverclass:'hover', onDrop:function(element){new Ajax.Updater('status', '/scriptaculousonrails/receive_drop', {asynchronous:true, evalScripts:true, parameters:'id=' + encodeURIComponent(element.id)})}})
//]]>
</script>
<div id="status"></div> </body></html>
```

Figure 36: HTML output of scriptaculousonrails

5 Apache MyFaces

Apache MyFaces is a project of the Apache Software Foundation and an implementation of the *JavaServer Faces* standard and is licensed under the Apache Software License. The JavaServer Faces standard has been developed since 2001. The Austrians Manfred Geiler and Thomas Spiegl started developing *MyFaces* which became an *Apache Software Foundation* project in 2004. MyFaces is the first free implementation of the JavaServer Faces standard and has some advantages compared to other implementations like many additional components that can be used by developers very fast and easy [MaScMü07].

Another major advantage is that the MyFaces Framework can be used like every other JavaServer Faces implementation, so it is not necessary to spend much time to learn MyFaces if a developer learned JavaServer Faces already. MyFaces comes up with some additional functions that can be divided into the following areas [MaScMü07]:

- Functions for simple reuse of site definitions (*Tiles* and *Portlet* support)
- Improved reuse of existing JavaScript and CSS Libraries
- A more comprehensive component library, especially for the user interface
- Components that allow the use of AJAX
- A multiplicity of additional settings that can improve the performance of JavaServer Faces applications dramatically

Apache MyFaces is compatible to most containers but here are the most important ones that are definitely supported [MaScMü07]:

- Tomcat 4.X, 5.X and 6.X
- JBoss 3.2.X and 4.0.X
- BEA Weblogic 8.1
- Websphere 5.1.2

For the installation the container Tomcat 5.5.25 was used as it is recommended on the Apache MyFaces site but the installation will be explained in the installation section. First let's have a look on the JavaServer Faces standard on which Apache MyFaces is based.

Apache MyFaces is divided into several sub-projects that are listed below [MyFa07a]:

- **MyFaces API** and **MyFaces Impl Modules**: JavaServer Faces implementation
- **MyFaces Tomahawk**, **Trinidad** and **Tobago**: Component Libraries that contain user interface widgets
- **MyFaces Sandbox**: Subproject that tests ground for new developments in the Tomahawk project
- **MyFaces Orchestra**: Extension packages for JavaServer Faces
- **MyFaces Portlet Bridge**: Integration module for the Portlet standard

The focus of this paper will be on the Tomahawk project because it provides a component called “Dojo Initializer” that allows to integrate the *Dojo Toolkit* which is a open-source JavaScript Framework [MyFa07c].

5.1 JavaServer Faces

JavaServer Faces, also known as JSF, is a Java based web application framework standard that follows as mentioned in an earlier chapter the MVC pattern [MyFa07b]. It enables developers to create web interfaces that are based on Java web-applications. It was for the first time officially released by Sun Microsystems in 2004. The goal was to create the best java-based web-solution as there were some others with known weaknesses [BaHüRö07] . The first official release was held on the Java Specification Request 127 (JSR 127) and defines JavaServer Faces versions 1.0 and 1.1, latter released on May 27th 2004 [Java07a]. Some intentions were to make things like internationalization, input validation or site navigation easier. The second major release with the version JSR 252 was published on May 11th 2006, two years after the last release. Currently, JSR 314 that is going to define JavaServer Faces 2.0 is in progress [Java07b].

To understand the basic concepts of the JavaServer Faces a short introduction into the specifics and differences to other frameworks is now given.

As mentioned in an earlier chapter the JavaServer Faces specification implements the Model-View-Controller pattern. The JavaServer Faces, from now on called JSF, differs from most other web-application frameworks as it defines a *User-Interface component tree* for every executed page. As a request is sent there is a defined life-cycle of this request that takes responsibility for duties like validation or calling the business-logic in the controller [BaHüRö07].

A view consists of hierarchical ordered user interface components like simple input boxes, buttons, tables or menus. The structure of the tree defines the design of the presented page and as it is hierarchical, for example a table can hold buttons or anything else as well. The root of every tree is a instance of the type *UIViewRoot*. Nearly every attribute of a component can be associated with *Value Bindings* that save the actual value to a specific destination. That offers the possibility to save for example the input of a text-box into the model or request it from the model. Referred to the Value Bindings, it is also possible to create *Method Bindings* that call a method when the action that defines the binding is activated, like a button or a hyperlink [BaHüRö07].

As the basics of the site-definition are explained let's have a look on the earlier mentioned life-cycle of a JSF request. As a Server receives a JSF request it is executed in six steps that follow the *Request Processing Life-cycle*, shortened RPL that are [BaHüRö07]:

- Restore View
- Apply Request Values
- Process Validations
- Update Model Values
- Invoke Application

- Render Response

Regularly all steps are executed unless there are errors that can cause a shortening of the execution like an error in the validation process. The first step is called *Restore View* and it rebuilds the saved component tree of a user or creates a new one if no one exists. The following *Apply Request Values* fills the component tree with the values that came with the request in the HTML parameters, Cookies or the HTTP header. Before data can be written in a model the data has to be validated and converted into the target format. This happens in step three, the *Process Validations,* which creates messages when errors in the validation happen and marks the values as not valid. If a validation throws an error the next steps will be skipped and the last one called *Render Response* will be executed. A simple example for an error in the validation could be a violation of the minimum length property of an input field. But as the conversion and the validation succeed the *Update Model Values* step is executed. It writes the valid data into to model referring to the Value Bindings that have been set. When the *Invoke Application* phase is started all the data has been written and the methods of the business-logic in the controller are called that are defined by the Method Bindings. Once completed the application creates a so-called “Outcome” that represents a value that defines the next view that should be created defined by a navigation path. If an error occurres in the *Invoke Application* step the actual view that created the request will be presented including error messages. A Renderer defines how a component looks on the client and how an application codes and decodes its parameters. Figure 37 shows the whole life-cycle in a graphical way [BaHüRö07].

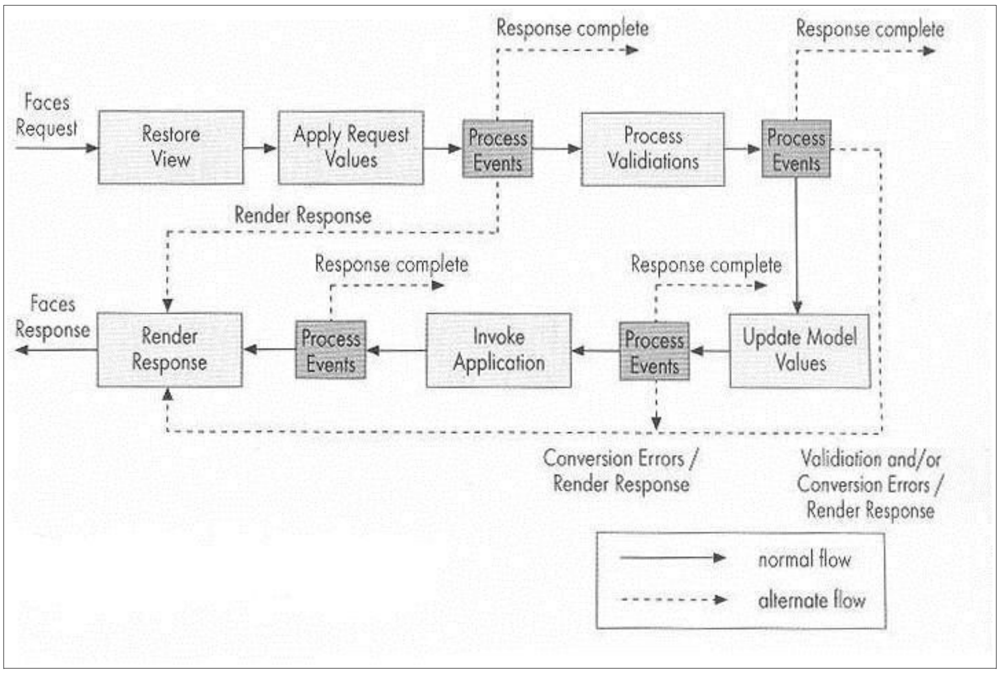


Figure 37: Life-cycle of a JavaServer Faces request [BaHüRö07, page 118]

5.2 AJAX-Components

JavaServer Faces was created to develop classic web-applications like they have been introduced at the beginning. JSF tries to keep the whole business logic at the server but AJAX applications try to bring some of the logic to the client, so there are several problems with the combination of JSF and AJAX but some approaches have been developed on Apache MyFaces which are now introduced [MaScMü07].

There are two major possibilities to implement AJAX functionalities into MyFaces applications. One way is to implement ready-to-use components that ship with MyFaces and the other one is that a developer has to code the JavaScript functionalities himself.

Apache MyFaces offers some AJAX components in its Sandbox Project that are still in development, so the API can be changed anytime. The components that are provided at the moment are [MaScMü07]:

- **AJAX Suggest**: Automatically completion of user input through a list of suggestions (Like Google Suggest uses it)
- **Auto Update Data Table**: Content of a tagged component are refreshed

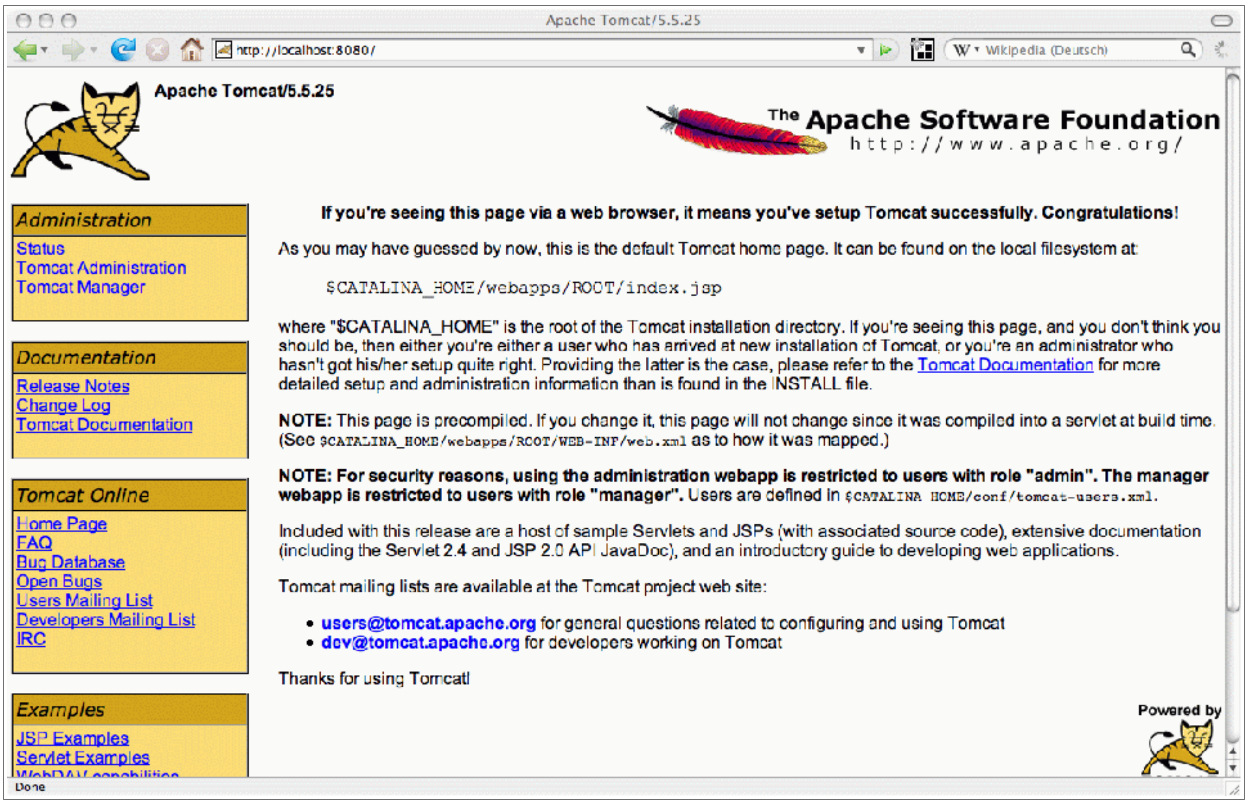
automatically

- **AJAX Form Components**: Automatic update of the data model when a user is editing a form
- **Partial Page Rendering**: A page is divided into a static and and reloading parts

For the use of AJAX components it is crucial to use a JavaScript Framework. Prototype and script.aculo.us are two examples, but both of them are not useable with MyFaces, because Prototype for example creates class names like `Event`. This name could be used by other frameworks or by self-written JavaScript code too, so incompatibilities can occur very fast. The only framework that is able to deal with this problem and provides the same grade of distribution and functionality is the *Toolkit* Dojo [MaScMü07].

5.3 Installation

The installation consists of three major parts that are Java, Apache Tomcat and Apache MyFaces with the MyFaces Tomahawk Support. For the examples Java in the version 5 for Mac OS X was used. As Java is installed on most systems there is no information provided how to install Java in this paper. The second step is to install Apache Tomcat which can be downloaded from the official Apache site under http://tomcat.apache.org/. Actually the latest release is 6.0.14 but for the examples version 5.5.25 was used because it is recommended by MyFaces at the moment. Once Tomcat is installed the server can be started. The `/bin` folder of the installation contains a `startup.sh` as well as a `shutdown.sh` that start and stop the server. If the server is started and the standard configuration is used, the server is availiable under http://localhost:8080/. If this site is called, the Tomcat Testpage appears that can be found in Figure 38.



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.jsp
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: This page is precompiled. If you change it, this page will not change since it was compiled into a servlet at build time. (See `$CATALINA_HOME/webapps/ROOT/WEB-INF/web.xml` as to how it was mapped.)

NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation (including the Servlet 2.4 and JSP 2.0 API JavaDoc), and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

- **users@tomcat.apache.org** for general questions related to configuring and using Tomcat
- **dev@tomcat.apache.org** for developers working on Tomcat

Thanks for using Tomcat!

Figure 38: Test-page of the Tomcat servlet container

Finally the MyFaces packages for MyFaces and the Tomahawk packages can be downloaded from http://myfaces.apache.org/download.html and copied to Tomcats web-application folder.

5.4 The Toolkit Dojo

The Open-Source Toolkit Dojo is very adequate for the use with MyFaces because it is divided into packages to avoid collisions with the same names. Furthermore the library is separated into functional parts with the advantage that only the used modules have to be imported by the developer and not the whole library. This results in the fact that the web-browser does not have to load the whole packages, but just the necessary ones. Dojos functionalities can be separated into the following categories [MaScMü07]:

- **dojo.event**: Event-handling by registering the element and the type of event
- **dojo.validate**: Module to execute client-side validation
- **dojo.dnd**: Drag-and-Drop functionalities

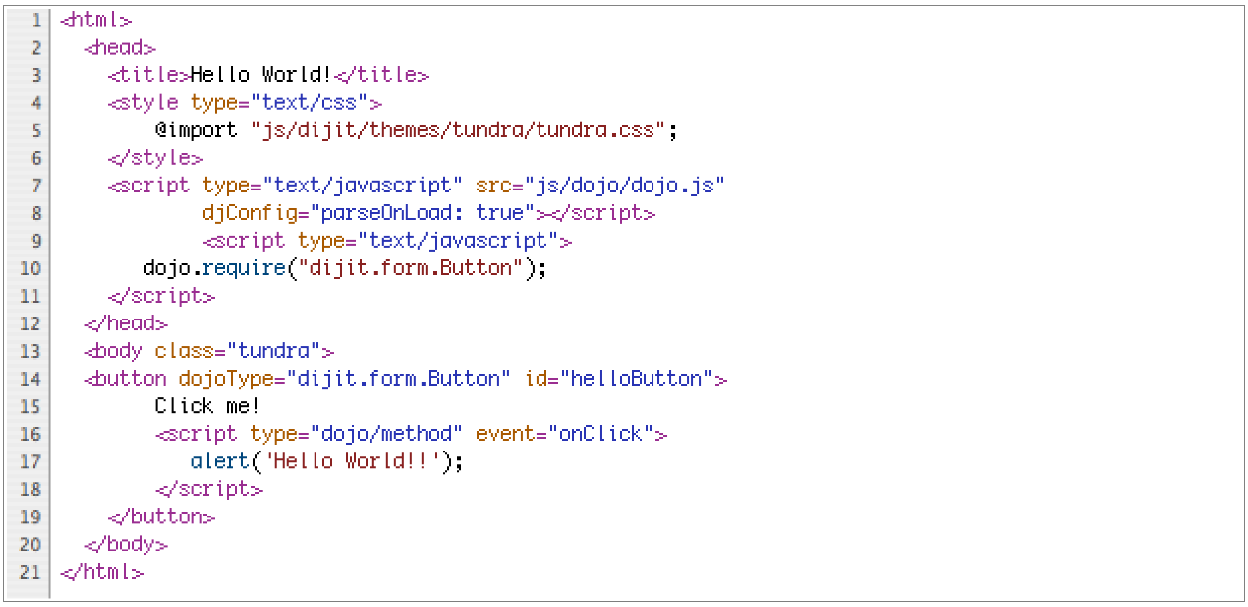
- **dojo.animation**: Changing the attributes of elements dynamically
- **dojo.io**: Access to platform-independent resources e.g. local files or AJAX requests
- **dojo.widget**: Client-Side control elements

Besides the Dojo Core project two more projects exist. The first one is called *dijit* and is a set of internationalized widgets and design themes [Dojo07c]. The second one is called *dojox* and provides some features like a drawing API, Offline functions or a charting functio [Dojo07d].

Dojo supports all major web-browsers, in detail there is support for Microsoft's Internet Explorer 6+, Mozilla Firefox 1.5+, Konqueror 3.5+ and the latest version of Apples Safari, 3.0 [Dojo07a].

5.4.1 Code example of Dojo

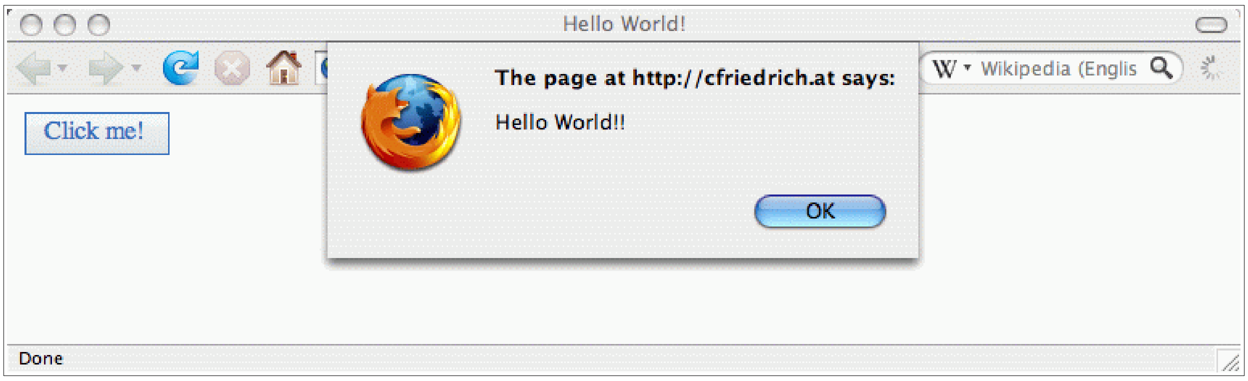
To provide a small code example for the Toolkit Dojo too, a simple “Hello World!” application is created as well. Figure 39 shows the code for the application that was created with Dojo. Line 5 defines a stylesheet that is loaded from the Toolkit. The next important line is number 7 which loads the `dojo.js` that contains all the further information of the Toolkit. Line 14 loads a button from the Dojo Library while line 16 through 18 assign the event when the button is clicked [Dojo07b].



```
<html>
  <head>
    <title>Hello World!</title>
    <style type="text/css">
        @import "js/dijit/themes/tundra/tundra.css";
    </style>
    <script type="text/javascript" src="js/dojo/dojo.js"
            djConfig="parseOnLoad: true"></script>
            <script type="text/javascript">
       dojo.require("dijit.form.Button");
    </script>
  </head>
  <body class="tundra">
  <button dojoType="dijit.form.Button" id="helloButton">
        Click me!
        <script type="dojo/method" event="onClick">
           alert('Hello World!!');
        </script>
    </button>
  </body>
</html>
```

Figure 39: Code for the "Hello World!" application with Dojo

Figure 40 shows the output in the web-browser in the state when the button is already clicked. The special thing is most likely the design of the button which was imported in the code and comes from the Toolkit.



Hello World!

The page at http://cfriedrich.at says:

Hello World!!

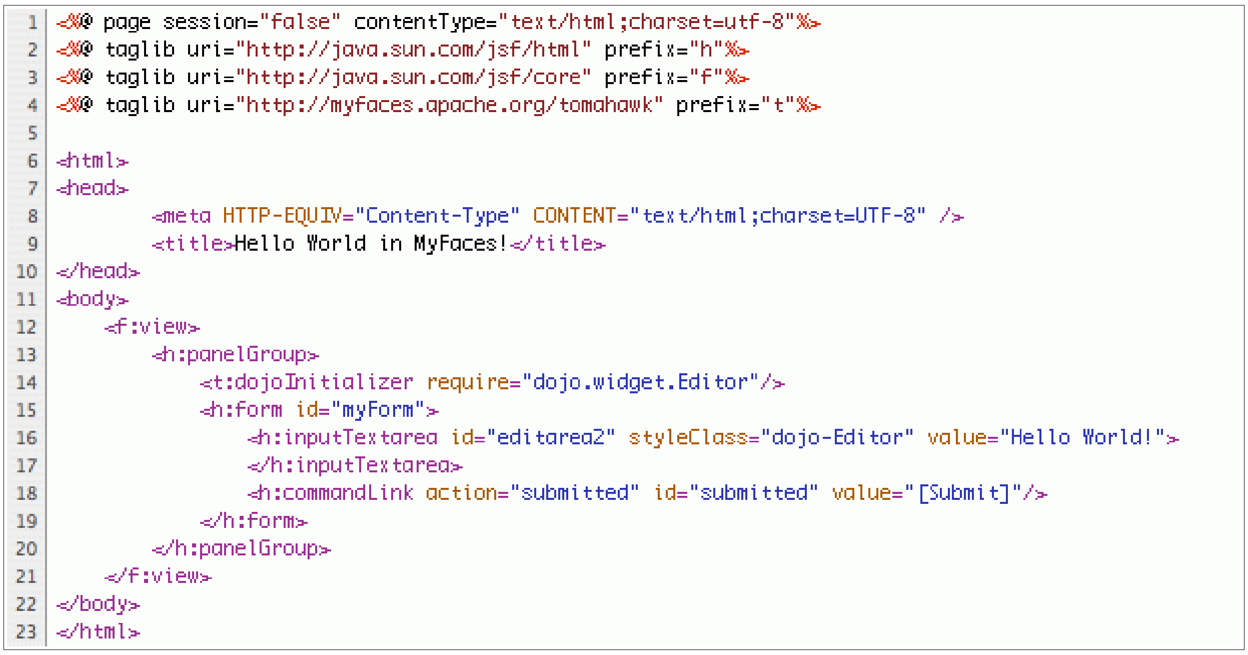
Figure 40: Screenshot of the Hello World application of the Dojo Toolkit

5.5 Code Examples in Apache MyFaces

Apache MyFaces ships with a lot of examples about MyFaces itself as well as the Tomahawk package. Furthermore there are sample applications of the Sandbox project available online. As this paper wants to see how JavaScript elements are implemented in Apache MyFaces, some of these examples are going to be introduced and a look at the source code will be given. To implement the examples oneself would lead to a very high amount of time setting the right parameters in the configuration files like the web.xml and the faces-config.xml and to write the java classes that are necessary to let these applications work.

Dojo Integration

The first code example is a MyFaces Hello World example that makes use of both the packages JavaServer Faces Core and MyFaces Tomahawk and implements a Widget of the Toolkit Dojo to show how simple it is to implement Dojo widgets. Figure 41 shows the code part of the JavaServer Faces file.

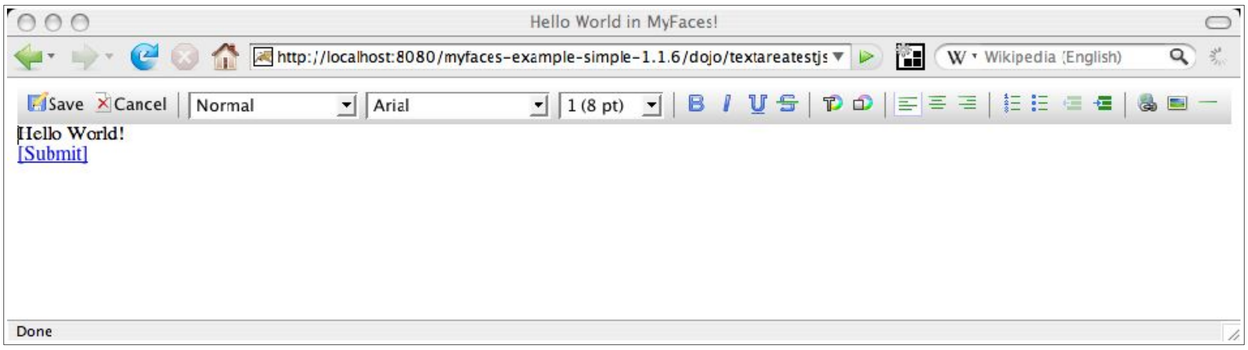


```
 1 <%@ page session="false" contentType="text/html;charset=utf-8"%>
 2 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
 3 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
 4 <%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
 5
 6 <html>
 7 <head>
 8         <meta HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=UTF-8" />
 9         <title>Hello World in MyFaces!</title>
10 </head>
11 <body>
12     <f:view>
13         <h:panelGroup>
14             <t:dojoInitializer require="dojo.widget.Editor"/>
15             <h:form id="myForm">
16                 <h:inputTextarea id="editarea2" styleClass="dojo-Editor" value="Hello World!">
17                 </h:inputTextarea>
18                 <h:commandLink action="submitted" id="submitted" value="[Submit]"/>
19             </h:form>
20         </h:panelGroup>
21     </f:view>
22 </body>
23 </html>
```

Figure 41: Code of the Hello World example in MyFaces

The file itself is written into a JavaServer Pages file as it is recommended by the JSF specification [MaScMü07]. Line 2 through 4 show the definition of the packages that are used and a prefix is assigned that is used later to define the right package. The next important code part is in line 12 where a `<f:view>` tag that is also specified by the JSF specification and has to enclose all the JSF tags. The tag `<h:panelGroup>` defines a HTML tag like the `h:` prefix shows and creates a `span` element. As mentioned in the MyFaces introduction, line 14 defines the `<t:dojoInitializer/>` tag which is defines by the Tomahawk package and is able to import a Dojo component. In this example a Text-Editor is loaded while line 15 through 19 define a inputTextarea which is a simple input box to enter text which can be designed with the Text-Editor component that was inserted before.

Finally, let's have a look at the output of the file in a web-browser that is shown in Figure 42. At the top of the page the inserted text-editor bar is shown which controls the input field that is right below it and has the default value “Hello World!”.



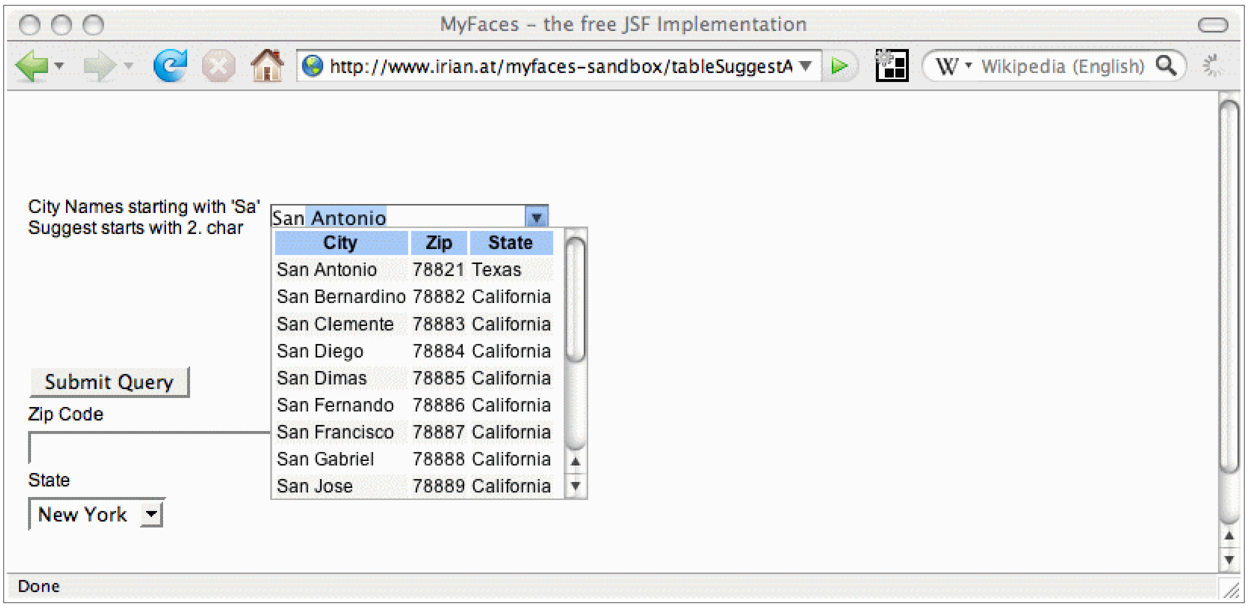
Hello World!
[Submit]

Figure 42: Screenshot of the Hello World application in MyFaces

The submit button could cause an AJAX call that saves the current input of the input field but this functionality is not implemented in this example.

Sandbox-AJAX example

As mentioned earlier, the MyFaces project Sandbox contains several components to implement AJAX functionalities. Several Sandbox examples are available on http://www.irian.at/myfaces-sandbox/tableSuggestAjax.jsf. This example contains just the view component and the controller is left out. In this case it is useful to provide the screenshot of the application first and have a look at the JavaServer Faces file later. Figure 43 shows the screenshot of the application. After two characters are typed, a request is sent to request suggestions for the input. The Sandbox component `TableSuggestAjax` was used to request the informations from the server. Once a item is selected the two text-boxes at the bottom of the page are filled out with the information that was retrieved by the server.


MyFaces - the free JSF Implementation
http://www.irian.at/myfaces-sandbox/tableSuggestA
Wikipedia (English)
Submit Query
Done

City Names starting with 'Sa'
Suggest starts with 2. char

San Antonio

City	Zip	State
San Antonio	78821	Texas
San Bernardino	78882	California
San Clemente	78883	California
San Diego	78884	California
San Dimas	78885	California
San Fernando	78886	California
San Francisco	78887	California
San Gabriel	78888	California
San Jose	78889	California

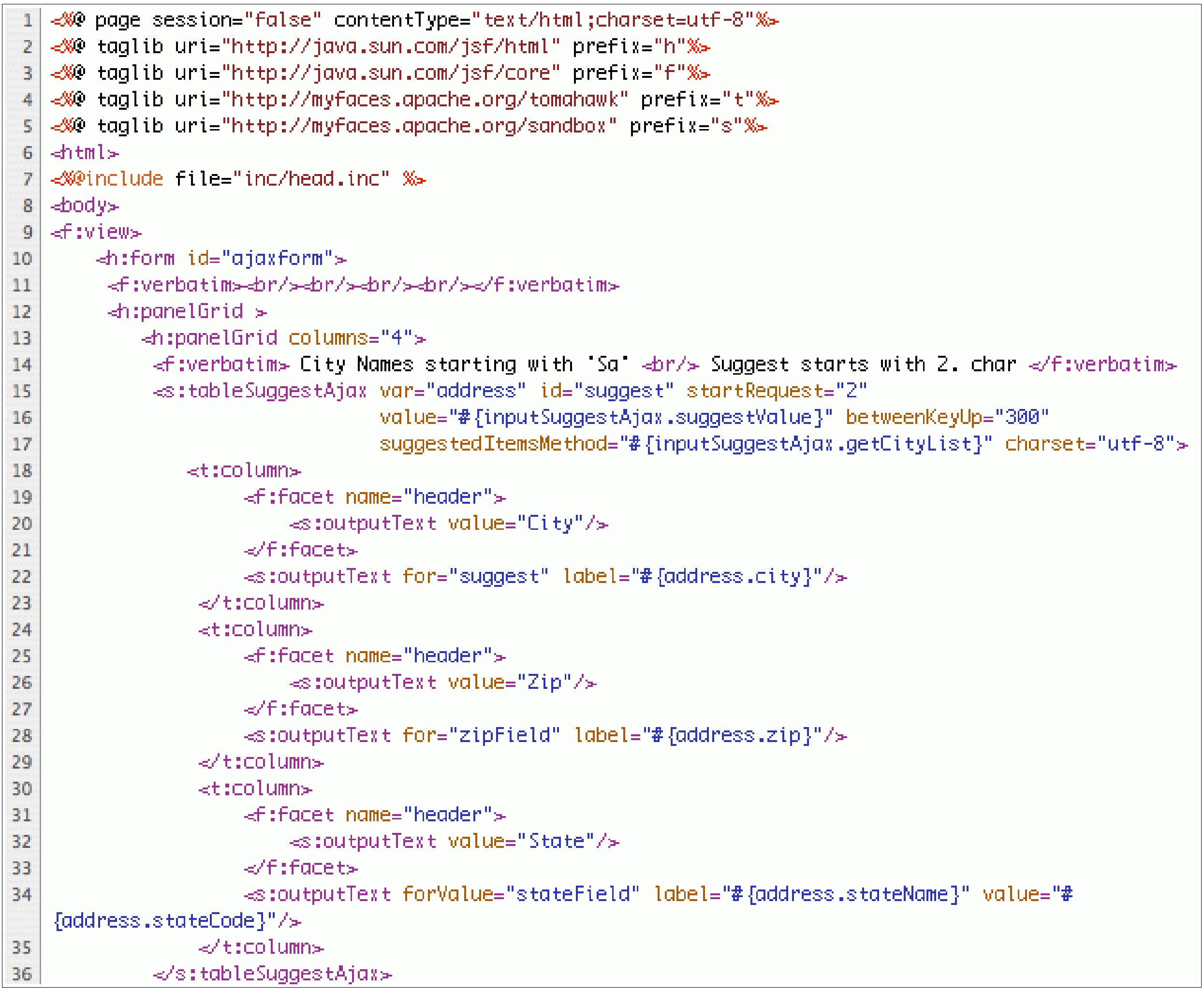
Zip Code

State

New York

Figure 43: Screenshot of the TableSuggestAJAX example

Figure 44 shows an abstract of the code, the code at the end defines the text and combo boxes at the bottom are left out to keep the code compact. Line 2 through 4 again define the familiar Core, HTML and Tomahawk components. New is Line 5 which defines the Sandbox package with an own prefix. Line 9 again defines the `<f:view>` that was used in the last example as well. The `<f:verbatim>` tag in line 11 enables the possibility to include raw HTML code into the JSF file as it is done in line 14 as well. Lines 15 through 18 are the crucial code-part of this example as the `<s:tableSuggestAjax>` tag defines the use of the Sandbox component with the same name. The parameter `startRequest` with the value of 2 defines that the AJAX request starts first when two characters are typed. `Value` defines the inputted value in the text box and calls a Java bean in its parameter. That is a Java class on the server that processes the request. That is called `inputSuggestAjax` and the method `suggestValue` that hands the inputted value to the server. `BetweenKeyUp` defines a period in milliseconds in which no AJAX request is sent if another character has been entered. Finally `suggestedItemsMethod` requests the suggestions from the server. Line 18 through 35 create the table that is shown below the input box with its three columns that are the City name, the ZIP code and the State and request the data for the suggestions with the notation `address.city`, `address.zip` and `address.stateName`. The `for` parameters in line 22, 28 and 34 set the values into other fields. For example `for="suggest"` enters the suggestion into the input text field so that the first suggestion is suggested completely in the input box. The `zipField` and `stateField` values that are used in line 26 and 34 put the values into the text-boxes that are left out in this code example but can be seen in Figure 43.



```
 1 <%@ page session="false" contentType="text/html;charset=utf-8"%>
 2 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
 3 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
 4 <%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
 5 <%@ taglib uri="http://myfaces.apache.org/sandbox" prefix="s"%>
 6 <html>
 7 <%@include file="inc/head.inc" %>
 8 <body>
 9 <f:view>
10     <h:form id="ajaxform">
11      <f:verbatim><br/><br/><br/><br/></f:verbatim>
12      <h:panelGrid >
13         <h:panelGrid columns="4">
14          <f:verbatim> City Names starting with 'Sa' <br/> Suggest starts with 2. char </f:verbatim>
15          <s:tableSuggestAjax var="address" id="suggest" startRequest="2"
16                              value="#{inputSuggestAjax.suggestValue}" betweenKeyUp="300"
17                              suggestedItemsMethod="#{inputSuggestAjax.getCityList}" charset="utf-8">
18             <t:column>
19                  <f:facet name="header">
20                      <s:outputText value="City"/>
21                  </f:facet>
22                  <s:outputText for="suggest" label="#{address.city}"/>
23              </t:column>
24              <t:column>
25                  <f:facet name="header">
26                      <s:outputText value="Zip"/>
27                  </f:facet>
28                  <s:outputText for="zipField" label="#{address.zip}"/>
29              </t:column>
30              <t:column>
31                  <f:facet name="header">
32                      <s:outputText value="State"/>
33                  </f:facet>
34                  <s:outputText forValue="stateField" label="#{address.stateName}" value="#
   {address.stateCode}"/>
35              </t:column>
36          </s:tableSuggestAjax>
```

Figure 44: Code of the TableSuggestAjax example

6 Comparison

As both web-application frameworks have been introduced and the basic concepts have been explained it is time to point out advantages and disadvantages of the two frameworks.

6.1 Advantages of Apache MyFaces / Ruby on Rails

Let's start with some advantages of both frameworks. One advantage that both frameworks share is that they are open-source and are so free to use, even for commercial use.

Apache MyFaces

Apache MyFaces comes up with a very powerful JavaScript Toolkit namely Dojo. Dojo brings the major advantage that it does not have to transfer the whole Library to the web-browser that requests functions of it. Instead just the packages that are explicitly requested are going to be transferred.

Furthermore the component library of MyFaces brings some very convenient AJAX components that are still in the Sandbox project. That means that the API can be changed anytime and is so still in development. To mention some examples those components bring support for functions like a auto-complete function for user input fields or automatic updating data-tables.

Ruby on Rails

The major advantage of Ruby on Rails is that the JavaScript Frameworks Prototype and script.aculo.us that have been introduced earlier are developed in concert with Ruby on Rails. This brings the advantages that both are adjusted to each other.

Another advantage of Ruby on Rails are the Rails Helpers that are very easy to use and avoid the writing of JavaScript Code. They also avoid the writing of redundant code parts in several parts of a web-page. Furthermore new versions of the JavaScript Libraries do not have to be considered as new versions of Ruby on Rails will provide revised Helpers.

Last but not least Ruby on Rails brings the advantage that it is actually designed for dynamic and interactive web-applications and so no new concepts are needed to implement AJAX components.

6.2 Disadvantages of Apache MyFaces / Ruby on Rails

After having a look on the advantages in Ruby on Rails and Apache MyFaces its disadvantages are discussed now.

Apache MyFaces

Apache MyFaces has a drawback that results from the fact that the JavaServer Faces specification was developed to create classic web-applications. Server side components are provided to make the development of complex applications easier. The concept is to keep the business logic on the server side. But the AJAX idea is to bring some business logic to the client, so solutions are needed to bring both concepts to work. Furthermore the Dojo Toolkit is the only JavaScript Library that can be used with Apache MyFaces, this is not necessarily a drawback, because it is one of the most famous frameworks, but still it is a constraint to be bound to a special Library.

Another drawback is that the mentioned component library of MyFaces is still in the Sandbox project so the use of these components in critical applications could become kind of risky.

Ruby on Rails

In the opinion of the author of this paper the major drawback of Ruby on Rails is that Rails does not have such a powerful programming language standing behind it as Apache MyFaces has, namely Java. With powerful not the language itself is meant, but the power of the Java- and Apache Community. Companies like Oracle, Borland and IBM created the JavaServer Faces specification so really big companies out of the IT sector will be supporting and providing input into the JavaServer Faces project.

From the viewpoint of AJAX it is difficult to find severe drawbacks. After doing a lot of investigation regarding the topic AJAX and testing some basic examples Ruby on Rails can be seen as a best practice example how to implement AJAX components in a web-application. From the viewpoint of enterprise applications the author personally is not able to judge the capabilities of Ruby on Rails.

7 Conclusion

There are two major conclusions that should be mentioned. The first one deals with AJAX in general and the second one with the two web-application frameworks.

The first conclusion of this paper is that AJAX brings a lot of advantages into the web-development but also has some issues that have to be considered. The introduction chapter showed that web-developers that want to implement AJAX components have to be aware of usability problems as well as new security risks of their applications. Even though AJAX is a very fancy term these days, especially for critical applications the use of AJAX components should be thought of twice.

The second conclusion of this paper is that both Apache MyFaces and Ruby on Rails provide the capabilities to implement AJAX components into web-applications. While Ruby on Rails was basically created to implement highly interactive web-applications the JavaServer Faces specification bases on the classic web-model. Despite this fact the JavaServer Faces respectively Apache MyFaces provide several ways to implement AJAX components. But in the end both frameworks have their strengths and weaknesses that have to be traded off to find the appropriate web-application framework for a specific requirement.

8 References

[Augu07] August, David; Ruby on Rails Weblog; http://weblog.rubyonrails.org/2006/8/7/ruby-on-rails-will-ship-with-os-x-10-5-leopard; retrieved on 2007-12-27

[BaHüRö07] Bartetzko, Detlef; Hülsebus, Arvid; Röwekamp, Lars; Im Angesicht des Web – Grundlagen der Java Server Faces; iX Special; Volume 2007; Issue 1; page 111-116

[Bosw07] Bosworth, Alex; Ajax Mistakes; http://alexbosworth.backpackit.com/pub/67688; retrieved on 2007-11-18

[Dojo07a] The Dojo Toolkit; http://dojotoolkit.org/support/faq/what-browsers-does-dojo-support; retrieved on 2007-12-27

[Dojo07b] The Dojo Toolkit; http://dojotoolkit.org/book/dojo-book-0-9/hello-world-tutorial; retrieved on 2007-12-27

[Dojo07c] The Dojo Toolkit; http://dojotoolkit.org/projects/dijit; retrieved on 2007-12-28

[Dojo07d] The Dojo Toolkit; http://dojotoolkit.org/projects/dojox; retrieved on 2007-12-28

[Garr05] Garrett, Jesse James; Ajax: A New Approach to Web Applications; http://www.adaptivepath.com/ideas/essays/archives/000385.php; retrieved on 2007-11-18

[Gibs06] Gibson, Becky; AJAX Accessibility Overview; April 2006; IBM; http://www-03.ibm.com/able/resources/ajaxaccessibility.html#issue; retrieved on 2007-11-18

[Goog07] Google; Google Suggest FAQ; http://labs.google.com/suggestfaq.html; retrieved on 2007-12-18

[Hayr06] Hayre, Jaswinder; Ajax Security Basics; http://www.securityfocus.com/infocus/1868; 2006-06-19; retrieved on 2007-12-08

[Hein07] David Heinemeier Hansson; http://www.loudthinking.com/about.html; retrieved on 2007-12-27

[Inma06] Inman, Shaun; Responsible Asynchronous Scripting;

http://www.thinkvitamin.com/features/ajax/responsible-asynchronous-scripting; retrieved on 2007-12-01

[Inst07] InstantRails Wiki; http://instantrails.rubyforge.org/wiki/wiki.pl; retrieved on 2007-12-27

[Java07a] The Java Community Process Program, JSR 127; http://www.jcp.org/en/jsr/detail?id=127; retrieved on 2007-12-27

[Java07b] The Java Community Process Program, JSR 314; http://www.jcp.org/en/jsr/detail?id=314; retrieved on 2007-12-27

[Loco07] Locomotive; http://locomotive.raaum.org/; retrieved on 2007-12-27

[Lubk07] Lubkowitz, Mark; Webseiten programmieren und gestalten; 3rd Edition; Galileo Press Bonn; 2007

[Mark07] Network Solutions LLC; Market share for browsers, operating systems and search engines; http://marketshare.hitslink.com/report.aspx?qprid=3; retrieved 2007-12-18

[MaScMü07] Marinschek, Martin; Schnabl, Andrea; Müllan, Gerald; JSF@Work; 1st Edition; dpunkt.verlag; 2007

[McEv05] McEvoy, Chris; Why Ajax Sucks (Most of the Time); http://www.usabilityviews.com/ajaxsucks.html; retrieved on 2007-11-18

[Micr07] Microsoft Developer Network; Model-View-Controller; http://msdn2.microsoft.com/en-us/library/ms978748.aspx; retrieved on 2007-12-08

[Mozi07] Mozilla Foundation; About Mozilla; http://www.mozilla.com/en-US/about/, retrieved on 2007-12-18

[MyFa07a] Apache MyFaces; http://myfaces.apache.org/index.html; retrieved on 2007-12-27

[MyFa07b] Apache MyFaces; http://wiki.apache.org/myfaces/; retrieved on 2007-12-27

[MyFa07c] Apache MyFaces Tomahawk; http://myfaces.apache.org/tomahawk/dojoInitializer.html; retrieved on 2007-12-27

[PaFe06] Di Paola, Stefano; Fedon, Giorgio; Subverting Ajax, 23rd CCC Conference; December 2006

[Raym07] Raymond, Scott; Ajax on Rails; O'Reilly; 2007

[Ritc07] Ritchie, Paul; The security risks of AJAX/web 2.0 applications; Network

Security, Volume 2007, Issue 3, March 2007, Pages 4-8

[Ruby07] Ruby on Rails; http://www.rubyonrails.org/; retrived on 2007-12-27

[Schw06] Schwarz, Michael; AJAX and the Search Engine Problems; http://weblogs.asp.net/mschwarz/archive/2005/08/06/421761.aspx; retrieved on 2007-11-18

[SunM02] Sun Microsystems; Designing Enterprise Applications with the J2EE Platform, Second Edition; http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html#1105855, retrieved on 2007-12-08

[Weba07] Unknown Author, Accessibility of AJAX Applications; http://webaim.org/techniques/ajax/#whynotajax; retrieved on 2007-11-18

[Wire07] Wireshark; Wireshark:About; http://www.wireshark.org/about.html; retrieved on 2007-12-18

[Zuck07] Zucker, Daniel; What Does Ajax Mean for You?; interactions, September and October 2007; page 10-12