Tiago Leandro de Castro Ribeiro **Web 2.0: User Interaction in Web Applications**

UMinho|2008

Universidade do Minho
Escola de Engenharia

Tiago Leandro de Castro Ribeiro

Web 2.0: User Interaction in Web Applications

Novembro de 2008

Universidade do Minho
Escola de Engenharia

Tiago Leandro de Castro Ribeiro

Web 2.0: User Interaction in Web Applications

Tese de Mestrado em Informática

Trabalho efectuado sob a orientação do
Professor Doutor António Nestor Ribeiro

Novembro de 2008

Tiago Leandro de Castro Ribeiro

"It's no longer the one-way Web"

Dorion Carrol, Technorati

Abstract

The main characteristic of the so called Web 2.0 lies with the improved user interaction. Therefore, Web 2.1 is the logical next step: bringing real-time user interaction to Web Applications.

Web Applications have come a long way in the last few years. This evolution brought with it new challenges, as application developers explore better ways to improve the user experience and bring it to the same level of desktop applications. Techniques such as AJAX are already quite documented and becoming well established, with major players like Google, Yahoo! and Microsoft, already taking advantage of it. On the other hand, Comet - the most interesting Reverse AJAX implementation - is a somewhat new concept undergoing lots of changes. As in other emergent Web technologies, achieving proofs of technical merits like scalability and network performance is still the main focus, not simplicity or ease of integration.

In this thesis, AJAX and Comet are brought together into a coherent and elegant architecture, designed in direct response to the challenges presented by the new paradigms of real-time user interaction in Web Applications. This architecture describes a clear and simple path for implementing these technologies effectively. As a showcase for this architecture, the author developed `simX` , a Persistent Browser-Based Game (PBBG). Such a system is by its nature a simulation, and as such attempts to leverage the well known effect of suspension of disbelief in order to merge people into a virtual world. For that purpose, it must overcome the limitations imposed by the underlying Web technology. The problem faced in this setting is that people generally go on about their lives with the safe assumption that the world will not cease to function if they stop interacting with it. Events like sounds, smells and images will all reach their senses even if no special action is performed.

In order to achieve a remotely comparable experience in this web game, several systems were developed, delivering relevant information in real-time, without requiring any user interaction. It's interesting to conclude that a path to improve user interaction is essentially reducing the requirement for user interaction.

Resumo

A principal característica da dita Web 2.0 encontra-se nas melhorias a nível de interacção do utilizador. Assim sendo, Web 2.1 será o passo lógico seguinte: dotar as aplicações Web da capacidade de interacção com o utilizador, em tempo real.

As aplicações Web tiveram um extenso percurso nos últimos anos. Esta evolução trouxe consigo novos desafios, à medida que os desenvolvedores de aplicações procuram novas formas de melhorar a experiência dos utilizadores e trazê-la para o mesmo nível que as aplicações de desktop. Técnicas como AJAX já estão bem documentadas e cada vez mais estabelecidas, com os grandes players como Google, Yahoo! e Microsoft a tirarem partido desta. Por outro lado, Comet – a implementação mais interessante de AJAX Reverso – é um conceito novo que ainda está sofrer várias alterações. Tal como acontece com outras tecnologias Web emergentes, alcançar determinadas provas de mérito técnicas como escalabilidade e desempenho em rede ainda são o foco principal, e não a simplicidade ou a facilidade de integração.

Nesta tese, AJAX e Comet são juntados numa arquitectura coerente e elegante, desenhada como uma resposta directa aos desafios apresentados pelos novos paradigmas da interacção em tempo real em aplicação Web. Esta arquitectura descreve de forma simples e clara o caminho para efectivamente se implementar estas tecnologias. Como cenário de demonstração, o autor desenvolveu o `simX` , um Jogo Persistente Baseado em Browser (PBBG, do inglês Persistent Browser-Based Game). Tal sistema é, por sua natureza, uma simulação e, como tal, procura alcançar o conhecido efeito da suspensão da descrença de forma a mesclar pessoas num mundo virtual. Por este motivo, deve ultrapassar as limitações impostas pela tecnologia Web em que assenta. O problema desta configuração é que geralmente as pessoas continuam a sua vida com a assunção segura de que o mundo não deixa de funcionar se deixarem de interagir com ele. Eventos como sons, cheiros e imagens atingiram todos os seus sentidos mesmo que nenhuma acção especial seja executada.

Para atingir uma experiência remotamente comparável neste jogo Web, vários sistemas foram desenvolvidos, mostrando informação relevante em tempo real, sem requisitarem qualquer interacção do utilizador. É interessante concluir que o caminho para melhorar a interacção do utilizador é essencialmente conseguida por redução da interacção do mesmo.

Acknowledgements

To start with, I must say that I owe much of this work to the support of the people around me.

I'd like to thank my supervisor, Dr. Nestor Ribeiro for accepting to guide me throughout this thesis. His input was critical in helping write down and organize the myriad of ideas floating around in my head. Without his keen eye I'm sure this document would look far less perfect.

I'm grateful to my friend, colleague and partner at *seegno*, Jorge Pereira, for his suggestions and for being the one person around who really and fully understands the work developed, and for being able to engage in useful discussions. I'm looking forward and confident in our success together.

Also, the support of Sérgio Castro, Rui Marinho and Luís Faria - in this particular order - was essential to get me along with this thesis, together with that of my friends in general and the flickr photography community I am so involved with.

On another level, I would have never gotten this far without the support of my family, to which I owe too much to even dare write more than a simple yet heartfelt "thank you".

And finally, to the one I am not only in debt, but also in love with - the foundation of my life, my safe harbour, my doctor, friend and lover - Cecilia. It's amazing how we both managed to get along during what were the hardest times in our academic life. Our life together is just starting now.

Contents

List of Figures

List of Tables

Abbreviations

AJAX	**A**synchronous **J**avascript **A**nd **X**ML
API	**A**pplication **P**rogramming **I**nterface
CSS	**C**ascading **S**tyle **S**heets
DHTML	**D**ynamic **HTML**
DOM	**D**ocument **O**bject **M**odel
HTML	**H**yper**T**ext **M**arkup **L**anguage
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
IRC	**I**nternet **R**elay **C**hat
MDI	**M**ultiple **D**ocument **I**nterface
PBBG	**P**ersistent **B**rowser-**B**ased **G**ame
RIA	**R**ich **I**nternet **A**pplications
SDI	**S**ingle **D**ocument **I**nterface
STOMP	**S**treaming **T**ext-**O**riented **M**essaging **P**rotocol
TCP	**T**ransport **C**ontrol **P**rotocol
XMPP	**x**tensible **M**essaging and **P**resence **P**rotocol
YUI	**Y**ahoo! **U**ser **I**nterface

To my love, family and friends. And seegno.

Chapter 1

Introduction

In recent years, with the increased interest in web technologies, the world has seen major advances in the way data is handled, distributed and presented. Initially, there was a shift from static to dynamic web pages, allowing variable information to be displayed and receiving limited input from users [1]. Still, these early age web applications still depended on inconvenient and cumbersome methods of interaction with the user.

Nowadays, new developments allow for improved and smoother interaction, much closer to the desktop experience. This evolution in interaction between users and web applications gave birth to a plethora of novel applications and services that use the web as their medium, often surpassing the functionalities of the "desktop approach", mainly due to its mobility, wide reach and redundancy.

Web Applications are now often in use for a large variety of purposes, including desktop replacements such as instant messaging, mail, word processing, contact management, or totally new applications like discussion boards, wikis, weblogs, social networks, search engines and e-commerce.

1.1 Background

The Web as we know it, is only 5.000 days old. Ten years ago, it would have been impossible to predict its success. Nowadays with all the devices connected to the Web, we have created a world wide system which is running uninterrupted, with a complexity that remind us of the human brain. Everything will be part of the Web, every device made will be able to communicate through it, turning into the ultimate ecosystem. Not only we have machine-machine communications for the basic data packet sharing, we also have a two-way conversations between machines and people. Habits change as people start taking advantage of this tremendously disruptive technology. Information becomes a commodity while people discover new reasons to share it and new ways to interact. The information age is over and we are now into the participation age [2].

This paradigm shift is often associated with the term *Web 2.0*, introduced by Tim O'Reilly in "What Is Web 2.0" [3], the next generation of the Web. It's all about interaction, collaboration and social networking, where pages are no longer static but more dynamic and fluid. In the Web 1.0 everything was about connecting computers. In the Web 2.0, it is all has richer and more responsive applications [4], not only closing the gap with the desktop, but also presenting new interactions [5].

AJAX (Asynchronous JavaScript And XML) [6] is commonly used to improve the interaction between the user and the server, providing a richer and more interactive user experience, improving the Web application's interactivity, speed and usability [5, 7]. However, it is no longer enough and it's now part of a much larger puzzle [8]. The level of interaction that users expect from the Web is forcing it to evolve. New solutions and techniques are required in order to turn the Web from a one-way street into the much dreamed digital highway. Reverse AJAX is one such class of techniques, allowing communication to take place without first having to be requested [9], thus freeing the user for the burden of always having to make first contact.

1.2 The Problem

Techniques such as AJAX are already quite documented and becoming well established, with major players like Google, Yahoo! and Microsoft, already taking advantage of it.

On the other hand, Comet - the most interesting Reverse AJAX implementation - is a somewhat new concept undergoing lots of changes. As in other emergent Web technologies, achieving proofs of technical merits like scalability and network performance is still the main focus, not simplicity or ease of integration.

Given the current state of things, implementing real-time user interaction is a daunting task for most developers, not only due to the need to consult and digest a large amount of sources but also deal with the often conflicting information.

Also, there are no reference implementations of even the most basic solutions that can be implemented with Comet, and no standard means to integrate with even the most popular Web Application frameworks.

1.3 The Approach

To address this problem, a comprehensive study of Comet Implementations and AJAX Toolkits will be performed. An architecture combining both technologies will be designed, and integrated with a selected Web Application framework. Secondly, reference implementations will be developed for typical real-time user interaction situations.

To bring this work to life, a Persistent Browser-Based Game (PGGB) will be developed, applying the developed architecture and showcasing techniques and best practices for approaching real-time user interaction problems.

1.4 Objectives

In order to meet the challenged in bringing real-time user interaction to Web Applications, and given the current state of the Web and the rapidly changing technological landscape, this work aims to:

- Establish a ground work for other developers, by providing a synthetic analysis of AJAX and Reverse AJAX technologies.
- Simplify integration of these technologies, paving the way for widespread applications of real-time user interaction
- Develop integration between Comet, AJAX Toolkits and a popular Web Application framework.
- Develop reference implementations of common user interaction needs, namely real-time `Chat` and `Events` components.
- Build a PBBG which takes advantage of the developed techniques and components.

1.5 Overview

Web Applications have come a long way in the last few years, allowing users to experience increased levels of interaction. To better understand this paradigm shift, the past, present and future of the Web must be comprehended. Chapter 2 provides an overview over the techniques that have supported this evolution.

In Chapter 3, AJAX and Comet are brought together into a coherent and elegant whole. This architecture is designed in direct response to the challenges presented by the new paradigms of real-time user interaction in Web Applications.

Chapter 4 explores a PBBG, a Web Application that takes advantage of the developed architecture. Such a system is by its nature, a simulation, and as such leverages the well known effect of suspension of disbelief in order to merge people into a virtual world. In the context of web-applications, the problem with this setting is that people generally go on about their lives with the safe assumption that the world will not cease to function if they stop interacting with it. Events like sounds, smells and images will all reach their senses even if no special action is performed. In order to achieve a remotely comparable experience in an web game, a system that delivers events without requiring any user interaction is, ironically, a great advance in user interaction.

Chapter 5 concludes this thesis with an analysis of the work developed, plans and ideas for further improvements and the author's personal remarks.

Chapter 2

State of the Art

This chapter describes the evolution of the Web throughout its different phases. Moreover, it provides an overview of the state-of-the-art Web technologies that can be used to implement real-time user interaction.

2.1 Evolution

Web Applications have come a long way in the last few years, allowing users to experience increased levels of interaction. To better understand this paradigm shift, this section explores the past, present and future of the Web.

2.1.1 Web 1.0

The Web relies on the HyperText Transfer Protocol (HTTP), which is an on-request communication system. The client is the only one that can initiate communication. This worked fine, because the main objective of the Web was to provide access to static pieces of information. The content was generally published by users with technical skills and the user interaction was essentially limited to following hyperlinks and bookmarking.

The main problem with applications in this early stage of the Web was that they were based on pages [10] that had to be loaded in their entireness each time a user performed an action. This resulted in slow performance and limited interactivity, and was not even comparable to typical desktop applications [11].

At the time, caching was introduced as means to reduce the time latencies that users experience when navigating through Web sites [12], but it wasn't enough, and introduced problems of its own, with more recent dynamic data sometimes being obscured by cached versions.

2.1.2 Web 2.0

During their interaction in a Web page, users often take actions that require a response. This usually results in requesting a new page and has important consequences, the most dramatic being the bandwidth and latency between requests.

Currently, there's a paradigm shift in Web development which is improving drastically the quality of user interaction in Web Applications. This is mainly due to AJAX [6], and due to its momentum, the Multi Page Interface model is being replaced by the Single Page Interface model, helping the shift from desktop application towards Web Applications [1].

While in a traditional web application the user waits for a response and gets an entire page reloaded for each request, with an AJAX web application the user is not aware when the request is sent and can continue to interact with the web browser. The communication with the web server happens in the background and the response returned to the user's browser contains only a small amount of data, which is used to update the page without reloading [13].

This paradigm shift is often associated with the term *Web 2.0*, introduced by Tim O'Reilly in "What Is Web 2.0" [3], the next generation of the Web. It's all about interaction, collaboration and social networking, where pages are no longer static but more dynamic and fluid.

In the Web 1.0 everything was about providing access to information. In the Web 2.0, as Tim Berners-Lee said, it is all has richer and more responsive applications, not only closing the gap with the desktop, but also presenting new interactions [5].

Web 2.0 relies on several key characteristics, which Sun [2], has summarized into:

- **Using the Web as a platform.** In his popular 2005 Web-published article, "What Is Web 2.0?", Tim O'Reilly points out that instead of relying on a proprietary operating system as a foundation for applications, the new development model uses the Web as a platform and open standards as APIs for developing Web services. This has profound implications as new data sources become available and the programming tools to manipulate them allow even moderately skilled programmers to create new Web services. The end result is the availability of many novel applications that manipulate and display previously unavailable data. No longer confined to the desktop, these Web-based applications are available from any browser, in any computer.

- **Employing dynamic user interfaces.** Anyone who has used Flickr, GMail, Google Suggest or Google Maps will realize that a new breed of dynamic Web Applications has emerged. These applications are examples of how the *webtop* is replacing the desktop. Technologies such as AJAX, described in detail in Chapter 3, allow Web Applications to look and act more like desktop applications. The most noticeable difference is that the communication lag to and from the server is almost completely eliminated. This enables Web-based applications to provide functionality previously unavailable.

- **Encouraging user participation and collaboration.** A Web service based on the new model leverages its users to build and enhance its value. Google is the premier example of this approach. Every user who follows a Google search to a Web site refines the PageRank search process to provide more accurate search results for the next user.

2.1.3 Web 2.1

Web 2.1 is the author's interpretation of the next step in Web Applications, that is already underway. The Web is now inherently multi-user. Users no longer visit Web pages only to obtain new information. In this day and age, they are as much consumers as they are producers of information. In a way, there is a democracy of information on the web, where everyone with access to an internet connection is, simply put, a citizen.

Also, in the dynamic society we live in, people have a much more accute sense of their time and of how their time is worth. This creates the need to do and obtain, things and information, faster and with less effort. It is no longer acceptable to wait while a whole page loads, as it is no longer acceptable having to press the Refresh button, or similar, when there is the immediate need to receive new information.

In this whole context, not only people feel an increased need to stay up to date towards what others are doing on the Web, but also they want this to happen naturally and with the minimum effort possible. This is more than the simple transference to Web Applications of expectations built around desktop applications. This is natural human desire to communicate and socialize, on their own terms and schedule.

While the introduction of AJAX has revolutionized Pull technology on the web, Push technology isn't widely implemented yet. The ability to send information directly to users and its widespread implementation will help Web Applications to really improve the way they present data to users.

2.2 AJAX

While the ability to update a Web page incrementally isn't new [14], it only gained wide approach when it was coined to the AJAX acronym [1], by Garrett [6] in 2005. The advent of AJAX brought the Web to a different level, freeing users from having to wait for the information to reload after a click. Relying on standard HTTP, this pull technique means that clients perform a request to the server and receive a response with the requested data, therefore maintaining full control of the communication process. This is how the Web was planned in the first place, but with an ever changing technological and social landscape, new methods and interaction paradigms are developed to meet the always changing needs of users.

AJAX appears as a generic model to improve interactivity, responsiveness and overall allowing Web Applications [15] to overcome the start/stop mechanism inherent to the Web. As seen in Figure 2.1, AJAX reaches its goal by being placed in between user and server [6]. It is evident how activity stemming from the users results in a call to the AJAX engine sitting client-side, which can be either homegrown Javascript code or one of several available libraries. The AJAX engine sends a request to the server and then receives its response. Processing on the server side is done as usual with the server typically unaware of anything AJAX-related. Once the engine receives a response, it usually takes care of invoking client-side code to reflect the new information upon the page. This process is repeated for as long as the user interacts with the page.

Applications based in AJAX methodology rely on an engine written in JavaScript, which takes care not only of presenting the user interface, but also takes care of handling communication with the server. At the core of an AJAX engine there is a JavaScript object, XMLHttpRequest, which implements an asynchronous method of communicating with the server. This prevents the user from having to wait while the user interface communicates data with the server. This separation between data flow and the presentation side of the web page means that specific elements are refreshed as necessary, this eliminating the full page refresh [16].

A Web Application akin in functionality and interactivity to a desktop application is usually know as a RIA application. Applet, Macromedia Flash, DHTML and AJAX are some of the most popular RIA technologies. While others require a browser plug-in to function, AJAX doesn't, by relying on the ubiquitous JavaScript engine.

A good example of a RIA application is Google Maps[1]. Dynamic HTML (DHTML) is the best-practice use of the available standards for displaying and interacting with a HTML document using javascript and DOM (Document Object Model). It provides the visual effects for web applications such as drag and drop, hides, reveals and interactive validation, allowing the available downloaded data to be used in a more interactive way [10].

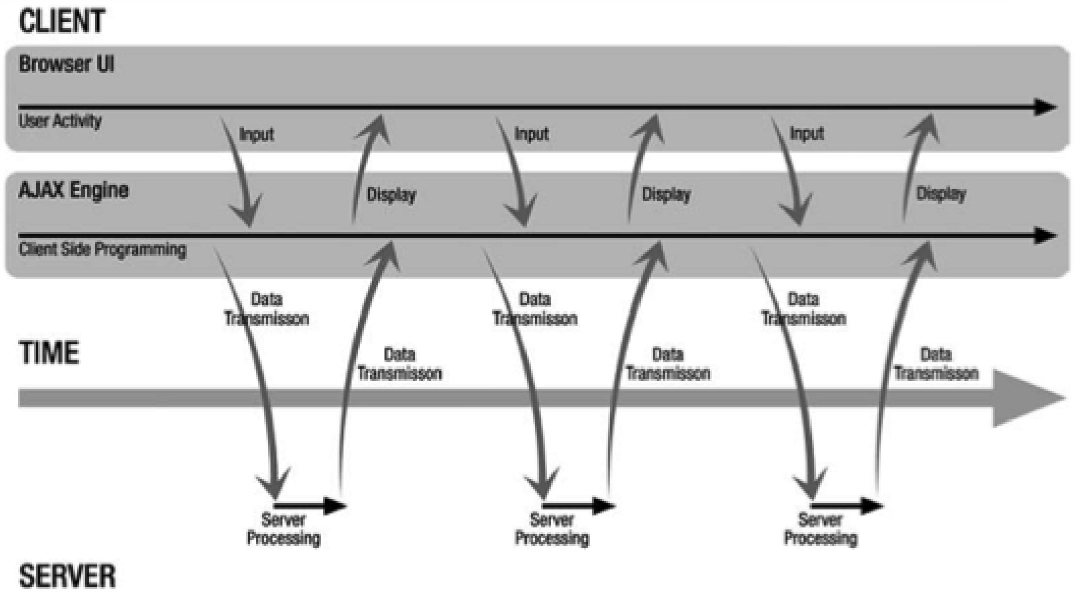


Figure 2.1: Simplified sequence of events in AJAX

[1]Google Maps Website: http://maps.google.com

The primary effect of AJAX is felt on the increasingly responsive and richer web applications. There is ample evidence in todays applications that take advantage of AJAX to support the claim that user experience is greatly improved. Next, we present these conjectures based in the work of Garrett Dimon, Cindy Lu and Coach Wei [5, 15, 17]:

Improved performance and smoother responses

Interacting with an AJAX page reveals a much faster response with only the relevant data being updated. This sort of interactions is on the opposite spectrum of having to wait for a reload of the entire page.

Automatic data refresh to keep the site updated

The information on a page can be kept up to date with AJAX performing a background poll for new data. It is also possible, for instance, to be constantly saving data without the user is on its way performing its tasks.

Visual effects enhance experience

New visual and animation effects are added to web applications for better communications with users. Fresh and light visual designs make Web 2.0 and AJAX-based applications look pleasant and inviting.

New functions and interaction techniques

New functions and interaction techniques such as collapse transition, expand transition, various ways to show invitations to actions, pop over windows are designed and implemented in AJAX-based applications.

These new interaction techniques create richer interactions and enhance the communications. In many AJAX-based applications, pop-up windows are replaced by pop over windows. The traditional OK and Cancel buttons are gone.

Multiple interactions

The interface and its presentation mechanisms are independent from the data communication process, which is asynchronous. This allows several operations to be initiated by the user without each having to wait for the underlying data communication to finish. Thanks to the asynchronous communication technique and the separation of data presentation from data exchange with the server, once the user submits a request, the user can move to another area immediately without waiting for the request process completed. In Flickr[2], for instance, you can activate the online editing feature for multiple pictures. You do not have to complete the edit in one form in order to activate another one.

Richer interactions on the Web

AJAX increases flexibility and productivity by allowing for the implementation of several mechanisms usually found in desktop applications, such as drag&drop, sliders, inline editing, etc.

Desktop interaction methods such as drag and drop, slider bar, zoom in/-zoom out functions, inline editing, etc. implemented in AJAX-based applications provide more flexible and powerful ways for users to interact with the web. For example, In Kiko[3], you can simply drag an appointment from one date cell to another to change the appointment date. Microsoft's live.com uses a slider bar for the user to change the view size of images. It also allows the user to drag and drop a selected image to a scratch pad.

[2]Flickr Website: http://www.flickr.com
[3]Kiko Website: http://www.kiko.com

2.3 Reverse AJAX

Reverse AJAX is a fairly recent technique, which got some highlights during the past year. Basically, the idea is to allow the server to push information directly to the clients, instead of having to rely on the client to pull information from the server. The main limitation of AJAX is exactly this inability to provide real-time information to the clients, because technically it is still bound by the underlying HTTP standard. So in practice, there is still the need to contact the server and ask for changes. While for the user this technological artifact might not be noticeable, it has its drawbacks, specially in terms of scalability.

Reverse AJAX is an attempt to bring real push capabilities to the web, and much like AJAX it's is not really a technology, rather a set of techniques to use other technologies in specific ways.

As previously discussed, classic non-AJAX web applications have a characteristic flow of events, as shown in Figure 2.2 [18]. This flow basically consists on user actions making the client perform requests to the server, which in turn performs appropriate processing and replies to the client with an entire new page that reflects the request. This process is repeated for as long as the user takes actions on the pages it is presented with.

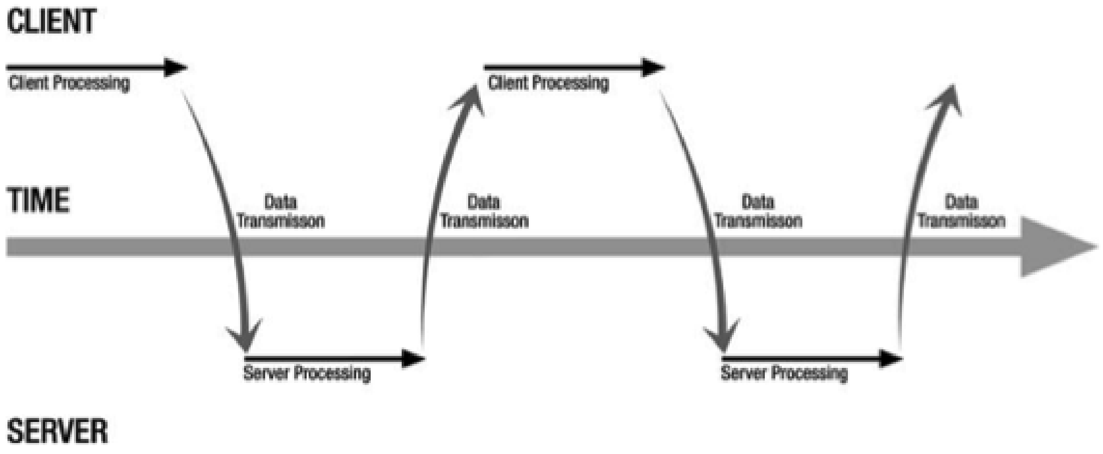


Figure 2.2: Basic sequence of events for the classic Web

With Reverse AJAX, a different flow of communication is introduced. Besides the sequence of events shows in Figure 2.1 [18], a server-side communication is initiated, thus providing for two-way communication.

Reverse AJAX is all about appearances and abstraction as it doesn't really exist, because the HTTP model with its stateless connections doesn't really allow for the server to push information to clients. There is no way for the server to start a connection to the client or to resume a connection that was somehow broken. However, a group of technologies allows for this problem to be abstracted from, and for all matters and purposes emulate the push behavior from the server. Currently, there are mainly three methods for achieving this effect.

2.3.1 Polling

Polling is an old technique, which basically consists of asking the same thing at given time intervals. It is comparable to a small child repeatedly asking "are we there yet?". The most basic example of this technique consists of the "<meta> refresh" tag in plain HTML, which consistently refreshes the whole page.

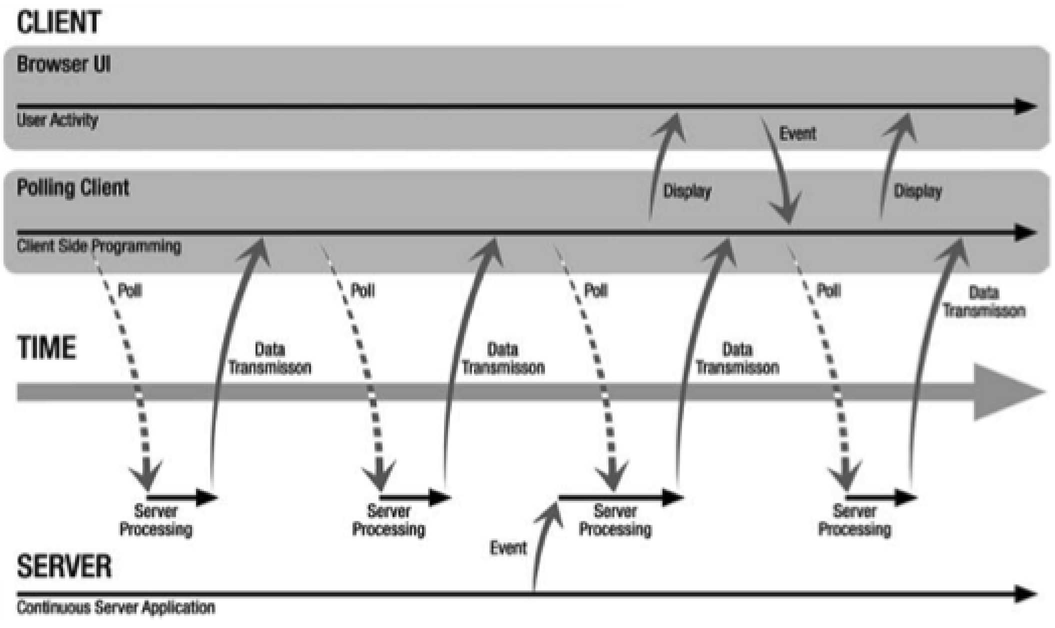


FIGURE 2.3: The sequence of events in the polling technique

As the new page replaces the previous one, unchanged data remains the same, but some data is replaced or shows up, which gives the appearance of an update. In terms of AJAX, a similar technique can be used to continuously update a part of a page. In practice, this is simply replacing user interaction with an automated request performed by the AJAX engine, as you can see in Figure 2.3 [18]. Notice that when no new data is present, the poll events do not result in any update being performed, as is the case in the first 3 poll events.

2.3.2 Piggybacking

Reverse AJAX can also be implemented through a piggyback mechanism, as shown in Figure 2.4 [18]. This is a passive method, as it requires user interaction in order to function.

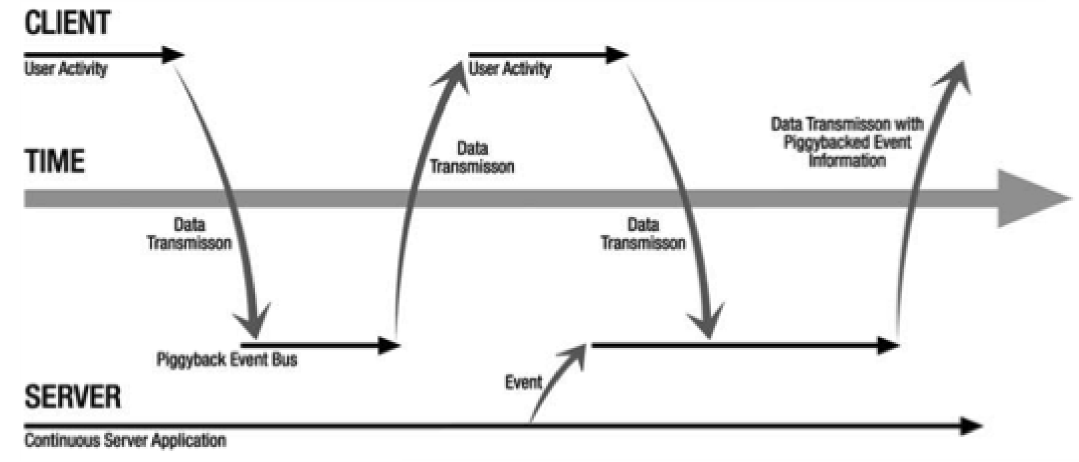


FIGURE 2.4: The piggybacking technique sequence of events

In this approach, whenever the server receives some AJAX request, it will reply with more than data strictly related to the request. Imagine a kid going to the neighbour to ask for a cup of sugar on request from his mother, and the neighbor replying "here's the sugar, and while you're here, also take your mother these fresh eggs I just got". Obviously, this saves someone a trip, so as an AJAX technique it shines in the fact that there is no extra load on the server. Information towards the client will be queued at the server, and sent it whenever a new request is performed by the client. One can also combine this technique with Polling and a large refresh time, so that updates are guaranteed to arrive even if no user action triggers the piggybacked reply.

Piggyback solves the problem of latency. However, another set of problems associated with the push approach still exists. Although a piggyback connection is shorter than the one used with push, it is still at risk of being closed because of a protracted idle period. To use piggyback, the server needs to authorize a request and allocate resources for keeping the connection alive for an unpredictable period of time [19].

2.3.3 Comet

With HTTP/1.1, the TCP connection between the server and the browser is kept alive until an explicit "close connection" message is sent by one of the parties or a timeout/network error occurs, thanks to the "persistent connection" feature. This allows the use of Comet [20] which is an umbrella term coined to Alex Russel, to describe a web application model in which a long-held HTTP request allows a web server to push data to a browser, without the browser explicitly requesting it.

Comet is also known by several other names, including AJAX Push, Reverse AJAX, Two-way-web, HTTP Streaming and HTTP server push among others and it has been implemented and included in several web servers including IBM Websphere [21] and Jetty [22].

In an application using streaming Comet, the browser opens a single persistent connection to the server for all Comet events, which is handled incrementally on the browser side. Each time the server sends a new event, the browser interprets it, but neither side closes the connection, as we can see in detail in Figure 2.5 [18].

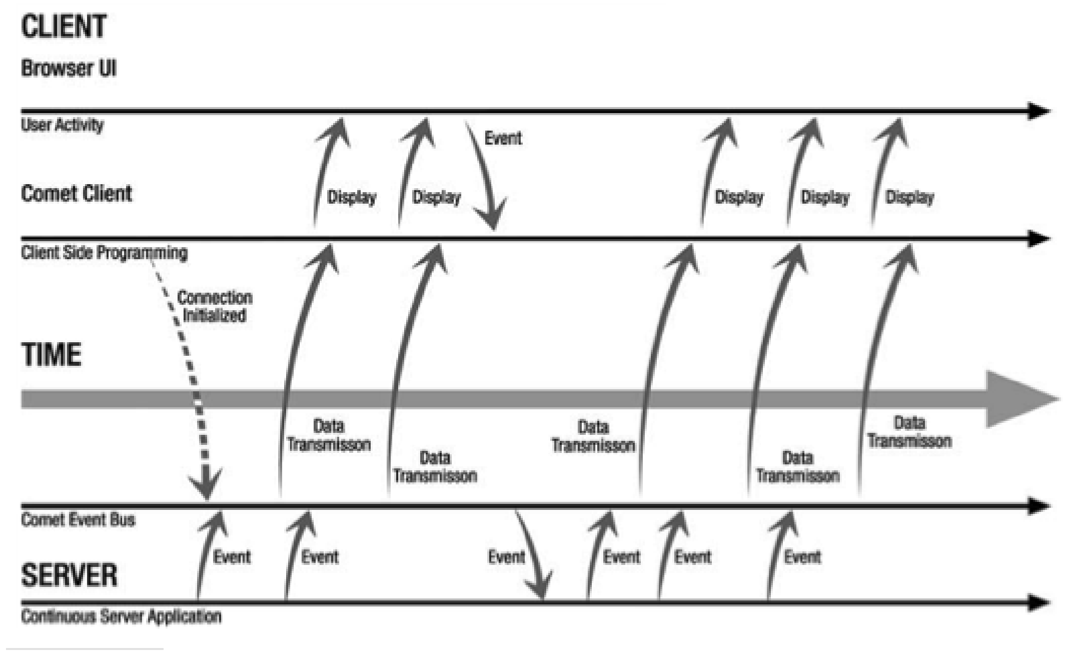


Figure 2.5: Comet sequence of events

It uses multiple techniques and hacks for achieving this interaction, in order to be cross browser compatible. These techniques relies in browser-native technologies such as JavaScript [23] and ActiveX, rather than on proprietary plugins, such as:

Hidden IFrame

A basic technique for dynamic web application is to use a hidden IFrame HTML element (an inline frame, which allows a website to embed one HTML document inside another). This invisible IFrame is sent as a chunked block, which implicitly declares it as infinitely long (sometimes called "forever frame"). As events occur, the IFrame is gradually filled with script tags, containing JavaScript to be executed in the browser. Because browsers render HTML pages incrementally, each script tag is executed as it is received. One benefit of the IFrame method is that it works in every common browser and it is also cross sub-domain. This technique has, however, three downsides: loading bar, hour glass and clicking noises in Internet Explorer browser.

XHR Streaming

The XMLHttpRequest (XHR) object, the main tool used by AJAX applications for browser–server communication, can also be pressed into service for server–browser Comet messaging, in a few different ways.

Instead of creating a multipart response, and depending on the browser to transparently parse each event, it is also possible to generate a custom data format for an XHR response, and parse out each event using browser-side JavaScript, relying only on the browser firing the `onReadyStateChange` callback each time it receives new data.

2.4 Other Technologies

There are other solutions besides AJAX and Reverse AJAX, currently in use to implement real-time Web Applications. Some rely on browser plugins, while others are still being drafted as standards. They are included here for reference only as they do not meet the requirements established for this work.

2.4.1 Flash XML Sockets

Flash sockets offer a solution that can be used in any browser that contains the plugin. The main disadvantage is that not all computer environments might have the plugin, and problems with the firewalls might become an issue [20].

2.4.2 Web Sockets

The HTML5 WebSocket standard defines the future of networking for browser applications, delivering full-duplex, bidirectional text-based communication between browsers and WebSocket servers [24]. The WebSocket connection handshake leverages the HTTP standard to integrate with existing Web infrastructure, such as proxies, firewalls and load balancers.

With WebSockets, browsers will be able to participate as first-class citizens in networked applications that are typically reserved for desktop clients, such as XMPP (Jabber) chat applications, as well as online gaming and financial trading applications.

In summary, it aims features such as seamlessly traverse firewalls and routers, allowing duly authorized cross-domain communication, integration with existing HTTP load balancers and with cookie-based authentication [25].

2.4.3 Server-Sent Events

The server-sent events defined in HTML 5 [26] defines a data format for streaming events to web browsers, and an associated DOM API for accessing those events. These events are sent to the client using a permanent HTTP connection with a content type “application/x-dom-event-stream” [27].

While the HTML5 specification is not in a finalized stage, the first public draft was published by the W3C in January 2008, and browser vendors have already began targeting features in the specification [25]. Ian Hickson, the editor of the HTML5 specification, says that this technology will enable all kind of real-time applications, without the hacks authors have to use in Comet implementations and such.

Chapter 3

Architecture

This chapter details the architecture developed by the author to combine an AJAX Toolkit, a Comet Implementation and a Web Application development framework. It begins with an analysis of existing alternatives and then lays out the requirements for such a solution. This conceptual information is then followed by the choices in design and development.

3.1 Analysis

Currently, there is no complete solution for real-time user interaction in Web Applications. HTML5 Server-Sent Event and Web Sockets could really improve this situation, but the technology is not yet available and will not be taken in consideration. On the other hand, Flash XML Sockets relies on third party plugins and with the proliferation of browsers and mobile devices, that isn't desirable.

AJAX applications are designed to have high interactivity and low user-perceived latency, but that isn't enough for real-time dynamic Web data, which needs to be propagated to users as soon as possible [28]. As for Reverse AJAX, adding Piggyback to the response would be a small improvement but not the solution to the problem. Using Comet's Hidden Frame technique is unacceptable for modern application usability requirements, since it displays the loading bar, hour glass pointer and causes clicking noises in Internet Explorer [29]. XHR Streaming is the best technique available, but it only works in Firefox and Safari [29].

Nevertheless, a viable approach is to abstract these several Reverse AJAX techniques, thus obtaining a coherent solution for this kind of problem. This solution has no problems with firewalls - as do others like *Flash XML Sockets* (Section 2.4.1) - and doesn't require plugins to be downloaded with modern browsers, relying solely on their JavaScript engine [20].

3.1.1 Requirements

For the design of this solution, there are key requirements that have to be met, not only by the final solution, but also by each of its underlying components, namely the AJAX Toolkit and the Comet Implementation:

Simplicity

Is defined as the effort needed to understand, design, implement, re-engineer, maintain and evolve a web application [1, 20]. It's a critical factor for the usage and wide acceptance of any new approach [1]. Due to the different implementations of AJAX and Comet in different browsers, it is essential to use well known libraries to increase simplicity and decrease the amount of code and learning curve.

Portability

Portability is the Software feature to be able to reuse the existing code instead of creating new code when moving software from an environment to another [30]. Web Applications that require extra actions from the user, such as downloading software to be able to run it such as Flash, or a plugin, decrease portability. Also, as JavaScript relies on different engines for different browsers, a component that works seamless in all browsers is indicated.

User-perceived latency

Is defined as the period of time between the moment a user issues a request and the first indication of a response from the system [1]. The most effective way to improve user-perceived performance is by allowing the user to interact asynchronously with the web application. This improves the application performance and keeps the user from refreshing the site to get the latest data.

Besides these global requirements, there are specific desirable characteristics for both the chosen AJAX Toolkit and Comet Implementation:

API Quality

A good API is intuitive, well structured and optimized for the most common usage, while flexible enough to cope with the specifics of each application.

Learning Curve

The learning curve is a measure of the complexity of the concepts that have to be learned in order to take advantage of a certain technology. It is a critical characteristic of any solution that aims to have widespread usage. This is one of the most important requirement when choosing both the AJAX Toolkit and the Comet Implementation, since the main goal of this thesis is to provide a solution without entry barriers.

Documentation and Community

The lack of good documentation can undermine the acceptance of even the best software. Together with the Community around a certain solution, it defines how easy it is for someone with a problem to find an answer.

Footprint and Efficiency

For an AJAX Toolkit this mainly means the size of the library that has to be downloaded together with the Web Application. For the Comet implementation this is related to the overhead of the communication packets.

License

In order to promote widespread usage, an unrestrictive license is of the upmost importance.

3.1.2 AJAX Toolkits

Using an AJAX toolkit allows developers to abstract from the complexity of developing AJAX applications - which is a tedious, difficult, and error-prone task. It also shields developers from the complexity of client communication and the incompatibilities between different web browsers and platforms. Therefore, the choice of AJAX Toolkit - its maturity, simplicity and completeness - play a major role in the success of a web application.

Several solutions were evaluated, some of which didn't meet the minimum requirements defined for the architecture. Based on the author's experience and the comparison made by Crandall [31], the final contenders, Prototype, Dojo, YUI and jQuery, can be compared in Table 3.1. The main features analyzed are the learning curve, API quality, documentation and fingerprint. Based on this, jQuery is the one that stands out for its simplicity to learn.

	Prototype	Dojo	YUI	jQuery
Learning Curve	Average	Bad	Average	Good
API	Average	Bad	Average	Average
Documentation	Average	Bad	Good	Bad
Filesize Range (KB)	46-137	18-276	2-300	10-44

Table 3.1: AJAX Toolkit Comparison

The Dojo Toolkit

The Dojo toolkit, like most AJAX Toolkits, provides a rich set of utilities for building responsive applications. The project started in 2004, it is backed by Dojo Foundation and sponsored and partnered with IBM, Sun Microsystems, Sitepen, AOL, BEA and Zend. Its initial aim was to prove that JavaScript and DHTML should be taken seriously. Besides full AJAX integration, Dojo provides cross browser 2D drawing APIs, offline data storage and history management, among other features.

The Yahoo! User Interface Library

The Yahoo! User Interface Library (YUI) was developed by Yahoo! for internal use, but has been open sourced. It is one of the most well documented JavaScript libraries, being extensively tested and focused in browser issues. Yahoo! uses it for all its services, and therefore the library is robust.

Prototype

Prototype is a JavaScript Framework that aims to ease development of dynamic Web Applications. It is a relatively simple layer that offers both shorthand versions for popular functions and a good amount of cross-browser abstraction [32].

As Daniel Carrera summarized [33], the disadvantages of using Prototype are:

- Pollutes the global namespace with many functions.
- Modifies base classes of JavaScript.
- Tries to change the nature of the language.

Extending base classes can break existing valid code, break other libraries, make the library less portable and make it harder to tell which features are part of JavaScript and which were added by the library [33].

jQuery

jQuery is a fast and lean JavaScript Library that simplifies HTML document traversing, event handling, animating, and AJAX interactions for rapid web development. It should appeal to the disciplined programmer who enjoys the rigor of its chained JavaScript and the power it packs into a few simple lines [32]. It is a fairly unique library in that it is so focused on making the DOM easier to manage and largely ignores other aspects of JavaScript programming [34].

3.1.3 Comet Implementations

Choosing a Comet implementation isn't easy since there is a wide range of options. The Table 3.2 presents the list of the current stable and multiplataform Comet implementations, according to *cometdaily.com* [35], featuring GlassFish, Jetty with Cometd, Lightstreamer and Meteor. This comparison details each implementation at several levels, such as supported transports, horizontal and vertical scalability and more. Meteor is the winner when referring to protocols, since it is configurable at that level.

	GlassFish	Jetty+Cometd	Lightstreamer	Meteor
General				
License	Apache 2.0	Apache 2.0	Commercial	GPL v2
Protocols	Bayeux	Bayeux	Bayeux	Configurable
Transports				
Polling	yes	yes	yes	yes
Long-Polling	yes	yes	yes	yes
Callback-Polling	yes	no	yes	yes
Iframe Streaming	no	yes	yes	yes
HTMLfile Streaming	no	no	yes	yes
XHR Streaming	no	no	yes	yes
Multipart Streaming	no	no	yes	yes
Scaling				
Horizonal (Nodes)	yes	yes	yes	yes
Vertical (Clients)	> 40,000	> 20,000	> 5000	> 10,000

Table 3.2: Comet Maturity Guide

GlassFish

GlassFish runs on top of Java, so it can be installed on any OS that supports the Java implementation. Its best features include enhanced clustering capabilities, which optimizes high availability and scalability for deploying architectures through in-memory session state replication [36].

Jetty with Cometd

Jetty is available on Java 1.4 and above, which makes it available on a large range of devices, from Android mobile phones to large multicore servers. It provides the core Cometd Java Bayeux implementation and it is optimized for the Jetty asynchronous features [36]. Jetty has an implementation of the Cometd event server that uses the Continuation mechanism for asynchronous servlets [37].

Lightstreamer

Lightstreamer, like GlassFish and Jetty, is deployable on any platform with Java. Adapters are available in Java, .NET and TCP sockets while Clients are based on Flash, Flex, Java SE, Java ME, and .NET. It uses it's own network protocol [36].

Meteor

The server daemon runs on any platform for which Perl is available and transports are completely configurable within simple constraints [36]. As its creator, Andrew Betts, mentioned in Comet Gazing Maturity [36], Meteor is all about simplicity, ease of use and getting on with solving a single problem effectively. It is a standalone tool to complement existing web applications, doesn't show any bias toward any framework or technology, and can be set up and run by anyone.

Orbited

Orbited[1] isn't listed in Table 3.2, because its author still doesn't qualify it as stable. As reasons for this choice, he points out [36] that the technology isn't mature enough in itself for any implementation to be deemed stable. Still, it's one of the most featured full Comet implementations available. It provides a pure JavaScript/HTML socket in the browser and it is a web router and firewall that

[1]http://www.orbited.org

allows one to integrate web applications with arbitrary back-end systems [2] and has been tested against 10,000 idling clients without problem [36].

Orbited allows you to write real-time web applications, such as a chat room or instant messaging client, without using external plugins like Flash or Java. It ships with support for many protocols out of the box and it works in all browsers, cross-port and cross-subdomain, with no loading bars, clicks, or spurious history entries. More, Orbited's shared-nothing architecture supports arbitrarily large server clusters, which are perfect horizontal scalability.

3.2 Design

A solution to integrate a Web Application with a Comet Implementation should be as simple as possible to setup, yet powerful enough to evolve. The Comet Implementation should be independent of the Web Application, so different technologies can be used with the least impact on how the Web Application is developed. Furthermore, the use of a personalized protocol should be allowed, so the user does not need to learn and implement new protocols.

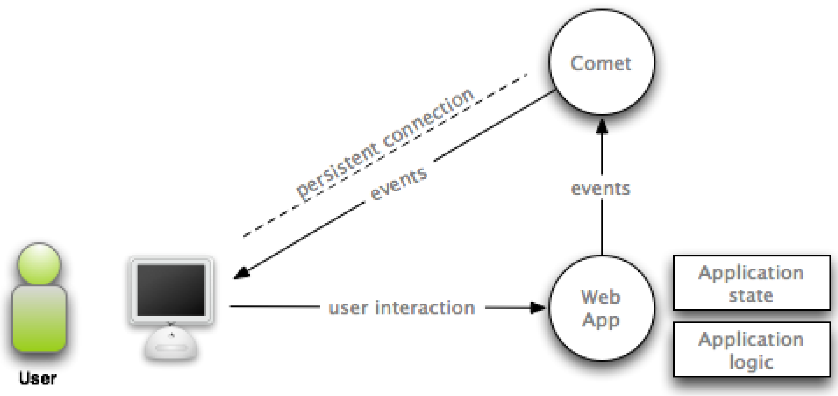


FIGURE 3.1: Solution design model

Figure 3.1 illustrated the chosen design model. The user connects to the Web Application and creates a persistent connection with the Comet Server. Any user

[2]Orbited Website: http://www.orbited.org

interaction will have an impact on the Web Application logic and data model. Some modifications to the data model will be propagated as events, to the Comet server, divided in channels. Certain user components will subscribe to these channels, and immediately receive these events. This is a well known design pattern, called publish/subscribe, where subscribers register their interest in an event, or a pattern of events, and are subsequently asynchronously notified of events generated by publishers [38]. This is how real-time communication is achieved.

Even if they can be seen as independent, the protocols used by the Comet Implementation must be supported by either the AJAX Toolkit or the application itself. Generic protocols are far too complex for the majority of applications, so, developers will usually create a personalized protocol for their application. Therefore, the best solution should be one that allows personalized protocols, and as many generic protocols as possible.

Cometd is a scalable HTTP-based event routing bus, comprised of a protocol specification called Bayeux, Javacript libraries (Dojo Toolkit), and an event server [37]. Using Cometd and Jetty for the server side and Dojo for the client side, we are able to provide a push-based client-server communication [1]. But this combination only supports the Bayeux protocol, and has no support for personalized protocols. Even more, Dojo has a big learning curve when compared with the other analyzed toolkits. As for Glassfish and Lightstreamer also do not support other protocols than Bayeux, making them a bad choice for any combination.

Both Orbited and Meteor support custom protocols, but only Orbited [39] has an extensive list of generic protocols support, such as IRC, XMPP, and STOMP, as detailed in Figure 3.2 [40]. This makes Orbited the best suited Comet Implementation when choosing a generic protocol, but on the other hand, Meteor presents the simplest Comet Implementation in regard to personalized protocols. Thus, integrating both Orbited and Meteor into a solution, would be the ideal, giving the developer the power of choice. Its worth noting that Orbited used to use the ”orbit protocol” for dispatch, before it switched to STOMP and ultimately TCPSocket+STOMP.

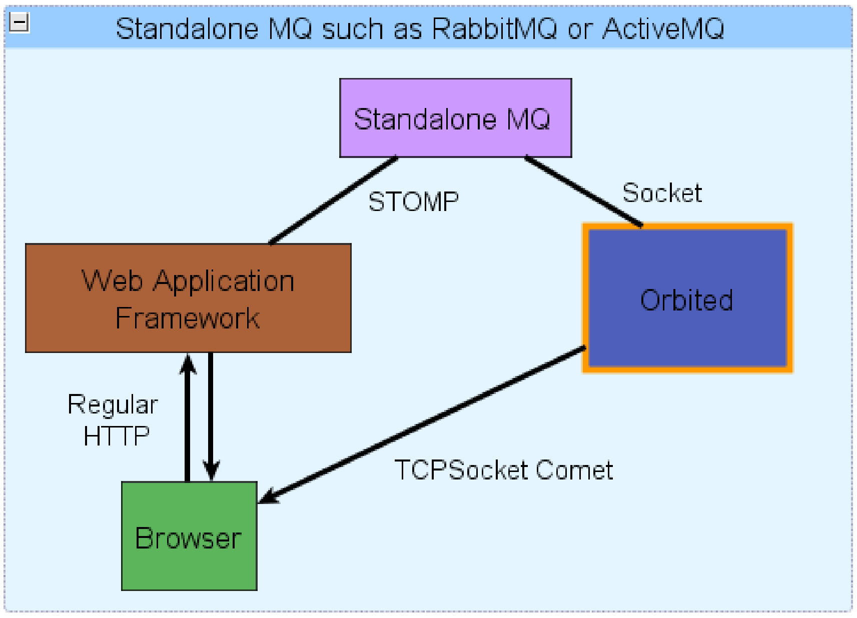

Standalone MQ such as RabbitMQ or ActiveMQ
Standalone MQ
STOMP
Socket
Web Application Framework
Orbited
Regular HTTP
TCPSocket Comet
Browser

FIGURE 3.2: Web Application Framework Integration with Orbited

Regarding the AJAX Toolkit, any choice would have to implement support for these generic protocols, seeing as none is available. Considering this, one should choose the toolkit with the most active community, and hope or ask for someone to implement it. jQuery fits the bill perfectly, with its ample documentation and vibrant community. On the technical side, its small footprint and the extensive plugins repository, makes it the best combination for Orbited and Meteor.

3.3 Development

Many Web Applications nowadays are developed with the aid of frameworks such as Django, symfony or Ruby on Rails. Such frameworks allow the addition or extension of functionalities through plugins. To ease Comet integration with such platforms, the ideal method would be to create a plugin that combines jQuery with Meteor and Orbited.

This section presents such a plugin for the symfony framework, `sfCometPlugin`.

3.3.1 Meteor

Integrating Meteor with the Web Application is quite simple. It requires a server with Perl and Apache2. The first step will be to download and run the Meteor daemon server.

Proxying

To setup the Meteor Server, the documentation in *meteorserver.org* suggests methods that include using firewall rules (iptables), having an extra *IP address* or having Apache in one machine and Meteor in another. These options are unpractical since they require administration privileges and/or deeper technical skills.

A simpler method is to setup the Apache server as a *Reverse Proxy*, using *mod_proxy*, passing all requests made to a specific subdomain (eg: comet.domain.com) to the Meteor daemon. This configuration is simple to understand and setup, as seen in Listing 3.3.1.

```
<VirtualHost *:80>
  ServerName comet.domain.com
  ProxyPass / http://localhost:4670/
  ProxyPassReverse / http://localhost:4670/
</VirtualHost>
```

Listing 3.1: Virtual Host Configuration

Client Side

Every time we need to use Comet for any component, we must include Meteor's script from subdomain *comet.domain.com*. `sfCometPlugin` provides a initializer for this, and when the `sfMeteorHelper` is included, Meteor JavaScript file is automatically loaded, as well as `sfMeteor.js` which contains convenience methods that wraps the need to have multiple Meteor callbacks, as can be seen in Listing 3.4.

```
<script type="text/javascript" src="subdomain/meteor.js"></script>
<script type="text/javascript" src="domain/sfCometPlugin/js/sfMeteor.js"></script>

$(document).ready(function() {
  initializeMeteor();

  ... component code
});
```

Listing 3.2: Including Meteor script manually

Including Meteor in a symfony template is as simple as Listing 3.3 describes. It loads the MeteorHelper, and initializes the Meteor connection, that could be used later when needed.

```
<?php use_helper('Meteor') ?>

<?php initialize_meteor() ?>
```

Listing 3.3: Including Meteor automatically

`sfMeteor.js` file is a dynamic JavaScript file, generated by symfony. It attaches a unique user id required by Meteor, in order to successfully deliver the messages.

```
/*** Meteor */
function initializeMeteor() {
  Meteor.host = "comet." + location.hostname;

  // Set this to something unique to this client
  Meteor.hostid = "<?php echo md5($sf_user->getUser()->getUniqueIdentifier()) ?>";

  Meteor.registerEventCallback("process", process);
}

function process(response) {
  try {
    if (response) {
      eval("var data = " + response + ";");

      eval(data.method + "(data);");
    }
  } catch(error) {
  }
}
```

Listing 3.4: sfMeteor.js

Meteor Class

Meteor class (Listing 3.5) implements the Comet interface, defining methods such as `addMessage`. This method is responsible to send a message to a specific channel, where it will be delivered to the users that subscribed to that same channel.

```
class Meteor implements Comet
{
  public static function addMessage($channel, $message)
  {
    $controller = fsockopen("127.0.0.1", 4671, $errno, $errstr, 5);
    $message = "ADDMESSAGE " . $channel . " " . $message . "\n";
    fwrite($controller, $message);
  }
  ...
}
```

Listing 3.5: Meteor Class

3.3.2 Orbited

Orbited is written in Python and requires Python 2.5 or newer. Orbited also requires Twisted, which it installs automatically on most platforms (and is otherwise installed trivially). Additionally, from 0.5 and onwards Orbited enables a revolutionary interaction model centered around the TCPSocket [3].

Orbited Class

Just like `Meteor` class, `Orbited` class implements the `Comet` interface, and can be used to send events to Orbited through symfony.

```
class Orbited implements Comet
{
    public function addMessage($channel, $message)
    {
      if (!$this->connected) $this->connect();

        if (!$this->socket)
```

[3] http://orbited.org/wiki/Installation

```
        {
          throw new Exception('Connection Lost');
        }
        try {
          $this->id = $this->id + 1;
          $this->sendline(self::VERSION);
          $this->sendline('Event');
          $this->sendline('id: '.$this->id);
          $this->sendline('recipient: '.$recipient);
          $this->sendline('length: '.strlen($body));
          $this->sendline();
          fwrite($this->socket, $body);

          return $this->read_response();
        } catch (Exception $e) {
          $this->disconnect();
          throw new Exception('Connection Lost');
        }
    }
  ...
```

Client Side

The essential file that should be included whenever Orbited is used is `TCPSocket.js`, and could be integrated as seen in Listing 3.6. Orbited initialization, depends on which protocol the developer chooses, as it provides protocols such as STOMP, IRC and XMPP. To achieve the same functionality as Meteor, one could simply use a STOMP server to integrate it with his Web Application.

```
<script type="text/javascript" src="http://domain.com:8000/static/TCPSocket.js">
</script>
```

Listing 3.6: Including Orbited script manually

Chapter 4

Case Study: simX

This chapter explores `simX` , a PBBG with unique requirements in terms of real-time user interaction. In `simX` , the user plays the role of a Puppy manager, where he can buy, trade and develop its Puppy characters. Playing the game involves rapidly reacting to the events happening in the game world. For this purpose, the user can rely not only on wits but also in leveraging the power of his community, in order to face the challenges presented. As such, communicating with other users and being aware of the events is a critical factor for success.

4.1 Introduction

There are plenty of online games nowadays. It can be said that the world of online gaming - both browser and non-browser based - is going through a ”golden era”. So, what’s the point of developing yet another one? There are two reasons people play browser games: one is that they offer a in-depth/enjoyable game that can be played anywhere, and two, the community [41].

Along with this successful recipe - or maybe as a consequence of it - many things in browser-based games still work as they did ten years ago. In fact, most of them still feel like a group of forms; in many you still have to operate the game’s features in the same way you operate a registration form.

Ten years ago this was more than enough: networks weren't that fast and Web technology wasn't the most interesting thing, but nowadays we have come to a point where there's a possibility to really feel you are in a game. To achieve this, we are sure to need:

Interaction

Quick, beautiful, painless interaction. Talking with friends, planning a raid to enemies, inviting people to trade items, it all has to be possible, and the overall usability of it has to be as pleasant as in a regular 3D game.

Ambiance

The player is inside a browser, but this is a role-playing game. One of the trickiest and most fundamental things to enhance the user's experience is to create an environment that feels real. We're inside a browser, but the player must be past it. To achieve that we have some handy tools: a good storyline, the possibility to evolve the character (you're bound to a personality stereotype when you create your character, but it develops with your actions throughout the game), and the non-linearity of the process. Which leads us to the next needed thing.

Randomness

Sure, the player has to be able to somehow predict what's the next step to take, but we're aiming at simulating reality: you can never be sure that something's going to happen the way you want to. If a player attacks another with fifty gang members and the other one only has five employees defending the territory, chances are it's an obvious win - but what if there's an inconvenient earthquake right in the middle of the attack? And if you're earning some real money because you have this very very good worker, how can you be sure that tomorrow he or she's still alive?

Objectives

We're not aiming blindly at making a full-out war between players. Being a war lord is one of the options you have, but you can be powerful through business, intimidation, societies or plain luck. Imitating life - that's the point.

Power

This is the thing that drives all people to play. The world of online gaming survives only because, deep inside, the player feels powerful. As he grows in the game, the smaller players seek out for his help, and the bigger ones start to notice his presence, keeping the ”food chain” alive endlessly. This is a key need for two reasons: it brings new players (captivated by older ones or just by virtual word-of-mouth) and it feeds imagination in role playing.

Using the latest cutting edge technologies it's possible to achieve these goals. It makes no sense to build another regular browser-based game, but it does make sense to build an interactive role-playing experience which can be as fun and beautiful as possible inside that same browser. It's not a new thing, but it is a new way of seeing it.

4.2 Requirements

The main goal for this case study is to prove that the Web is capable of leveraging new technologies to improve interaction between the users and Web Applications. We intend to show how to integrate real-time components easily into a Web Application. Not only real-time components can enhance the gaming experience. Three components will be presented in this chapter, and all of them could be used in almost every current Web Applications that have a heavy business logic and user base:

Hijax Component

Reloading the page just to get a different piece of information isn't always necessary. Using Hijax, a technology that combines AJAX and Javascript itself, we can provide a Web Application similar to a Desktop application, only changing the page components that would have different content.

Chat Component

It has been described throughout this document that one of the most interesting and under-explored areas in browser-based gaming is peer-to-peer interaction or any real-time communication. There are ways to solve these issues - technically speaking - but most of them never really needed new technologies to be improved: they just needed a different approach towards the overall gaming concept. One of those issues was the real meaning of communication: talking with other players. Real, plain text ping-pong.

Event Component

In a system where changes may occur at any moment, it's important to have a mechanism which alerts the user to the consequences of any event related to him.

The event system should alert the user in real-time of any important activity. This should increase gameplay since it is the component that will warn the user about all the things that happens around his objects, friends and societies.

4.3 Technology

`simX` is built using **symfony**, a PHP 5 framework, for the server-side language. It was chosen to implement this game with a framework because they are designed to support and provide facilities for the creation of Web Applications, automating many of those tasks and delegating boilerplate to configurations.

Frameworks typically are designed with certain design paradigms in mind, the most notable case being the Model-View-Controller (MVC) pattern [42] for separating business rule related code from presentation and control, or achieve conciseness by avoidance of repeated code. Frameworks exist for many languages and platforms; some of most famous ones are Django[1] (Python), Rails[2] (Ruby) or Catalyst[3] (Perl).

In Figure 4.1 is presented the architecture of `simX` . As we are going to implement the solution designed in the previous chapter, we will use `sfCometPlugin` (Section 3.3) to integrate Comet and AJAX within the Web Application. When building the components, the AJAX Toolkit will jQuery, which is bundled with `sfCometPlugin`. Regarding the Comet Implementation, the developer should choose wether to use Meteor or Orbited, depending on the business logic and component requirements.

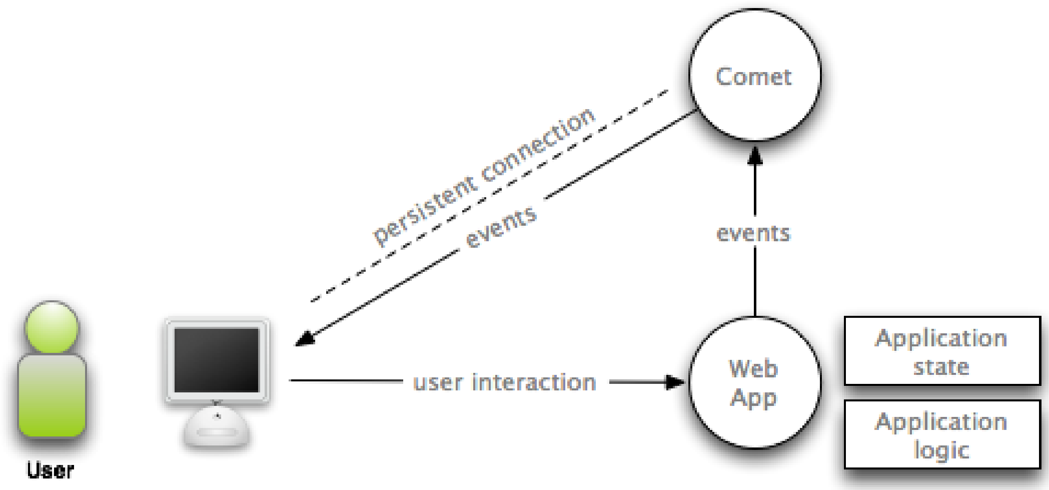


FIGURE 4.1: simX Architecture

[1]Django Website: http://www.djangoproject.com
[2]Rails Website: http://www.rubyonrails.com
[3]Catalyst Website: http://catalyst.perl.org

4.4 Hijax

Hijax [43] is a method that consists in transforming a Multiple Document Interface (MDI) in a Single Document Interface (SDI), where a more fluid interaction experience, closer to the desktop approach, is provided. This is a key feature that's commonly implemented in modern Web Applications, helping saving bandwidth.

4.4.1 Requirements

As some browsers and/or users don't have JavaScript enabled by default, the Hijax solution must degrade gracefully. Giving feedback to the user that the page is loading is very important as well, because it's a new way to interact with the application. Feedback should also be provided to the user using a loader, since it is the most commonly accepted visual interaction. This loader should be only displayed in the areas being updated, indicating that the application is still running.

Bookmarking links and keeping them spiderable is also a fundamental requirement, avoiding breaking the current model. As new improvements are added to the application itself, it is important to provide a seamless integration with the appropriate accessibility. Allowing the use of back and forward buttons is a step in that direction, a feature most of the AJAX implementations still lack.

4.4.2 Architecture

Using JavaScript to intercept links and form submissions, that are using class "hijax" and passing the information via XMLHttpRequest instead of following the link or posting the form, one can then select which parts of the page need to be updated instead of updating the whole page. This alone won't make applications really rich - AJAX is a lot more than fusing blocks of content onto the page. However, it's certainly a handy technique for making applications a bit richer, while gracefully degrading and with minimal programming effort [44].

4.4.3 Implementation

Using unobtrusive JavaScript, using jQuery, we are able to target all the links that we want to *hijax*, as detailed in Listing 4.1. The function `pageLoad` is responsible to load a remote page and changing the content with the newly updated page. Using this implementation, we are able to bookmark hijax pages, since we update the window hash, an anchor used to navigate withing a webpage. To enable back and forward navigation, we are using the jQuery History Plugin [4].

```
$(document).ready(function() {
  function pageload(hash) {
    if (hash && (window.location.hash[1] == '/')) {
      $("#container").load(hash, '', function() {
        $('#inside').show('normal');
        $('#indicator').fadeOut('normal');
      });
    }
  }

  $.historyInit(pageload);
});

function loadPage(page) {
  var hash = page.attr('href');
  hash = hash.replace(/^.*#/, '');

  $('#container').hide('fast',function(){$.historyLoad(hash)});
  $('#indicator').remove();
  $('#indicator').fadeIn('normal');

  return false;
}

// Unobtrusive javascript, that targets hijax links
function hijax() {
  if (window.location.pathname == '/') {
    $('a.hijax').click(function() {
      loadPage($(this));
    });
  }
}
```

Listing 4.1: Hijax implementation

[4]History Plugin Webpage: http://plugins.jquery.com/project/history

4.5 Chat

Nowadays we can find many chat-like solutions to enhance communications. Some of them lack the true meaning of real-time, being based on timers, and some others are really excellent implementations, such as Google Talk[5] or meebo[6]. Those were the examples we wanted to follow and integrate in our project, and enhance so that our chat system could become a core, essential piece in the game.

4.5.1 Requirements

The goal was to use the chat as a tool for the game, and not as a simple means of communicating, which means that not only the player had the ability to talk with other players, it had to be able to browse through different channels.

For instance, if all the players around the world were to communicate in a single chat window, the confusion would be impossible to overcome, and the main goal of this would be thrown away, but if there were different channels for context-based communication, things could become really interesting and ease player interaction. Besides that, a context menu, shown when clicking on each available player in the chat room, made possible to act on each player, helping the client to really understand what he could do or not do with each of them. The somewhat unrealistic scenario of doing things in group without really understanding what the group was thinking was replaced with a fully context-driven component which allowed everyone to interact directly with the course of information - in real-time. `simX` should support different channels such as:

- General: where all the players can chat.
- Society: where only society members can chat.
- Trade and Private: specific channels for private conversation.

[5]gmail Website: http://www.gmail.com
[6]meebo Website: http://www.meebo.com

4.5.2 Architecture

There are a wide range of applications based on online chat, such as meebo[7], GMail or even facebook[8]. These applications rely on a Jabber IM server as the backbone of the chat component. Real-time communications with clients are performed by the server via XMPP, an open XML based protocol that works on a concept similar to email [45]. In our case, we wanted to do something simpler, that could be integrated easily in any Web framework.

It will be presented two implementations, to demonstrate the power of both Meteor and Orbited. Using Orbited integrated IRC support, a common, text-based protocol that is popular for chat, and where any developer can easily reflect the channels configuration. On the other hand, it is also possible to integrate a chat application easily within a Web framework using Meteor.

4.5.3 Meteor Implementation

Meteor is the component where the messages are published to each channel and transmitted to the subscribed users. There are a wide list of chat channels, such as general, society, trade and private. The most difficult ones to implement are the society, trade and private, since there's the need to secure the data, and control who can subscribe to those channels. This can be achieved using a message hashing scheme, but as this isn't in the scope of this thesis, it would be implemented as future work.

AJAX will be used to synchronize the user list from the different channels, from time to time, because Meteor alone can't know weather or not a user disconnects without sending the disconnect event. Other methods could be used, such as the server sending a list of online users periodically to every user.

Using Meteor's function `joinChannel` (Listing 4.2), we are able to subscribe to any channel as long as we know its name. The second parameter is the number

[7]meebo Website: http://www.meebo.com
[8]facebook Website: http://www.facebook.com

of past messages we want to view. Using jQuery, it's really easy to toggle chat channels, using *CSS* and the jQuery functions `show()` and `hide()`.

When opening a page where the chat module is active, the first thing to do is initialize the Meteor connection as can be seen in Listing 3.3.

```
function channel(name) {
  try {
    Meteor.joinChannel(name, 11);

    Meteor.connect();
  } catch(error) {
  }

  // Hide previous channel
  ...

  // Display selected channel
  $('#chat #' + name).show();

  // Updating current channel
  ...

  // Focus in the input box
  $('#chat #talkbox').focus();
}
```

Listing 4.2: Changing a channel

One of the most important calls in the chat module is the chat function (Listing 4.3). Its main function is handling incoming messages that are tagged with chat function, being dispatched from the main callback function into the following function:

```
function chat(data) {
  $channel = $('#chat #' + data.channel);

  $channel.append('<p class="' + data.channel + '">');
  $channel.append(data.username + ': ' + data.message + '</p>');
}
```

Listing 4.3: The chat function

To send messages to the Web Application, a remote form is used, using symfony jQuery Helper, as can be seen in Listing 4.4. After being sent, the message is handled by `simX` , where is parsed and sent to the Meteor daemon. The message is dispatched to Meteor using the code shown in Listing 4.5.

```
<?php echo jq_form_remote_tag(array('url'     => '@chat',
                                    'loading' => '$("#talkbox").val("")')) ?>

  <?php echo input_tag('talkbox', '', array('class' => 'textbox',
                                            'autocomplete' => 'off')) ?>
  <?php echo input_hidden_tag('channel', '') ?>

  <?php echo submit_tag('', array('title' => 'Say it in the chat window',
                                  'value' => 'Say it', 'class' => 'button')) ?>
</form>
```

Listing 4.4: Chat remote form

```
public function executeMessage()
{
  $channel = strip_tags($this->getRequestParameter('channel', 'general'));
  $talkbox = strip_tags($this->getRequestParameter('talkbox'));

  // Is there a message?
  $this->forward404Unless($this->getRequestParameter('talkbox'));

  $message = array('method'   => 'chat',
                   'channel'  => $channel,
                   'username' => $this->getUser()->getUsername(),
                   'message'  => $talkbox);

  Meteor::addMessage($channel, addslashes(json_encode($message)));

  // Save message to database
  $chat = new Chat();

  $chat->setChannel($channel);
  $chat->setMessage($talkbox);
  $chat->setUserId($this->getUser()->getId());

  $chat->save();

  return $this->renderText('OK');
}
```

Listing 4.5: symfony chat action

4.5.4 Orbited Implementation

Orbited comes with IRC.js, a complete IRC implementation that helps the implementation of chat components, the considered "Hello World" of Comet. *TCP-Socket* is used to connect to a remote server and the Orbited daemon serves as the bridge between scripts in the browser and the target TCP server. In the example given in Listing 4.6, just the global channel is implemented, but the concept is the same for the remaining.

```
var hostname = "irc.domain.com"
var channel = "#global"
var port = 6667
nickname = "<?php $sf_user->getNickname() ?>"
var client = new IRCClient()

client.onopen = function() {
    client.nick(nickname)
    client.ident(nickname, "8 *", "<?php echo $sf_user->getName() ?>")
    client.join(channel)
}

client.onmessage = function(sender, place, msg) {
    if (place == channel) {
        var nick = sender.slice(0, sender.search("!"))
        print_output(nick + ": " + msg)
    }
}
var print_output = function(s) {
    var board = document.getElementById("board")

    // make output HTML safe
    s = s.replace("&", "&amp;", "g").replace("<", "&lt;", "g").replace(" ", "&
    nbsp;", "g")
    board.innerHTML += s + "<br>"
    board.scrollTop = output.scrollHeight
}

var send_message = function() {
    var input = document.getElementById("input")

    print_output(nickname + ": " + input.value)
    client.privmsg(channel, input.value)
}
```

LISTING 4.6: Orbited IRC JavaScript

4.6 Events

One of the main problems in complex Web Applications such as this is that the user tends to get lost and never really enjoy all the features the system offers. In fact, in many cases it seems that the more useful a component is, the less the user uses it - or even knows about it. In a game, this can be a big problem, because if the user doesn't know how to reach a certain functionality the game experience will be less interesting; sometimes those functionalities are very important or even essential to progress throughout the game and when the user doesn't find them or takes too long to understand how to reach them the interest in the game is lost.

4.6.1 Requirements

In this event-driven simulation environment, it's important that the user gets to know, in real-time, when something important happens - and from that point on he can move there and check the event with more detail. The fundamental requirement for this component is that it should be always visible, in the same place. That should help the users to follow the activity with no effort at all. The main events that should be alerted to the user are:

- When a user receives a message from another user (a user-related event)
- When someone adds him as friend (a user-related event)
- When a puppy has a problem (a work-related event)
- When he's being attacked (a war-based event)
- When their society has a news to deliver (a society-based event)

4.6.2 Architecture

Using `sfCometPlugin` (Section 3.3), we're able to integrate a event system easily. Each user has a private channel where all the events that related to him are published. When something important happens, these events are handled by the event tracker and immediately displayed to the user, allowing him to browse through all of those sections easily and when there's a real urgent need to do it.

4.6.3 Implementation

Creating a private channel for each user is the most important part. After initializing Meteor, as you can see in Listing 4.7, the user joins automatically his own channel, which is generated by the server, combining a random factor with his `user id`, described in Listing 4.8.

```
$(document).ready(function() {
  initializeMeteor();
  Activity.initialize();
});

var Activity = new function() {
  this.initialize = function() {
    try {
      Meteor.joinChannel('<?php echo $sf_user->getUser()->getChannel() ?>', 1);
      Meteor.connect();
    } catch(error) {
    }
  }

  this.activity = function(data) {
    $('#events p').remove();
    $('#events').append('<p display="hidden">' + data.message + '</p>');
    $('#events p').fadeIn();
  }
}

// Meteor Prototype
function activity(data) {
  Activity.activity(data);
}
```

Listing 4.7: jQuery event component initialization

```
public function getChannel()
{
  return md5($this->getStripped() . sha1($this->getId()));
}
```

Listing 4.8: Users private channel method

Sending events to each user is done at the model level, after inserting the related objects into the database. That way, we can integrate our model with any other application, and the events are sent anyway. To send the messages to each user, we use the Meteor class, described in Section 3.3.1. In Listing 4.9 is the code related to "Add Friend" event. The `EventTable` is responsible for storing all user and society activity, related to all the important events in the system.

```
public function addFriend($friend)
{
  if ($this->isFriend($friend))
  {
    return false;
  }

  $relation = new Friend();

  $relation->setUserId($this->getId());
  $relation->setFriendId($friend->getId());

  $relation->save();

  EventTable::log(Activity::FRIENDS, '%1% added %2% from friends list',
                  array('%1%' => $this->getName(), '%2%' => $friend->getName()));

  return $relation;
}
```

Listing 4.9: Adding a Friend event

In the `Event` class, we override the `postInsert` function, that is always executed after object insertion in the database. As you can see in Listing 4.10, we make use of `Meteor::addMessage` method that sends the event automatically to the user related with it.

```
public function postInsert($event)
{
  $message = array('method'   => 'activity',
                   'channel'  => $this->getUser()->getChannel(),
                   'username' => $this->getUser()->getStripped(),
                   'message'  => $this->getMessage());

  Meteor::addMessage($this->getPimp()->getChannel(), addslashes(json_encode($message)));

  parent::postInsert($event);
}
```

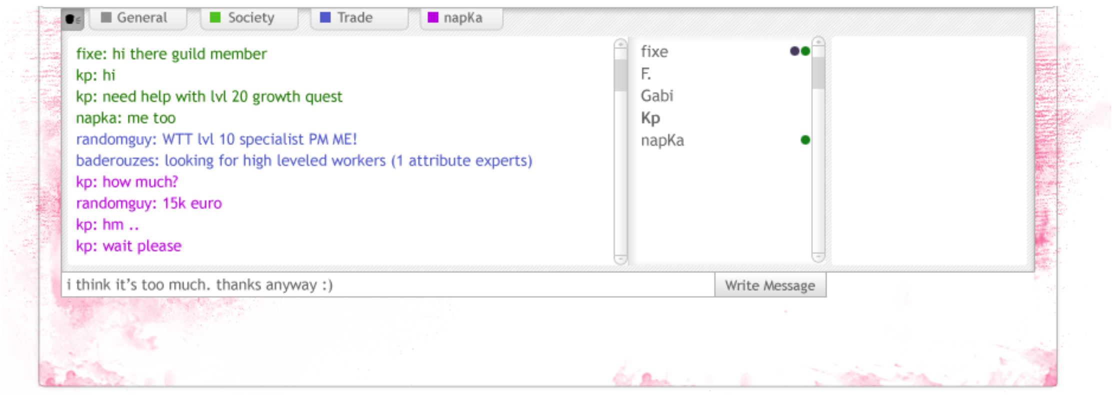
Listing 4.10: Event postInsert

4.7 Interface

The Chat interface supports various channels, as seen on Figure 4.2. There is a general channel for global chat, where all the players can communicate, a trade channel for players who wish to trade objects, a society channel where only members of the society can intervene, and private channels for communication between players.

The Event interface is integrated in a very simple way, as seen in Figure 4.3. With this integration, the player doesn't have to be visually disturbed anytime an event occurs. If he wants to check for other events, he can go to his *dashboard* and check the Events widget (Figure 4.4)

The user *dashboard* can be seen in Figure 4.5. It is the most important part of `simX` , combining the developed components within a coherent whole and focusing on the attributes of the player.

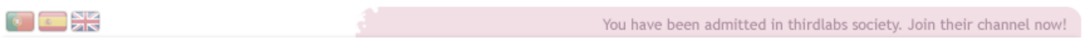


fixe: hi there guild member
kp: hi
kp: need help with lvl 20 growth quest
napka: me too
randomguy: WTT lvl 10 specialist PM ME!
baderouzes: looking for high leveled workers (1 attribute experts)
kp: how much?
randomguy: 15k euro
kp: hm ..
kp: wait please

fixe
F.
Gabi
Kp
napKa

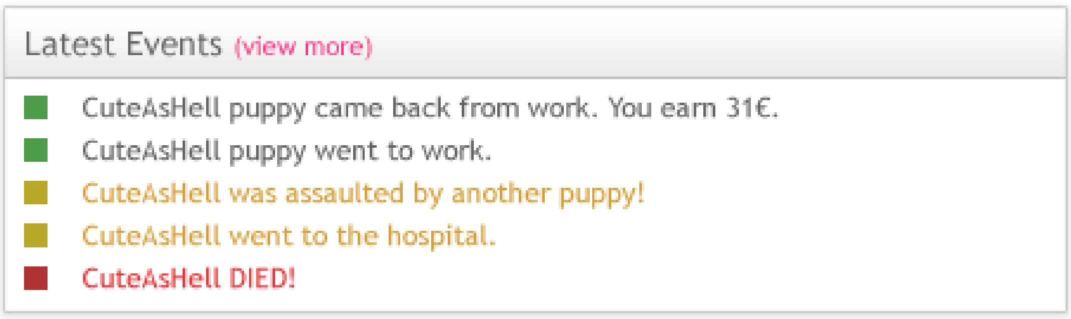
i think it's too much. thanks anyway :)

FIGURE 4.2: The Chat component being used in simX



You have been admitted in thirdlabs society. Join their channel now!

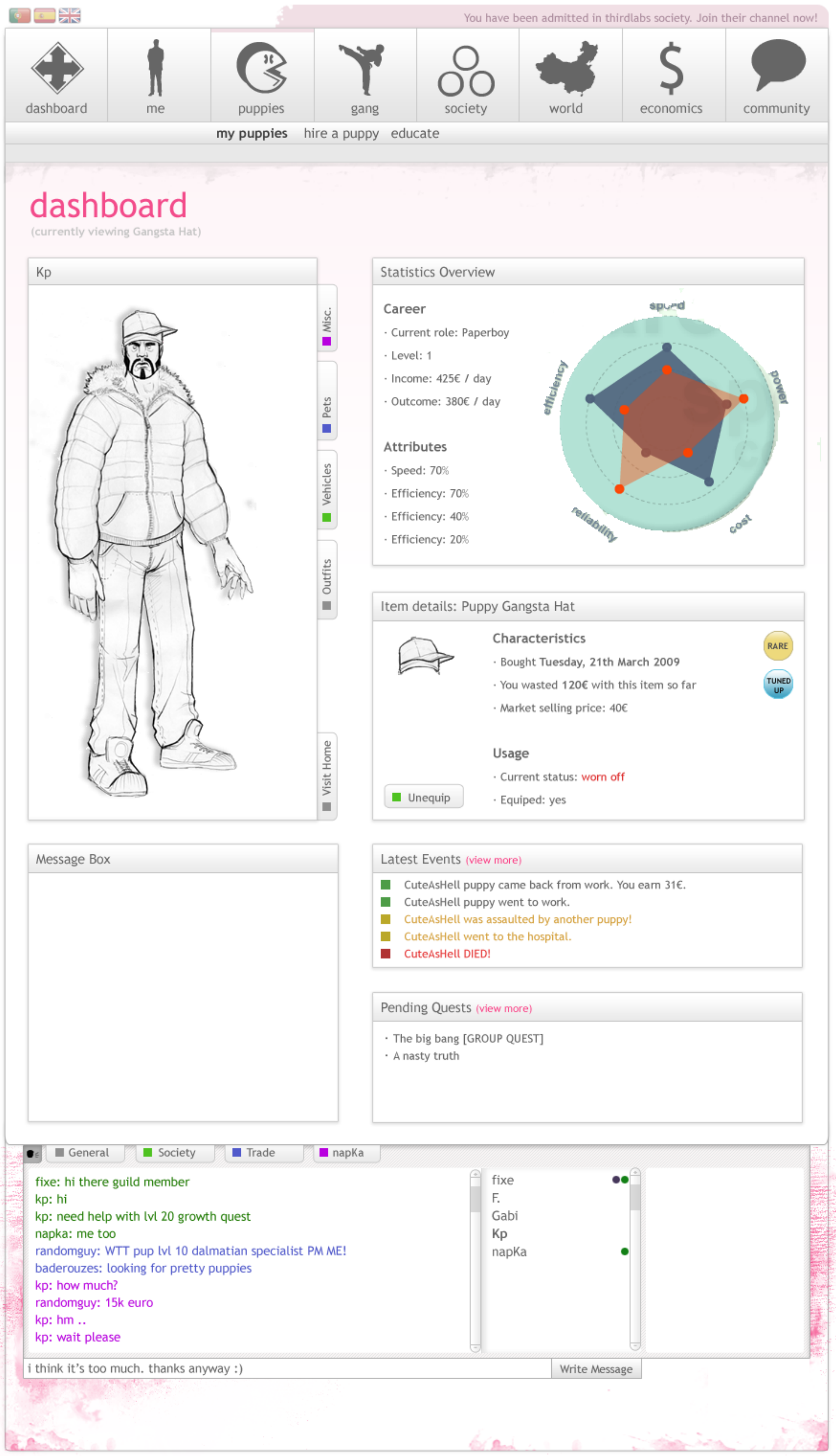
FIGURE 4.3: The Event component being used in simX



Latest Events (view more)

🟩 CuteAsHell puppy came back from work. You earn 31€.
🟩 CuteAsHell puppy went to work.
🟨 CuteAsHell was assaulted by another puppy!
🟨 CuteAsHell went to the hospital.
🟥 CuteAsHell DIED!

FIGURE 4.4: The Event widget in simX user dashboard



You have been admitted in thirdlabs society. Join their channel now!

dashboard
(currently viewing Gangsta Hat)

Kp

Statistics Overview

Career
· Current role: Paperboy
· Level: 1
· Income: 425€ / day
· Outcome: 380€ / day

Attributes
· Speed: 70%
· Efficiency: 70%
· Efficiency: 40%
· Efficiency: 20%

Item details: Puppy Gangsta Hat

Characteristics
· Bought Tuesday, 21th March 2009
· You wasted 120€ with this item so far
· Market selling price: 40€

Usage
· Current status: worn off
· Equiped: yes

Message Box

Latest Events (view more)
- 🟩 CuteAsHell puppy came back from work. You earn 31€.
- 🟩 CuteAsHell puppy went to work.
- 🟨 CuteAsHell was assaulted by another puppy!
- 🟨 CuteAsHell went to the hospital.
- 🟥 CuteAsHell DIED!

Pending Quests (view more)
· The big bang [GROUP QUEST]
· A nasty truth

fixe: hi there guild member
kp: hi
kp: need help with lvl 20 growth quest
napka: me too
randomguy: WTT pup lvl 10 dalmatian specialist PM ME!
baderouzes: looking for pretty puppies
kp: how much?
randomguy: 15k euro
kp: hm ..
kp: wait please

fixe
F.
Gabi
Kp
napKa

i think it's too much. thanks anyway :)

FIGURE 4.5: simX Dashboard

Chapter 5

Conclusions

This chapter presents an overview on the work developed in this thesis, aimed at simplifying the process of bringing real-time user interaction to Web Applications. The results achieved at both the architectural level and the case study implementation are also discussed, together with plans for future development and the author's personal remarks.

5.1 Overview

The variety of Web Applications available nowadays is enormous. Although the potential for improvements on the user interface level has grown immensely with the widespread use of AJAX, there is still ample room for improvement, specially as applications evolve to include interaction and feedback between users. Currently, the main challenge lies in providing information to users in real time, coherently across all browsers and mobile devices, in a way that is independent from the activity they might be performing on the application.

Several techniques - collectively dubbed Reverse AJAX - attempt to fill in this blank spot, with Comet being the most promising. As many in the community recognize, the issue is that Comet is currently a moving target, and the landscape is expected to change significatively over the next months. Any Comet implementation that claims to have a stable and mature architecture probably just has a stalled development effort. This simply speaks to the fact that Comet is still a very new and relatively unknown technology.

The immediate effect for developers attempting to integrate Comet into their applications is that there is really no reference implementations, no well-known practices, no established patterns and no significant efforts to make the technology accessible. Anyone intending to develop even a small application making use of Comet not only has to consult and digest a large amount of sources, but also deal with the often conflicting information.

This thesis attempts to fill in this gap by providing not only a reference implementation and development patterns, but also a use case that leverages Comet technology to design and build sample components that can be used in other applications, in order to provide better user interaction.

5.2 Results

The study of AJAX and Reverse AJAX alternatives undertaken in the first two chapters provide a comprehensive introduction to the issues at hand and a realistic analysis of the state-of-the-art. This alone is relevant for anyone trying to grasp the background, concepts and options available and works towards the goals of this thesis, more specifically that of lowering the entry barriers and making these technologies accessible to the average developer.

The architectural solution developed provides a good reference implementation for integrating Comet technology in any Web Application. The `sfCometPlugin` is not only ready for production usage but is in fact already functioning in other projects.

Additionally, the `Events` component is an innovation not only in concept, but also in its simple and effective implementation, which meets its purpose of easy integration into any Web Application. Together with the `Chat` component, they provide two solid building blocks for more complex applications.

On the downside, this thesis could have provided an analysis on the actual impact that the developed components have in the user interaction, but the game hasn't yet been released to do such a study.

Finally, the `simX` project is on its way to become a good example on how to leverage this sort of technology, and the initial feedback from the first few beta testers is as exciting as it is rewarding.

5.3 Future Work

Comet is itself under heavy development, so it is only natural that the solutions presented in this thesis will need to be constantly adapted to meet the advancements and any possible changes in paradigm that will surely arise.

The `sfCometPlugin` - which currently works with Meteor and Orbited - can be improved to work with other Comet implementations. The Cometd implementation is of special interest due to its large and active community, and work is already underway to integrate with it. Another avenue for improvement would be to implement the plugin for other frameworks, particularly the widely used Django and Ruby on Rails.

Improvements will come naturally to the the `Events` and `Chat` components - developed for the use case - as they are adapted to work in other applications and gain a wider API. One area of development already underway is the addition of private channels with different types of encryption. This will allow users to have confidence in the security of their data.

As for the `simX` project, although the basics are in place and it already offers some degree of functionality, it still requires heavy work in order to fully implement the level of simulation that is its ultimate goal. One sub-project already underway is the development of a system that implements world-class simulation events, independent from the Web Application, delivered as Comet events. Another interesting idea is the creation of an application that interfaces with external RSS feeds and delivers real-world news as events into the system.

Last but not the least, after discussing this thesis with Michael Carter - Orbited's author - work is underway to synthetize a HOW-TO document describing the solutions and techniques described herein, for publishing in Orbited's documentation section.

5.4 Personal Remarks

Web 2.0 is about bringing people together - connecting and communicating [46]. Personally, I would say that the next step could easily be called Web 2.1, which is all about bringing people together in a way they can actually interact in a natural and significant way.

Working on this thesis has been a challenge of its own. It is one thing to learn about different technologies, but it is an entirely different matter to assemble that knowledge for others to learn from it. Great effort was placed into finding the right balance between completeness and size, with the hope that readers wouldn't be bored and leave this work gathering dust in a shelf. Also, a lot of time was spent struggling with the right way to express concepts, methods and attempting to explain things in a progressive way. Overall, I feel that this process has not only improved my english-expression skills, but has also contributed towards increasing my participation in several online communities.

On a more technical side, it was a great joy to learn in such depth about a technology that I truly believe in and which I honestly believe will push the Web forward into the future. This work has given me the opportunity to explore best practices, gather knowledge and build work that is of great help in my current activity as a Systems Engineer at *seegno*, a company that I recently co-founded, with a personal mission of creating innovative work in Web Application development and a vision of a world where the Web is the ultimate platform for user interaction.

It is also with great pride that we will be integrating this technology into *e-sharing*, a project that was initially born at this University as the humble *sharingLESI* and which has gone a long way since its first inception. Given its emphasis on collaboration and user interaction, it offers the perfect ground for improving an existing Web Application with this real-time user interaction, which unsurprisingly, is exactly what this thesis was all about.

Bibliography

[1] Ali Mesbah and Arie van Deursen. A Component - and Push-based Architectural Style for AJAX Applications. Technical report, Delft University of Technology, 2008.

[2] Sun. Accelerating Your Business To Web Speed. Technical report, Sun Microsystems, September 2006.

[3] Tim O'Reilly. What Is Web 2.0, September 2005. URL `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`.

[4] Tim O'Reilly. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. *Social Science Research Network Working Paper Series*. URL `http://ssrn.com/abstract=1008839`.

[5] Cindy Lu. The Impact of AJAX on User Experience, October 2006. URL `http://www.apogeehk.com/articles/AjaxUserExperienceStrategiesPart1.html`.

[6] Jesse James Garrett. AJAX: A New Approach to Web Applications, February 2005. URL `http://www.adaptivepath.com/ideas/essays/archives/000385.php`.

[7] Juergen Haas. AJAX - Paradigm For Responsive And Interactive Web Pages, August 2007. URL `http://linux.about.com/b/2007/08/02/ajax-paradigm-for-responsive-and-interactive-web-pages.htm`.

[8] Nicholas C. Zakas, Jeremy McPeak, and Joe Fawcett. *Professional AJAX, 2nd edition*. Wiley Publishing, 2007.

[9] Katherine Martin. Developing Applications Using Reverse AJAX, March 2007. URL `http://today.java.net/pub/a/today/2007/03/22/developing-applications-using-reverse-ajax.html`.

[10] Greg Wilkins. Why AJAX Comet? Technical report, Webtide, June 2006. URL `http://www.webtide.com/downloads/whitePaperWhyAjax.html`.

[11] Linda Dailey Paulson. Building Rich Web Applications with AJAX. *Computer*, 38(10):14–17, 2005.

[12] Themistoklis Palpanas and Balachander Krishnamurthy. Reducing retrieval latencies in the Web: the past, the present, and the future. Technical report, University of Toronto, 1999.

[13] Chrisina Draganova. Asynchronous JavaScript Technology and XML (AJAX). Technical report, London Metropolitan University, 2007.

[14] Coach Wei and Rob Gonda. *A Brief History of AJAX*. SYS-CON, 2008.

[15] Coach K. Wei. AJAX: Asynchronous Java + XML?, August 2005. URL `http://www.intranetjournal.com/articles/200508/ij_08_23_05a.html`.

[16] AjaxWith. Comet: The Next Stage for AJAX, September 2008. URL `http://www.ajaxwith.com/Comet-The-Next-Stage-for-Ajax.html`.

[17] Garrett Dimon. Front-End Architecture: AJAX and DOM Scripting, May 2006. URL `http://v1.garrettdimon.com/archives/front-end-architecture-ajax-dom-scripting`.

[18] Frank Zammetti. *Practical DWR 2 Projects*. Apress, 2008.

[19] Exadel. Ajax On-Demand. White Paper, 2007.

[20] Engin Bozdag. Push solutions for AJAX Technology. Master's thesis, Delft University of Technology, 2007.

[21] IBM. Websphere Application Server, 2007. URL `http://www.ibm.com/software/webservers/appserv/was`.

[22] Mortbay Consulting. Jetty 6 Architecture, 2006. URL `http://docs.codehaus.org/display/JETTY/Architecture`.

[23] Steve Champeon. JavaScript: How did we get here?, 2001. URL `http://www.oreillynet.com/pub/a/javascript/2001/04/06/js_history.html`.

[24] Ian Hickson and David Hyatt. HTML 5 - A vocabulary and associated APIs for HTML and XHTML, October 2008. URL `http://www.w3.org/html/wg/html5`.

[25] Michael Carter. Independence Day: HTML5 WebSocket Liberates Comet From Hacks, July 2008.

[26] Ian Hickson. WHATWG Web Applications 1.0 specification, October 2008.

[27] Arve Bersvendsen. Event Streaming to Web Browsers, September 2006.

[28] Engin Bozdag, Ali Mesbah, and Arie van Deursen. A Comparison of Push and Pull Techniques for AJAX. Technical report, Delft University of Technology, January 2007.

[29] Michael Carter. The State of Comet, 2008. URL `http://orbited.org/svn/orbited/branches/0.5/presentations/myspace/slides_outline`.

[30] Garey. Software Portability: Weighing Options, Making Choices. *The CPA Journal 77*, 2007.

[31] Chuck Crandall. JavaScript Toolkit Comparison, September 2006. URL `http://www.ja-sig.org/wiki/display/UP3/Javascript+Toolkit+Comparison`.

[32] Peter Wayner. Inside open source AJAX Toolkits, March 2008. URL `http://www.infoworld.com/article/08/03/03/10TC-open-source-ajax_1.html`.

[33] Daniel Carrera. Choosing an AJAX Toolkit, February 2008. URL `http://daniel.carrera.name/2008/02/08/choosing-an-ajax-toolkit`.

[34] Daniel Carrera. Choosing an AJAX Toolkit - Part ii - jQuery, March 2008. URL `http://daniel.carrera.name/2008/03/01/choosing-an-ajax-toolkit-part-ii-jquery`.

[35] CometDaily. Comet Maturity Guide, May 2008. URL `http://cometdaily.com/maturity.html`.

[36] CometDaily. Comet Gazing: Maturity, March 2008. URL `http://cometdaily.com/2008/03/14/comet-gazing-maturity`.

[37] Greg Wilkins. Cometd with Jetty, August 2006.

[38] P. Th. Eugster, P. A. Felber, R. Guerraoui, and A. m. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35:114–131, 2003.

[39] Adrian Weisberg. Integrating Orbited with Web App Frameworks, September 2008. URL `http://orbited.org/blog/2008/09/integrating-orbited-with-web-app-frameworks/`.

[40] Adrian Weisberg. Integrating orbited with web app frameworks, September 2008. URL `http://orbited.org/blog/2008/09/integrating-orbited-with-web-app-frameworks/`.

[41] T. J. Community Interaction, June 2007. URL `http://www.galaxy-news.net/content/71_community_interaction.html`.

[42] Fabien Potencier and François Zaninotto. *The Definitive Guide to symfony.* Apress, 2007.

[43] Jeremy Keith. Hijax, January 2006. URL `http://domscripting.com/blog/display/41`.

[44] Michael Mahemoff. Hijax: Graceful Degration, January 2006.

[45] David Kuhn. Ever wondered how Gmail/Facebook chat works?, May 2008. URL `http://technophiliac.wordpress.com/2008/05/28/ever-wondered-how-gmailfacebook-chat-works`.

[46] Adam Owen. Web 2.0 Thoughts and Suggestions, February 2008. URL `www.adamowen.com/blog/2008/02/18/4`.

Appendix A

Code

```
User:
  actAs:
    Timestampable:
      options: []

  columns:
    username:
      type: string(255)
      unique: true
    name:
      type: string(255)
    level:
      type: integer
    popularity:
      type: decimal
      default: 0
    power:
      type: decimal
      default: 0
    leadership:
      type: decimal
      default: 0
    strength:
      type: decimal
      default: 0
    money:
      type: decimal
      default: 0
```

Listing A.1: Database schema for the User model

```
Chat:
  actAs:
    Timestampable:
      options: []

  columns:
    user_id:
      type: integer(11)
    channel:
      type: string(255)
    message:
      type: clob

  relations:
    User:
      class: User
      local: user_id
      foreignAlias: Chats
      foreignType: one
      onDelete: CASCADE
```

Listing A.2: Database schema for the Chat model

```
<?php use_helper('Form') ?>

<div id="chat">
  <ul id="options">
    <li class="tab general" title="Change to world chat">
      <div class="left"></div>
      <?php echo jq_link_to_function('G', 'Chat.channel("general")') ?>
      <div class="right"></div>
    </li>

    <li class="tab trade" title="Change to trade chat">
      <div class="left"></div>
      <?php echo jq_link_to_function('T', 'Chat.channel("trade")') ?>
      <div class="right"></div>
    </li>

    <?php if ($sf_user->getUser()->getSociety()): ?>
    <li class="tab society" title="Change to society chat">
      <div class="left"></div>
      <?php echo jq_link_to_function('S', "Chat.channel('$societyChannel')") ?>
      <div class="right"></div>
    </li>
    <?php endif ?>
    <li class="tab special">
      <div class="left"></div>
      <?php echo jq_link_to_function('Leave', 'Chat.quit()') ?>
      <div class="right"></div>
    </li>

    <li class="tab special">
      <div class="left"></div>
      <?php echo jq_link_to_function('Clear', 'Chat.clear()') ?>
      <div class="right"></div>
    </li>
  </ul>

  <div id="divisions">
    <div id="talk">
      <div id="general" style="display: none"></div>
      <div id="trade" style="display: none"></div>
      <div id="<?php echo $sf_user->getUser()->getSociety()->getChannel() ?>"
    style="display: none"></div>
    </div>

    <div id="list"></div>
  </div>

  <div id="sentence">
    <?php echo jq_form_remote_tag(array('url'     => 'chat/message',
                                        'loading' => '$("#talkbox").val("")')) ?>

      <?php echo input_tag('talkbox', '', array('class' => 'textbox',
                                                'autocomplete' => 'off')) ?>
      <?php echo input_hidden_tag('channel', '') ?>

      <?php echo submit_tag('', array('title' => 'Say it in the chat window',
                                      'value' => 'Say it',
                                      'class' => 'button')) ?>
    </form>
  </div>
</div>
```

Listing A.3: Chat component template

Appendix B

Configuration Files

```
MaxTime 240
PingInterval 3
Debug 0
SyslogFacility none

[iframe]
HeaderTemplate HTTP/1.1 ~status~\r\nServer: ~server~\r\nContent-Type: text/html;
    charset=utf-8\r\nPragma: no-cache\r\nCache-Control: no-cache, no-store, must-
    revalidate\r\nExpires: Thu, 1 Jan 1970 00:00:00 GMT\r\n\r\n<html><head><meta
    http-equiv="Content-Type" content="text/html; charset=utf-8">\r\n<meta http-
    equiv="Cache-Control" content="no-store">\r\n<meta http-equiv="Cache-Control"
     content="no-cache">\r\n<meta http-equiv="Pragma" content="no-cache">\r\n<
    meta http-equiv="Expires" content="Thu, 1 Jan 1970 00:00:00 GMT">\r\n<script
    type="text/javascript">\r\nwindow.onError = null;\r\nvar domainparts =
    document.domain.split(".");\r\ndocument.domain = domainparts[domainparts.
    length-2]+"."+domainparts[domainparts.length-1];\r\nparent.Meteor.register(
    this);\r\n</script>\r\n</head>\r\n<body onload="try { parent.Meteor.reset(
    this) } catch (e) {}">\r\n~channelinfo~\r\n
Persist 1

[xhrinteractive]
HeaderTemplate HTTP/1.1 ~status~\r\nServer: ~server~\r\nContent-Type: text/html;
    charset=utf-8\r\nPragma: no-cache\r\nCache-Control: no-cache, no-store, must-
    revalidate\r\nExpires: Thu, 1 Jan 1970 00:00:00 GMT\r\n\r\n
    .......................\r\n~channelinfo~\r\n
Persist 1

...
```

Listing B.1: Meteor configuration file

```
[global]
reactor=select
# reactor=kqueue
# reactor=epoll
proxy.enabled = 1
proxy.keepalive = 0
session.ping_interval = 40
session.ping_timeout = 30

[listen]
http://:8500
# uncomment to enable ssl on port 8043 using given .key and .crt files
#https://:8043
#
#[ssl]
#key=orbited.key
#crt=orbited.crt

[static]

[access]
* -> localhost:61613
localhost:8000 -> irc.domain.com:6667

[logging]
debug=STDERR,debug.log
info=STDERR,info.log
access=STDERR,info.log
warn=STDERR,error.log
error=STDERR,error.log

#Don't enable debug by default
enabled.default=info,access,warn,error

# Turn debug on for the "Proxy" logger
[loggers]
#Proxy=debug,info,access,warn,error
```

Listing B.2: Orbited configuration file