BSc (Hons) Computing

Final year project

Cross-Browser Framework for Server-Push over HTTP

Adam Gritt

Supervisor: Michael Jones

1st June 2009

Abstract

The HTTP protocol presents a fundamental obstacle in the development of real-time web applications. This project seeks to overcome and abstract that limitation from the developer.

Through a process of iterative development, a browser-agnostic framework was created which not only succeeds in this goal, but improves over existing implementations. Server-push over HTTP is transparently enabled through use of the framework.

A series of applications were developed demonstrating capabilities of the framework, and these were succesfully tested and evaluated by users.

Ultimately, the fundamental limitation of HTTP's client-pull model was overcome, though at a very small latency penalty compared to traditional desktop applications.

Copyright

Signed:

Date:

The author can be contacted at arcgritt@gmail.com

Acknowledgements

Mike Jones for his invaluable guidance throughout the project

All of the DEC lecturers for helping and teaching me over the last four years

Questionnaire participants for selflessly donating your time

Mum and Dad for being so supportive of me and always believing in what I could accomplish

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Web as an Application Platform

Over the past decade the web browser has evolved from being a simple remote document viewer into a fully fledged application platform. Modern browsers support audio, animation, scripting and dynamic loading as standard, with video soon to be included in the formal specification (Hickson & Hyatt, 2009, section 4.8.7). The ubiquity of the browser has led to the increasing adoption of rich internet applications (RIAs); applications which offer the responsiveness, features and functionality approaching that of desktop applications (Chaganti, 2007, p1; Deitel & Deitel, 2008, p32). Raggett (2005) goes as far as to say that developers are "deserting" Windows in favour of the Web.

The client-server architecture of the web provides the advantages of scalability, security and centralisation (Reiff, n.d., p1-3). The high-level language used (HTML), facilitates the rapid development of complex applications at a very low cost compared to ad-hoc architectures. Few, if any, modifications are required to client software, expediting easy deployment over large networks. Deitel & Deitel (2008, p4) attribute the rise in RIAs to improvements in hardware, internet speed and business models that are easy to transfer to these applications. Updates to the software take place on the server and automatically affect every user (Deitel & Deitel, 2008, p30). Due to the highly competitive nature of the Web, client software is readily and freely available from a multitude of organisations.

1.2 HTTP Push

Despite its many benefits, the Web, by design, has a barrier preventing certain applications from migrating to its architecture. The HTTP protocol (Fielding *et al.*, 1999) permits information to be sent from the server only upon request; this is known as client-*pull*. Real-time applications such as instant messaging require server-*push*, allowing the server to send information as and when it becomes available. Fortunately, in the very least an emulation of server-push can be accomplished using client polling; where the client periodically checks the server for new data. HTTP is a stateless protocol, meaning information does not persist between requests (Fielding *et al.*, 1999). This conflicts with the fundamental principle of RIAs, which retrieve information asynchronously and intermittently.

1.3 Cross-Browser Compatibility

The average internet user now spends over 17 hours per week online, whilst heavy users average 42 hours (Cole, 2009, p4). Users don't need to learn how to use a browser when visiting a new site (Raggett, 2005, slide 5), usually ignoring a website rather than update or change software (Heller, 2009). Likewise, "more than half of users prompted to upgrade a plug-in upon visiting a site" abandon their visit (Smith, 2007). For these reasons, it is extremely important for websites to function correctly without requiring user action.

Various RIA platforms exist, each with varying market penetration and compatibility *(see appendix A)*. Adobe Flash and ECMA JavaScript are clearly the most prevalent. Flash can claim a market penetration of over 95%, but only 51% of users have the latest version (riastats.com, 2009). Similarly, JavaScript has near-universal native support, but implementations differ between browsers. The first problem, therefore, is the ability of a system to reach a full audience; cross-browser compatibility is required in order to exploit the web's dominance.

1.4 Objectives

Code reuse is a fundamental aspect of modern software development; it can reduce costs by an order of magnitude, but more importantly reduces the development time and duration of a project (McConnell, 1996, p537). Software frameworks (hereafter simply 'frameworks') are the pinnacle of code reuse and provide specific functionality

which can be utilised by a developer. Contrasting with the role of libraries, it is the framework which dictates the control flow of the program rather than the caller (Riehle, 2000, p72). If implemented well, frameworks can be improved and changed without necessitating change to the applications themselves. The system will be developed with code reuse in mind and if the architecture permits it, will take the form of a framework.

The freedom of bi-directional data transfer could allow nearly any business model or desktop application to be adapted to a web architecture. This project will research and evaluate the most suitable software for this purpose, the primary objectives being:

Server-push over HTTP - the ultimate goal and purpose for which this project is being undertaken

Cross-browser compatibility must be achieved in order to exploit the web's most valuable asset

Code reuse will be a central concern and the extent to which this is achieved will measure the quality of the project

Chapter 2

Background Research

2.1 Client Software

It is good practice to create content which can reach the widest possible audience, irrespective of hardware or software (Jacobs & Walsh, 2004, s4.3). Today, in an increasingly competitive market, the dozen most popular clients constitute 98% market share, whilst a single client holds no more than 45% (NetApplications, 2009). Past experience using JavaScript during industrial placement has spurred the author's search for a different approach to web application development. Custom-written JavaScript, whilst lightweight and efficient, is extremely time-consuming to write and is therefore impractical for a project of this scope within the timescale. Though JavaScript is a native component of all major browsers, implementations and dialects differ and even standards-compliant pages do not behave uniformly between clients (Snook *et al.*, 2007). Large websites, such as BBC News, MSN and Wikipedia can afford to create separate, but identical, implementions of their website to suit differing clients. However, for many organisations this cost is too high and the intricacies and quirks of individual clients must be abstracted from the developer. JavaScript frameworks and third-party plugins have arisen to provide this layer of abstraction.

2.1.1 JavaScript Frameworks

Among the JavaScript solutions are Google Web Toolkit (GWT), Dojo and jQuery. Dojo and jQuery are both libraries which simply abstract the different browser implementations from the programmer; code is written in JavaScript and deployed as normal, functions are dynamically chosen at runtime by the libraries themselves. The primary advantage is that a single implementation will work in all supported clients.

They also implement common 'utility' functions which do not exist in the Document Object Model (DOM) API (Harmon, 2008, p26), and include complex widgets such as text-editors (Harmon, 2008, p55). These libraries have become increasingly relevant as, due to browser compatibility improvements, their size decreases and performance improves (Snook *et al.*, 2007, p1-2).

GWT takes an entirely different approach whereby code is written in Java, then compiled to a separate JavaScript implementation for each DOM (Fain *et al.*, 2007, p10). There are many advantages to this; Java's polymorphism simplifies a process which, in plain JavaScript, is overly complex and its strong typing eliminates errors which would otherwise be hidden until runtime. The GWT compiler provides optimized, compressed and obfuscated code for each output implementation (Burnette, 2006, p15). A disadvantage to GWT's method is that if a new DOM is developed (such as recently, Google's own Chrome browser), an updated GWT compiler would be required; intricacies exploited by GWT are likely to prevent graceful degradation. Importantly, GWT is designed to facilitate integration between server-side Java ('servlets') and client side Java (later, transparently compiled to JavaScript) - code can be shared between the client and server (Burnette, 2006, p3).

2.1.2 Plugins

Browser plugins provide near-perfect cross-platform compatibility, however the plugins themselves are not available for all platforms (notably mobile phones and linux64 (Adobe, 2009)), or have poor market penetration *(see appendix A)*. In some environments such as intranets, this could be a minor issue, however client independence is always desirable and in the case of public websites, required. The sole plugin that meets this criteria is Adobe Flash Player which rivals even JavaScript in its ubiquity. Flash, originally used to provide animation and multimedia capabilities (Gay, n.d., p4), has evolved into a full RIA platform with the introduction of Adobe Flex. Flex utilises Flash on the client side, inheriting its market penetration, and is designed for integration with many server platforms including PHP and Java (Fain *et al.*, 2007, p5).

2.2 Questionnaire

The overall objective of the questionnaire will be to determine any usability issues which exist with various RIA platforms. The web was not originally intended for RIAs, and as such there are certain limitations with how they interact and integrate

with existing systems. For example, many RIAs have bookmarking issues because different pages are accessed through the same URI; this issue is typically overcome using a "hack" involving hidden `iframe` and `anchor` HTML elements.

2.2.1 Construction

The questionnaire will be divided into five distinct sections; web browsing, instant messaging, browser games, other browser applications and demographics. Due to time constraints it will not be possible to use stratified sampling, therefore bias may be introduced into the results (Nicholls *et al.*, 2006). For this reason, several control questions will be asked which already have substantiative and unbiased results in the public domain. The accuracy of these results will then be used as a basis to evaluate the questionnaire.

All questions will be closed-ended in order to illicit the highest number of responses to the questionnaire. Closed-ended questions better communicate the frame of reference to respondents and have greater specificity (Converse & Presser, 1986, p34-5). Likert scales will be used to ascertain user preference towards particular technologies.

The questionnaire was sent to students studying Batchelors and Masters degrees at The Business School at Bournemouth University. This target group was chosen due to a likelihood of having experience with the subject matter.

2.2.2 Results

There were 78 responses to the questionnaire.

Context

Approximately 80% of respondents said they spend over an hour a day using a browser and more than half of these said over three hours. This corresponds with a survey by Cole (2009) stating an average of 2.5 hours per day. Over 90% of respondents were between 18 and 23 years of age, whilst two thirds were female. Data mining was performed on the result set in order to detect whether this introduced bias. Link analysis was performed based on gender, and resulted in a kappa coefficient of 0.1649 *(see appendix C)*; the measure of agreement against what could be expected simply by chance. This is extremely low, and even then is probably attributed to overfitting; this very strongly indicates that the gender disparity introduced little to no bias in results.

Analysis

Corresponding with the results of Heller (2009) and Smith (2007), study participants generally agreed that they ignore sites that do not function correctly in their browser *(see table B.4)*. Participants generally disliked installing third-party plugins and felt they had no choice about installing them *(see tables B.6 and B.7)*. There was a strong preference towards Ajax websites rather than Flash or plain HTML, however a third did not know which type they preferred *(B.8)*. This indicates the transparency of the web browser as an application platform and the seamless transition to 'Web 2.0'.

Desktop-based instant messengers were rated significantly better than browser based clients *(B.10 & B.11)*. This may be related to participants' dislike of keeping their browser open *(B.14)*. Participants showed no preference about using browser-based messengers when desktop equivalents were available *(B.13)*, though there were a diverse range of answers to that question. This combination of results indicates that respondents do find a unique utility in browser-based messengers.

Whilst 71.79% of respondents had played a browser-based game, just a third had ever played a multiplayer browser game *(B.15 & B.18)*. Multiplayer games were conclusively rated better than their counterparts *(B.16 & B.19)*.

Just a sixth of respondents said that they used other browser-based applications regularly *(B.21)*, though they agreed the performance of them was generally good *(B.22)*. Asked if there were features missing from these applications, results were somewhat ambivalent; however, this would tend to indicate that it is neither performance nor features which deters users. Perhaps there are simply no advantages to these applications, as previously mentioned users do not like to keep their browser open; security and privacy concerns may also play a part.

Whilst providing useful data, the Likert scale reponses showed very few users choosing the extreme responses. This is a typical result of many questionnaires due to psychology (Nicholls *et al.*, 2006) and is referred to as *error of central tendency* (Oppenheim, 1992, p233).

2.3 Existing Software Evaluation

Several examples of real-time browser applications were examined to ensure proof-of-concept and discover the most efficient technique. Firebug, a web development plugin for Firefox, was used to examine the networking architecture of these applications.

2.3.1 MSN Web Messenger

The web version Microsoft's popular instant messaging client interconnects seamlessly with the desktop version. The architecture consists of a single parent window containing the contact list, which spawns a child window for each user conversation. These child windows are programmatically linked in JavaScript and can communicate; all receive events are processed by the parent window and then distributed locally to the appropriate child. Send events are handled independently by child windows.

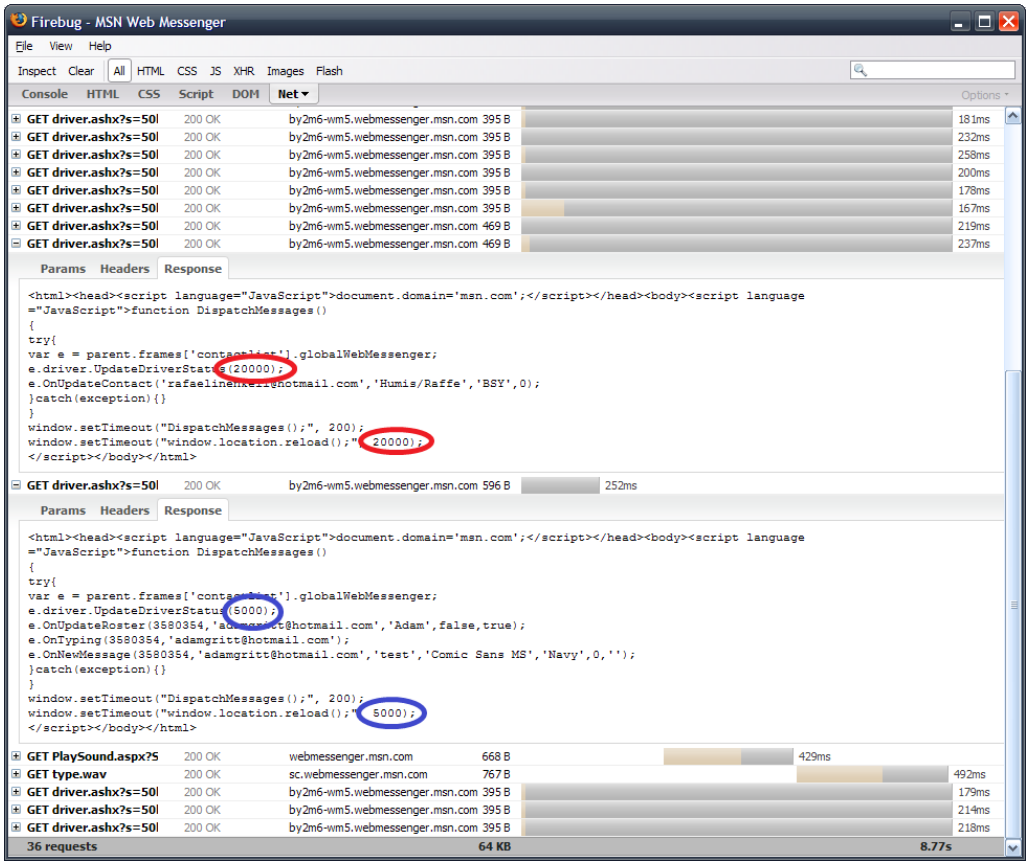


Figure 2.1: MSN Web Messenger network activity
Red: the standard polling rate of 20 seconds is continued
Blue: the polling rate is changed to 5 seconds because new events are received

Data retrieval is accomplished using a dynamic polling rate mechanism *(see figure 2.1)*, and is thus a pull mechanism, not push. It isn't RESTful though, and the end result is the same. The client's parent window polls the server intermittently every 20 seconds to check for events such as contact sign-ins and messages. Should the event be a message, the server will instruct the client to increase its polling

rate to 5 second intervals. After 80 seconds of inactivity, the server instructs the polling rate to be decreased again to 20 second intervals. Instructions are received as HTML+JavaScript and are then executed by the client, and messages are sent by the client as standard name-value pairs using POST.

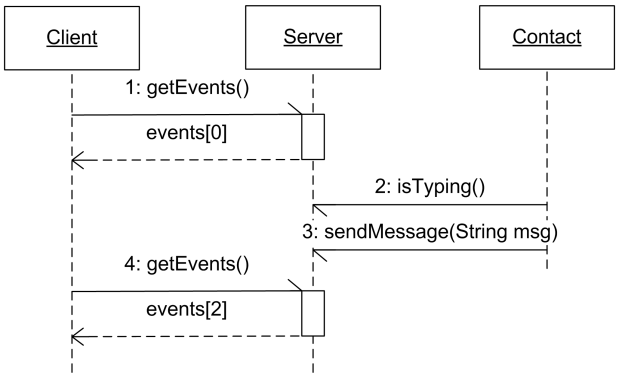


Figure 2.2: UML sequence diagram of MSN Web Messenger

This architecture is highly inefficient and exhibits a poor response time. In a worst case scenario, a message could take up to 20 seconds plus latency to receive. In a best case scenario, an average of 2.5 seconds plus latency. Additionally, client bandwidth is wasted by intermittent polling which takes place even when no events are available to the client *(see event 1 in figure 2.2)*.

2.3.2 Facebook Chat

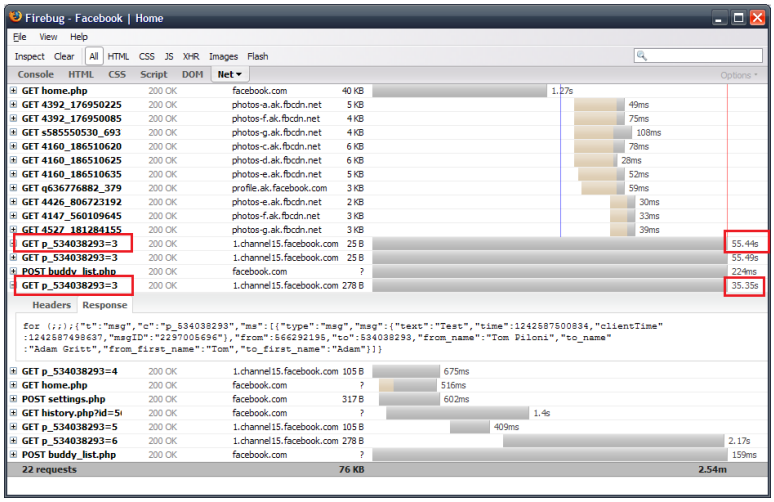


Figure 2.3: Facebook Chat network activity

Facebook's web messaging system uses a technique called long-polling, where client polls are held on the server until events need to be returned. Polls are held for up to 55 seconds, after which they are returned regardless of events *(see figure 2.3)*. Notice how the same request returned twice after 55 seconds, but once after just 35 seconds when a message was received. It is likely that Facebook's server architecture causes a timeout after 60 seconds for security reasons. In the Facebook system *(figure 2.4)*, the delay between events 3 and 7 should be little more than the client's latency. The contact's second message *(5)* is cached by the server until the client can send another request. The delay in a client receiving messages will be at most twice the client latency and on average would be just 1.5 times client latency. This is likely to be around 0.5 seconds - an order of magnitude less than MSN Web Messenger.

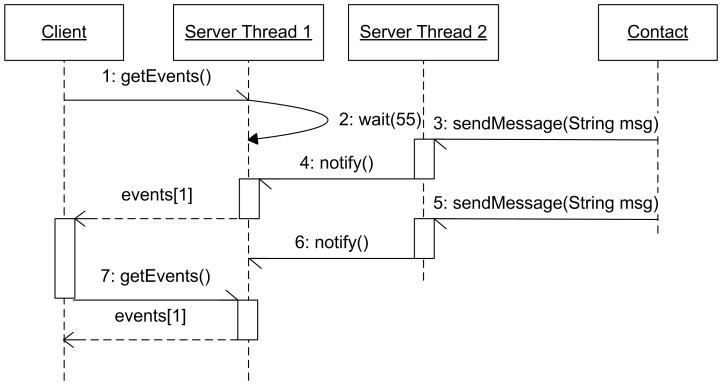


Figure 2.4: UML sequence diagram of Facebook Chat

2.3.3 GPokr

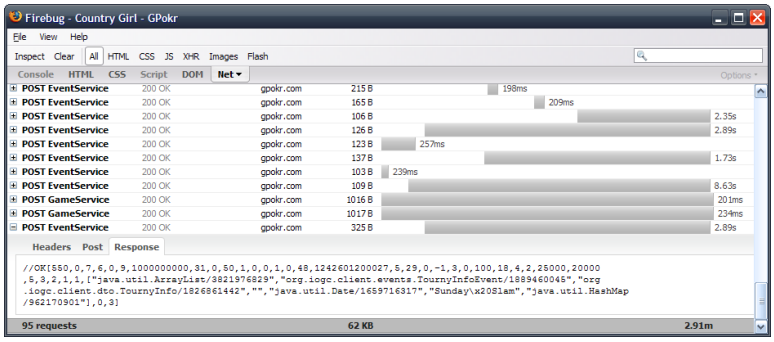


Figure 2.5: GPokr network activity

GPokr is a web-based Texas hold 'em poker game, which uses a combination of Ajax, GWT and Java (GPokr.com, 2006; Dewsbury, 2008). The client-side communication architecture is similar to that of Facebook chat, however GPokr disseminates events to multiple users. Unlike the other two implementations which have been examined, GPokr sends and retrieves *data* only, rather than executable JavaScript. This is both more efficient and more secure. It is less flexible, though, because software changes would require both the client and server to be updated rather than just the server. The MSN and Facebook implementations both send JavaScript code which is blindly executed by the client. This could conceivably be exploited by an attacker, whereas the data-only approach of GPokr naturally prevents this as no executable code is transmitted. Figure 2.5 shows the two types of calls; GameService for send events and EventService for receive events. GameService transactions are immediately returned after being successful, whilst EventService transactions are held on the server for the server-push mechanism.

2.4 Research Conclusions

The neologism 'Comet', coined by Russell (2006), is used to describe a technique used in both Facebook Chat and GPokr. Comet facilitates client-server communication at near minimum latency, and is the technique which will be used during the project.

The survey highlighted a preference for non-Flash websites *(see table B.8)*, and research showed a higher browser compatibility for JavaScript as an RIA platform. For these reasons, GWT will be used as the development platform and client-side architecture. GWT uses Apache Tomcat, a Java servlet container, as its server technology. Java servlets can be thought of as lightweight applications with inherent HTTP support. A single servlet instance persists between a potentially unlimited amount of client connections. This allows information persistence entirely in memory and without the use of secondary storage, a cause of increased latency.

The author has a moderate level of experience using Java; this should be mostly transferable to the use of Apache Tomcat and somewhat transferrable to GWT, which shares the Java language but nothing else. The author has had no past experience using either Java servlets or GWT. The current release of GWT is version 1.5, however version 1.6 is expected to be released during the course of this project. A brief evaluation of the new version will be conducted to discover any potential benefits which could be gained by an upgrade.

2.5 Technology Evaluation

A prototype application (Google, 2008f) was created using GWT in order to learn more about the software and allay any remaining doubts about its suitability. GWT is designed to be used with the Eclipse Java IDE (Google, 2008a), though the procedure for creating a project is unintuitive and repetitive. During development, GWT applications are run in the GWT Development Shell, known as 'hosted mode'. This Java shell could also be used if the application were to be deployed internally, as its performance is noticably superior to browsers' JavaScript engines. Eclipse and the Java compiler eliminate most potential problems, such as type errors and unreachable code. The development shell console is able to catch and display runtime exceptions such as NullPointer and ArrayIndexOutOfBounds. Notwithstanding incorrect logic, or faults in the GWT compiler itself, all problems can be eliminated before code even reaches the browser, an environment in which JavaScript is hard to debug due to a lack of debugging tools.

One of the principal advantages of GWT is that the same code can be run on both the client *and* server (Burnette, 2006, p3). For example, a name validation method which checks a string for numbers can be invoked on the client before form submission, and again on the server before it is acted upon. Due to latency, this particular example would execute much faster than a remote procedure call (RPC) to the server (Crane *et al.*, 2006, p11), saving client time and server processing power.

A GWT application starts off at `EntryPoints`, classes which bear a strong resemblance to the `main` method in Java. A GWT class should implement the `EntryPoint` interface, and subsequently an `onModuleLoad()` method. Like the Java's main method, this is the first instruction to be executed. GWT applications are composed of widgets, which are essentially extensions of HTML elements. For example, the `TextBox` widget is written to the browser as an `input` HTML tag. Unlike HTML however, these widgets should behave nearly identically across all supported browsers. Widgets can be extended and grouped using Java inheritance. Resembling Java's layout managers, GWT uses 'panels' to organise page layout, widgets which hold other widgets. These various similarities make GWT well suited to someone with Java experience.

GWT outputs six different JavaScript implementations, one for each of the major browser DOMs. This allows it to make optimisations for each browser, and elimates the usual browser-detection statements that clutter traditional JavaScript code. The GWT compiler transparently implements a significant amount of the java.lang

and java.util packages (Google, 2008b). Generic collections such as LinkedList and HashMap can be compiled directly to JavaScript. In addition to this, some classes are reimplemented with reduced functionality. One example of this is the Random class (java.util.Random -> com.google.gwt.user.client.Random). Most importantly though, GWT implements all of the standard language features of Java including:

- primitives, strings and arrays
- inheritance
- class, method and variable scoping
- static and final variables (and thus constants)
- interfaces
- constructors
- abstract classes and methods
- overriding and overloading
- control structures, including; for, while, do-while and switch statements
- the instanceof keyword
- serialisation

Notably absent are threading, reflection and object cloning. Most of these language features, such as disallowed instantiation of abstract classes, are not enforcable by JavaScript runtime environments because they are not part of the ECMA specification. They are instead enforced by the fact that source code is written in Java; the GWT compiler will throw any errors which it catches from the Java compiler before it starts compiling to JavaScript. These complex language features will allow vastly simpler source code and less code repetition in the implementation of the software.

2.5.1 Servlet Container

The Tomcat implementation used in GWT hosted mode is more than sufficient for its role in this research project (Wilkins, 2008b). There are alternatives though, with the popular Jetty distribution quickly gaining market share on its rival. Ultimately, there is no reason to use Jetty in this project, however in a production environment its improved performance would be beneficial.

Both Tomcat and Jetty are servlet containers forming part of the Java Enterprise Edition (EE) specification. Java EE is a comprehensive suite designed to facilitate service-oriented architecture (SOA). If the software created from this project were to be deployed in a production environment, it would likely utilise this type of archi-

tecture which is based around componentisation and interoperability between those components.

A specific example of this, which requires real-time communication, is that of a holiday booking system. The SOA architecture allows the travel agency to integrate with the services of other businesses, such as booking for hotels, flights and theme parks. Should a vacancy become reserved whilst the user is deciding, the client model should update to prevent a double-booking situation.

Chapter 3

Methodology

3.1 Development Methodology

Choosing the most appropriate development methodology is of paramount importance, especially in large projects. Current methodologies including agile, waterfall and spiral models share the prerequisite stage - planning. This includes background research, technology evaluation and in some cases a feasibility study. In the case of this project, that stage has already been completed. The use of a methodology is based upon the premise that a project cannot be conceptualised in its entirety. It is therefore necessary to decompose the development process into smaller stages which can each be undertaken rigorously.

3.1.1 Waterfall

The waterfall model illustrates the idea of a sequential development process where each stage (requirements, design, etc.) is completed in turn, and not revisited.

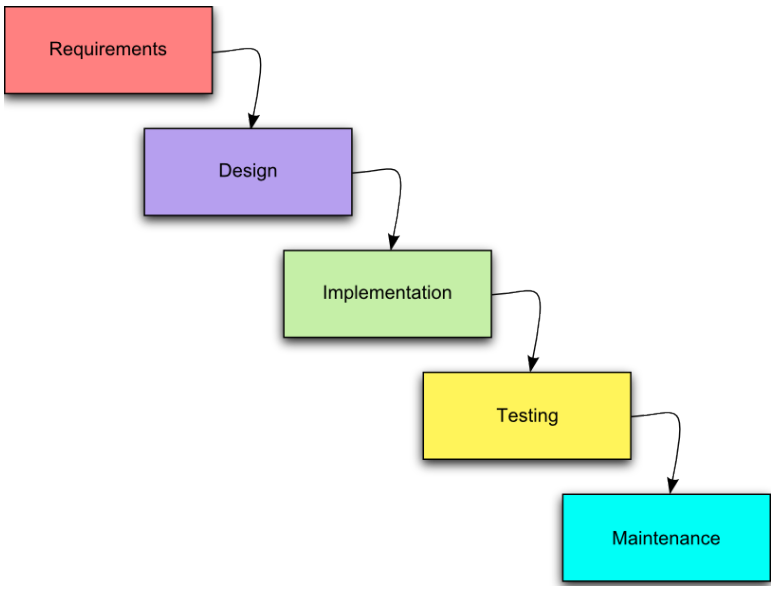


Figure 3.1: Waterfall model
Adapted from Smith (2009)

Critical Analysis

McConnell (1996, p72) estimates that rectifying a requirements issue after deployment costs 50 to 200 times more than during the requirements stage. The issue is compounded by the fact that it is often impossible to fully develop requirements until later stages in the development lifecycle (Boehm, 1988, p63). Clients themselves often do not fully understand their own requirements, or request additions to their software after project completion (Boehm, 1988, p63). In this case, development will be using two technologies which are completely new to the developer. Although high-level requirements are clear, detailed requirements are not straightforward and unexpected problems are likely to occur using the new technologies.

3.1.2 Spiral

The spiral model, developed as an extension of the waterfall model, emphasises the importance of design and planning at all stages in the development process (Boehm, 1988, p64). Successive prototypes are used to refine and isolate requirements.

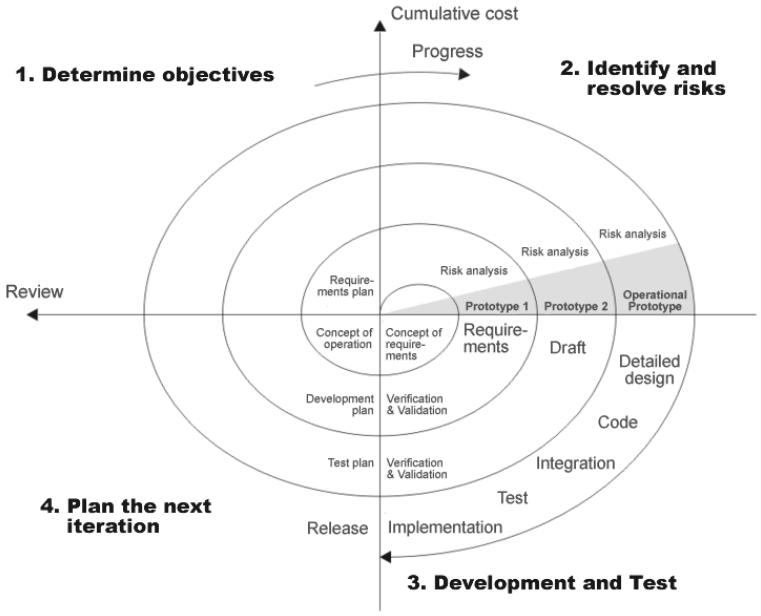


Figure 3.2: Spiral model, Boehm (1988, p64)
Source: Nutschan (2008)

Critical Analysis

Having been designed for use in large projects, the spiral model is extremely time consuming as it cannot easily be scaled down for use in small projects like this. The iterative process has clear advantages with quality assurance; issues will be caught at an early stage in one of the prototypes. The spiral model is able to cope well with changes that occur during the software development process. The scope of this project is limited - it is a single-person project spanning just a few months. Major changes are unlikely to occur during the project as it has to meet a specific deadline.

The spiral model bears a similarity to the writing of this report; initially written as a prototype, with errors and omissions expected. Entire sections are likely to be removed and rewritten more concisely, analogous to the redevelopment of software modules to improve performance. Further refinements such as proof-reading, akin to testing, are made to the report until the quality meets requirements.

3.1.3 Agile

Individuals and interactions over processes and tools
Working software over comprehensive documentation

> Customer collaboration over contract negotiation
> Responding to change over following a plan
> The Agile Manifesto (Beck *et al.*, 2001)

Agile software development is the most recent of the three methodologies being examined, and refers to a collection of principles which emphasise evolving requirements and iterative development. Contrasting with the other two models, agile methodologies are designed for small projects, and purposefully omit rigid planning; this is due to the likelihood that plans will be rendered inadequate because of requirements changes. Agile methodologies have consistently shown good results in small groups of less than a dozen developers (Koch, 2005, p21), however efficiency drops off sharply if the project size is much above this (Koch, 2005, p22).

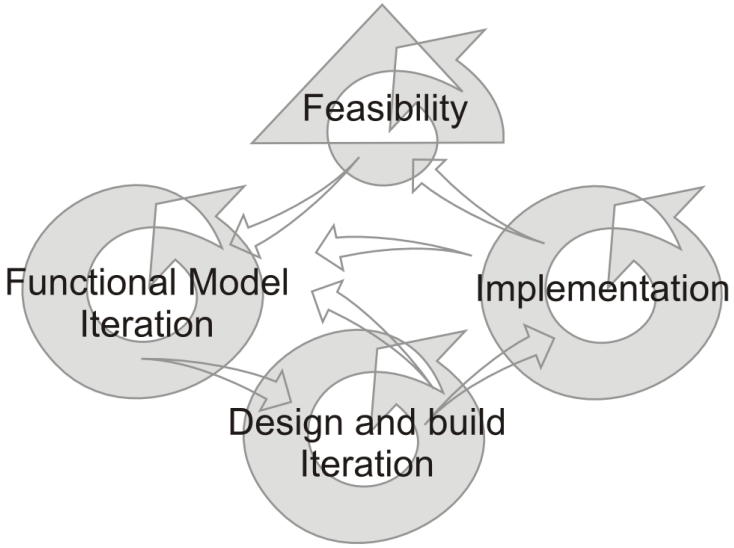


Figure 3.3: Dynamic Systems Development Method,
Source: Dekker (2008)

Figure 3.3 illustrates the iterative processes used by DSDM and most other agile methodologies. The process of iteration, which usually involves prototyping of some kind, produces software of higher quality.

Critical Analysis

The research nature of this project, and the developer's unfamiliarity with the software tools being used, are likely to lead to unexpected issues and changing requirements. This evolution of requirements is natural, as the developer will seek to continually

refine the research area. Agile's focus on these issues makes it very well suited for the project.

The most significant disadvantage of agile methodologies is the lack of documentation normally produced. While this documentation may be unnecessary for the original development team, it can be essential for developers who make enhancements and additions to software (Koch, 2005, p103), sometimes decades later. The piece of software being developed in this study is a proof of concept and an evaluation of the mechanism itself. The apathy towards documentation exhibited by agile software development is not a significant problem; it would be beneficial but is certainly not essential.

3.2 Chosen Methodology

Little research has been conducted on the methodology of the archetypal solo programmer (Hollar, 2006, pVII), which is, of course, the capacity in which this project will be conducted. Most methodologies are designed with teams in mind, and some are even reliant on this; thus in their original forms they are unsuitable for use in this project and must be adapted. Much of the planning documentation in large software development projects is produced to overcome communication problems. In a team, each member must know their apportionment, abide by the same standards and understand how their code will integrate with the rest of the project. These are all non-issues when there is only a single developer, as is the case here.

A methodology following agile principles will be used, with an emphasis on software quality through iterative development. The software tools being used, both GWT and servlets, are completely new to the developer; aspects and restrictions may arise unforeseen. Due to this, rigid and detailed design will not be conducted, however high level design will take place. The 'agility' of the methodology is designed to accommodate these changing requirements with as little disruption as possible.

Code reuse is one of the main objectives of the project, and yields greater cost and time savings than any other rapid development practice (McConnell, 1996, p527). The development of the framework will allow a series of applications to be created with minimal overhead compared to a single application. That said, if only a single application was being constructed, then the framework would increase the overall time of the project. In that situation it may still be desirable to split the networking architecture from the application architecture. This is known as separation of concerns, whereby the piece of software is separated into modules. These can then

be developed independently, with only the linking points or '*hotspots*' remaining the same. This type of modularity allows individual components to be developed by independent teams, which can help avoid the diseconomies of scale associated with large projects.

3.2.1 Testing

Regression testing will be used to maintain software quality during development and as the size of the project increases. This testing will be performed as each discrete piece of functionality is added. This type of testing is perfectly suited to agile methodologies, and is one of the testing approaches specified by Extreme Programming (XP).

3.3 Revision Control

Revision control is the process whereby changes to source code are monitored and recorded. In projects of ever-increasing size and complexity, the management of source code can become difficult due to concurrent use. Concurrent development by multiple programmers is usually a necessity and revision control prevents edit conflicts (Koch, 2005, p33); where recent changes are overwritten by files which were branched from old revisions. Revision control can also be used to isolate and fix bugs in the software. If a bug arose as a result of a change to the software, this specific version can be located and the problem can be fixed relatively quickly. The developer needs only to look at the most recent changes, rather than trawl through an entire class or indeed repository.

The agile development approach being used makes revision control even more important (Koch, 2005, p94). The continuous testing approach employed, whilst being time-efficient, is not comprehensive. It is likely that bugs will go undiscovered for significant amounts of time. Even more importantly, as the project is the development of a framework upon which other software is based, snapshots of past versions must be retained as the end-user software will rely on this. Backwards compatibility of software in conjunction with the framework could also be an issue, however due to the scope of this project it may be more efficient, and more agile, to make these corrections individually (Koch, 2005, p34). Compatibility issues primarily arise with deployed software, where it is not always possible to force an upgrade.

Chapter 4

Requirements and Evaluation Method

4.1 Overview

As part of the agile methodology being used, requirements analysis must take place. Though the methodology discourages detailed and rigid design, clear requirements are required for any project to succeed, not just software engineering projects. Functional specification does not usually take place in agile development because it delivers little considering the time investment that is required.

The most efficient use of requirements engineering in an agile project is to document only those requirements that are unlikely to change. These requirements define the scope of the project and prevent 'feature creep', where additional but unnecessary features are added - often a problem in agile development. In this project requirements will be prioritised, allowing less important features to be dropped if the project runs out of time.

4.2 Prioritised Requirements

The software has a single primary requirement which *must* be achieved for the project to be a success:

- server-push communication over HTTP

Additionally, there are three secondary objectives which will measure how *useful* the software is in a real-world context:

- compatibility with as many web browsers as possible
- packaging as a library, or ideally as a framework, in order to facilitate code reuse
- communication at near or equal to client-server latency

Finally, there are two tertiary objectives which will be used to measure the *quality* of the software. These requirements are unnecessary, but desired, and may be ignored if time does not permit their inclusion.

- equal or better performance than existing systems
- simple and unrestrictive interaction with the library/framework

4.3 Evaluation Methods

Since the development artefact is a framework through which other software can operate, examples of this software must be created. Three software pieces will be developed, each placing progressively higher requirements upon the framework:

Minimal the most basic use for which the framework could be used; should this application fail, the project will be deemed a failure

Intermediate the anticipated normal level of stress which could occur for most uses of the framework

Maximal the highest amount of stress which the framework could reasonably be expected to handle; should this application succeed, the project will be deemed a complete success

Ideally, all of the applications will function as desired; however, should one or more perform below standard, the progressively increasing requirements will allow an interpolation of test results in order to determine the success of the framework itself.

4.3.1 User Evaluation

Independent users will be asked to test the applications and evaluate their performance - a questionnaire will be used for this purpose. The user evaluation will provide an unbiased review of the software and provide tangible evidence of the success or failure of the project, as a whole or in part.

Chapter 5

High-Level Design

5.1 Ajax Connecting Testing

As previously mentioned, Facebook and Gpokr both implement a 55 second timeout on connections *(see figures 2.3 and 2.5)*. It is assumed that there exists a browser-enforced timeout for Ajax connections. A simple testing application will be created to investigate this. The following pseudo-code illustrates how the application will be implemented:

Algorithm 1 Client side code for the test application

for $i = 1$ to 20 **do**
  timeoutTest(i) {asynchronous method}
end for

Algorithm 2 The timeoutTest method

$startTime \leftarrow currentTimeMillis()$
DO ajaxRequest(i)
$endTime \leftarrow currentTimeMillis()$
DO printLine $endTime - startTime$

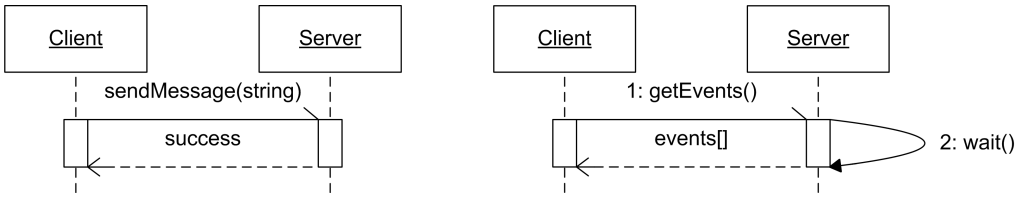
Note that the servlet environment will create a new thread for each request.

Algorithm 3 Server side code for the ajaxRequest method

$n \Leftarrow i \times 5$ {get time in seconds}
sleep(n) {sleep server thread for n seconds}
return

5.2 Asymmetric Response

Each of the systems reviewed in the existing software evaluation *(see section 2.3)* implemented a standard remote procedure call (RPC) pattern. In particular, each system used separate channels for sending *(figure 5.1a)* and receiving *(figure 5.1b)* data. Both Facebook and GPokr use what is essentially a *reverse*-RPC mechanism; the client gives the server permission to tell it what to do.



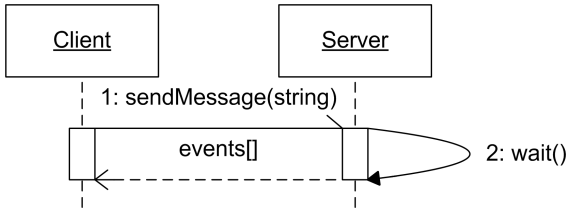
(a) A standard sendMessage RPC (b) A server-push getEvents reverse-RPC

Figure 5.1: Existing use of RPC and reverse-RPC

The response sent by the server in figure 5.1a is essentially wasteful; HTTP presumes reliable transport, and as such any systems built on top of it can also make this assumption. Typically HTTP is implemented using the TCP/IP stack, wherein TCP guarantees delivery of a message, or in the very least offers notification that transmission was unsuccessful (Cerf *et al.*, n.d.). A 'success' response using HTTP is completely superfluous; the reason for its existence is that standard HTTP systems do not implement server-push, but should still close the connection gracefully.

A better implementation (hereafter referred to as 'asymmetric response') is to retain the HTTP request and return events using it later *(see figure 5.2)*; the server tells the client what to do despite the nature of the originating RPC call. This implementation could be up to 50% more efficient than existing implementations of Comet. This approach to networking has no benefit with protocols other than HTTP, where there are no limits on the number of connections that can be made, and server-push is not restricted.

Figure 5.2: An asymmetric response sendMessage RPC



5.3 Evaluative Software

The primary goal of the software artefacts will be to evaluate the performance of the framework. A reasonable level of usability will be required in order for independent users to test and evaluate the software.

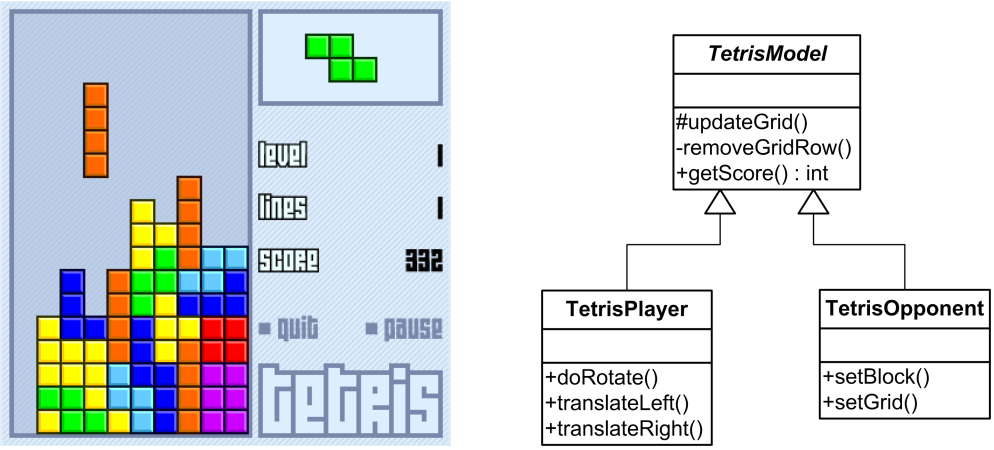
5.3.1 Chatroom

An instant messaging system will be developed with support for at least 3 simultaneous users. A message submitted by a user should be delivered to all other users in a timely manner. Frequency of communication will be low to very low and bandwidth requirements will also be low. This type of communication is common among Ajax systems, such as in live news feeds and live sports scores.

5.3.2 Tetris

The Tetris artefact will demonstrate whether it is possible to mirror a dynamic data model in real-time across HTTP.

Figure 5.3a illustrates a standard Tetris game which supports a single player. The player can rotate blocks and move them left or right; these commands will need to be mirrored across the network. The Tetris artefact will consist of two tetris games - the user's view and a real-time representation of an opponent's view. For every five rows which are scored by a player, the opponent will lose a row; this provides meaning to the real-time interaction. Java inheritance will ease the creation of the two views *(see 5.3b)*.



(a) A standard, single-player game

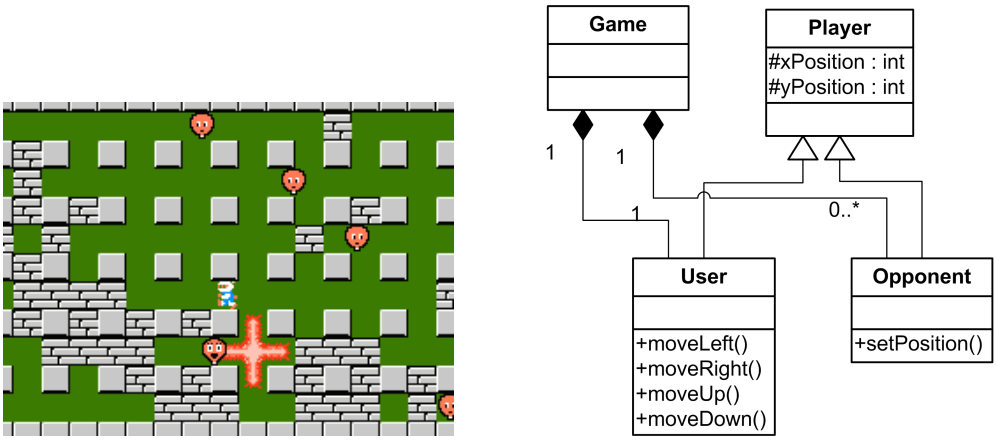
(b) Java inheritance in the new application

Figure 5.3: Tetris

5.3.3 Bomberman

This artefact will build upon the requirements of the Tetris artefact by introducing dependence upon the recency of the data model. A player's actions depend upon the locations of other players in the game, and therefore the data model must be completely up-to-date for the player to make an informed decision.

Figure 5.4a illustrates a standard Bomberman game. Classically the player (near screen centre) must defeat artificial intelligence (AI) enemies (orange balloons) by laying 'bombs' which destroy objects in adjacent squares. The artefact will allow between two and four players to compete against each other rather than against AI. As the environment is shared between users rather than each user having their own, synchronisation of events will be extremely important.



(a) A standard, single-player game (b) Java inheritance in the new application

Figure 5.4: Bomberman

In addition to these features, the Bomberman application will include the entire functionality of the chatroom. The frequency of user activity will be higher than in the other two applications, and this will be compounded by the increased number of players. The server will have to process n^4 events whilst each client will have to process $4n$ events. This will further increase the workload of the framework and demonstrate the efficacy of using an asymmetric response architecture.

5.3.4 Summary

Table 5.1: Artefact communication requirements

Artefact	Latency	Bandwidth	Frequency
Chatroom	Low	Low	Low
Tetris	Low	Medium	Medium
Bomberman	High	High	High

Table 5.1 illustrates the demands that each artefact will be place upon the framework and its network architecture.

Latency: the effect of latency upon the performance of the application; 'high' denotes performance is highly dependent upon *low* latency

Bandwidth: the expected bandwidth requirements of the application; unlikely to cause many issues due to modern high-bandwidth internet connections

Frequency: the frequency of data events occuring; places higher dependence upon both bandwidth and latency

The chatroom client is expected to perform acceptably despite high latency and low bandwidth. Communication will be infrequent due to the time it takes a user to write a message. Conversely, the Bomberman clone will not tolerate high latency, because a user's actions are directly dependent upon those of other users.

5.4 Framework

The framework in itself has no executable; it must be used in combination with another application. For this reason it will be developed in combination with the simplest artefact, the chatroom.

5.5 Development Methodology

A three-cycle iterative development process will be used in order to develop the framework and artefacts:

First this will encompass the development of the full functionality of the chatroom application. For simplicity, the chatroom and Comet architecture will be developed in a single codebase.

Second this will focus on the separation of the Comet architecture into a framework. The Tetris and Bomberman applications will then be developed using the framework.

Third performance and efficiency improvements will be made during this phase; these improvements are expected to be discovered as knowledge of the development tools increases.

Chapter 6

Implementation Issues

6.1 First development iteration

6.1.1 Remote Procedure Calls

The GWT remote procedure call (RPC) framework is the fundamental mechanism which the framework will operate through. Whilst various RPC standards such as SOAP, XML-RPC and RMI already exist, none are ideal for a JavaScript implementation; for this reason, GWT-RPC was created (Burnette, 2006, p33).

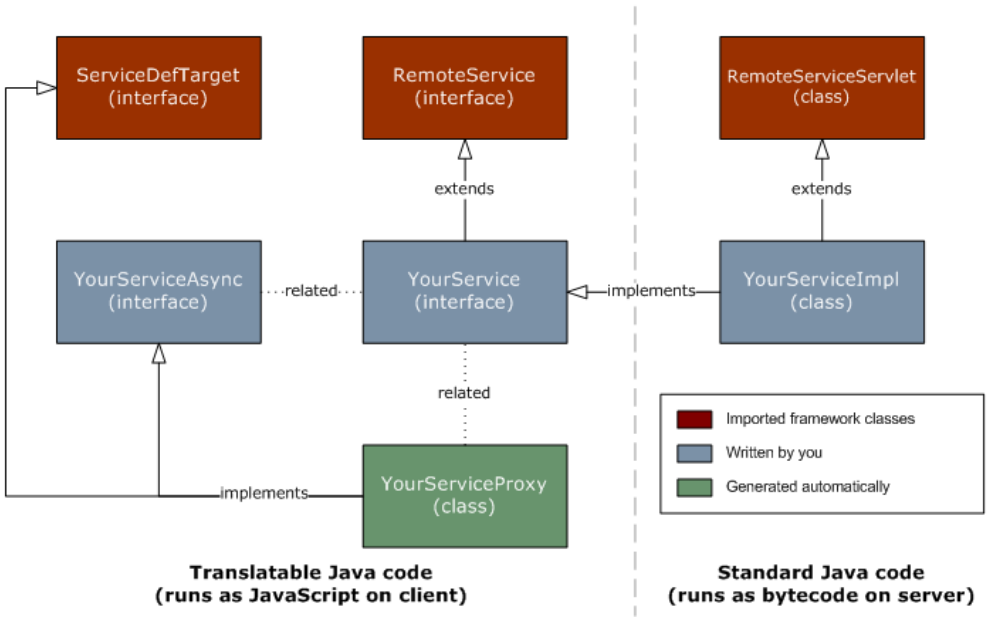


Figure 6.1: A basic and standard implementation of GWT-RPC
Source: Google (2008c)

The `RemoteService` interface is simply a marker interface, it has no Java source code; however its inclusion instructs the GWT compiler to include specific JavaScript code at compile time. It is this JavaScript code which handles the encoding and transmission of asynchronous requests.

Any object which implements either the GWT IsSerializable or Java Serializable interface can be encoded using GWT-RPC, though the Object class itself is not serialisable. Classes are allowed to implement these interfaces only if they obey certain characteristics (Google, 2008e). This can be summarised to mean that a class is serialisable if it is composed solely of primitives and arrays, or objects which themselves are composed solely of these.

The `RemoteServiceServlet` (RSS) class handles the deserialisation of requests and serialisation of responses. Normal GWT servlets should implement the associated service interface and thus allow the RSS to invoke RPC calls. The RSS class automatically and transparently supports HTTP's `GZip` and `deflate` compression methods. This leads to significant bandwidth improvements compared to uncompressed communications as most messages are less than 0.5kB.

6.1.2 Connection Timeout

The testing application described in section 5.1 was implemented using GWT-RPC. The expected result was that connections lasting over 60 seconds would not respond correctly. The actual results are shown in table 6.1. Even connections of more than 100 seconds latency did not cause a timeout on either the server or client.

Test	Expected	Actual	Comment
A	5000	5205	Correct
B	10000	10159	Correct
C	15000	20169	A + expected
D	20000	30139	B + expected
E	25000	45162	A + C + expected
F	30000	60133	B + D + expected
G	35000	80156	A + C + E + expected
H	40000	100133	B + D + F + expected

Table 6.1: Actual test results using GWT hosted mode browser
(results after test H omitted)

When the GWT hosted mode browser ran tests A and B concurrently, the results became irregular. A pattern emerged and this is listed in the `'Comment'` column. Furthermore, results using Firefox were different *still*, see figure 6.2.

Test	Expected	Actual	Comment
A	5000	5013	Correct
B	10000	10011	Correct
C	15000	15011	Correct
D	20000	20009	Correct
E	25000	25010	Correct
F	30000	30010	Correct
G	35000	39987	A + expected
H	40000	49988	B + expected
I	45000	59990	C + expected

Table 6.2: Actual test results using Firefox browser
(results after test I omitted)

The results show that tests A through F were run concurrently. Interestingly, using IE7 the results were the same as shown in figure 6.1. An investigation revealed the following:

> A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy.
>
> Source: Fielding *et al.* (1999, section 8.1.4)

The anomalous results shown in figure 6.2 were caused by a deviation from normal standards by the Firefox client (MozillaZine, 2008).

Implications

In order to meet the secondary project objective of cross-browser support, the application framework will be required to work with just two connections available. As one connection will be held on the server at all times, this may cause a significant bottleneck with the single remaining connection for client requests *(see figure 6.2)*.

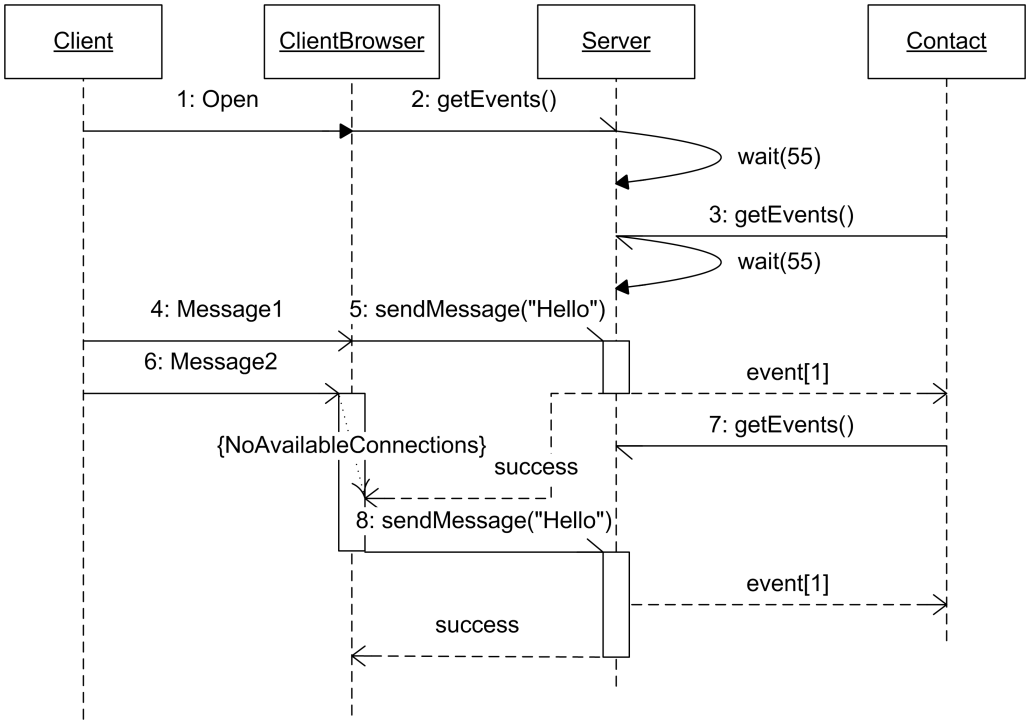


Figure 6.2: UML sequence diagram showing the delay caused by a two-connection limit.
Note the delay between events 6 and 8 (labelled NoAvailableConnections).

A client which allows more than two connections, such as Firefox 3 which uses up to 6 (MozillaZine, 2008), would not suffer from this problem. The same situation is possible in reverse though, with the server having events waiting but no connections on which to return them. For Firefox it is perhaps advisable to retain more than one connection on the server, however any implementation must be convinced of a client's capability before doing so. Identification by the request header user agent string would not be sufficiently trustworthy; it is *easily* possible to change Firefox to the HTTP standard value of two connections, in which case a retention of two connections by the server would render the client unable to transmit data.

Additionally, the persistent connection quota in Firefox is shared between *tabs*; this means that should a user open an application in *six* separate tabs, they will all cease to function. The solution to this is simple, though; using the session identifier *(see section 6.1.6)* the servlet can refuse connections from a client if it already has active sessions in the same browser.

6.1.3 Server Side Reflection

Reflection is the ability of a program to observe and modify its own behaviour at runtime. RPCs usually use reflection; a string is sent across a network and through self-examination a class invokes the method which matches that string. Figure 6.3 shows a string encoded by GWT for use in its RPC mechanism. The string is instructing the server to execute (and possibly return the result of) the `clientOpen` method, in an object of type `CometService`.

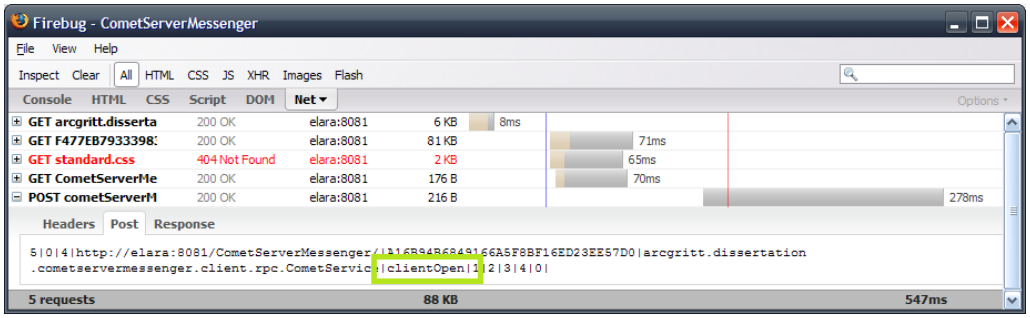


Figure 6.3: Remote procedure calls in Google Web Toolkit

6.1.4 Client Side Reflection

The cardinal functionality of the client is the reverse-RPC mechanism, figure 6.4 shows an example of this. A `nullRequest()` call is made containing no information, the client doesn't know what response to expect, but is able to call a different method for each of the three events returned *(sequence events 11, 12 and 13)*.

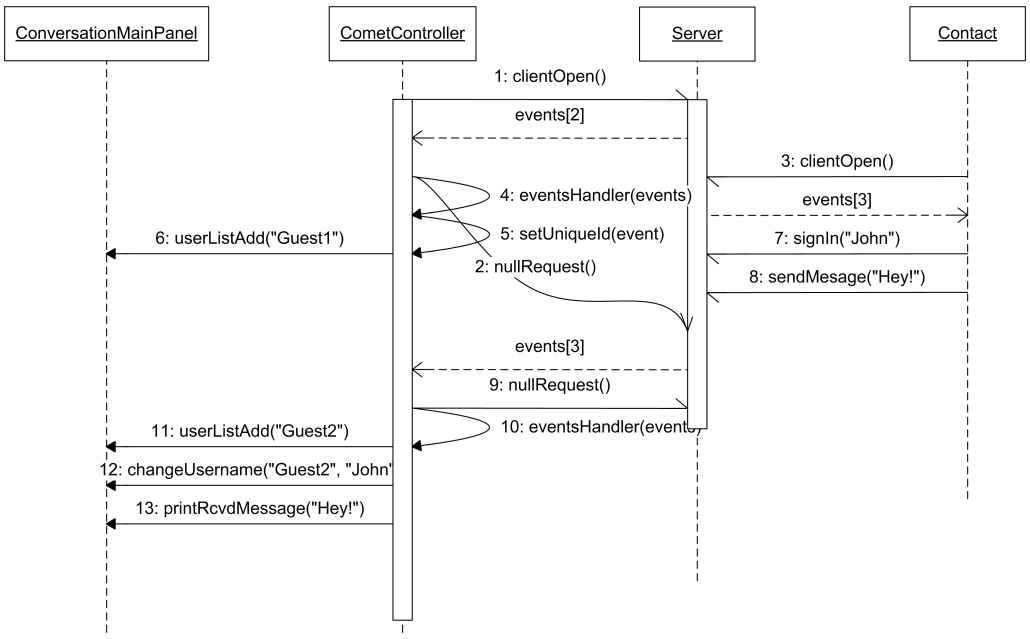


Figure 6.4: UML sequence diagram showing a reverse-RPC mechanism

Client side reflection is not supported by GWT, and therefore had to be implemented as part of the Comet framework. This essential functionality is implemented using the instanceof keyword supported by GWT *(see figure 6.5)*.

Figure 6.5: Client side reflection using the `instanceof` keyword

```
1  private void eventHandler(ClientEvent[] events) {
2      if (currentPersistentRequests < 1) {
3          incrementRequests();
4          convoSvc.nullRequest(callback);
5      }
6      if (events != null && events.length > 0) {
7          for (int i = 0; i < events.length; i++) {
8              ClientEvent currentEvent = events[i];
9
10             if (currentEvent instanceof Message) {
11                 mainPanel.addMessage((Message) currentEvent);
12                 return;
13             }
14             if (currentEvent instanceof UserJoin) {
15                 mainPanel.addUser((UserJoin) currentEvent);
16                 return;
17             }
18             if (currentEvent instanceof UserList) {
19                 mainPanel.updateUserList((UserList) currentEvent);
20                 return;
21             }
22         }
23     }
24 }
```

Upon receiving a response, the client checks whether it has any outstanding requests on the server; if it doesn't, a `nullRequest` RPC call is made so that the server can continue to push events to the client. As the null request is made asynchronously, the client can continue to process the events which it received in the last response. Unlike Java, GWT does not provide a method to retrieve a string representation of an object's type.

Figure 6.6 shows the class hierarchy of events in the Chatroom application. A series of if-statements are used to detect the ClientEvent object's true type; it is then cast back to this type and passed to a method which will execute the appropriate logic.

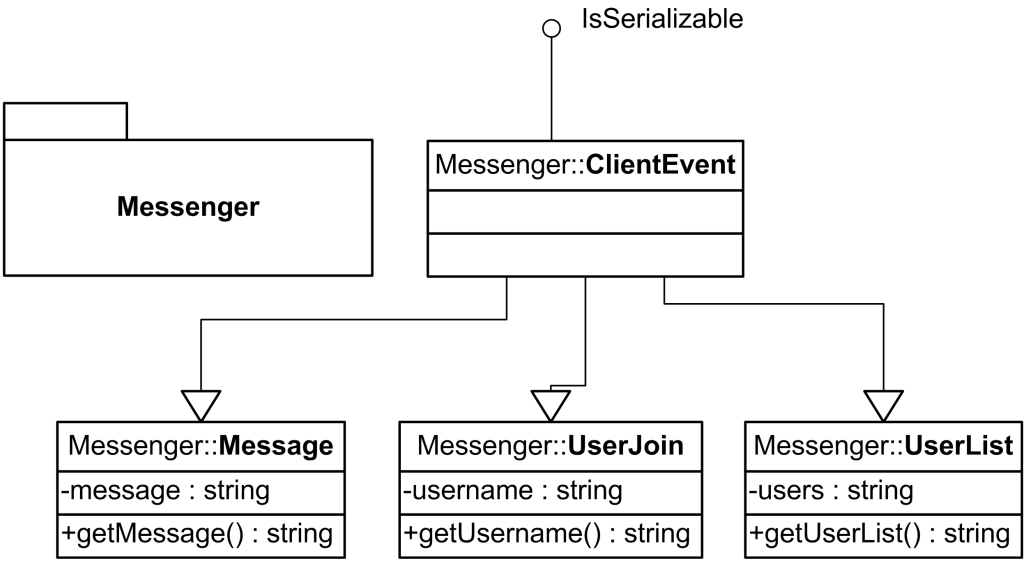


Figure 6.6: UML class diagram showing the hierarchy of serialisable events

6.1.5 Synchronization

The servlet environment creates a new thread for each request it receives. In the event that two requests are processed concurrently, two threads will be active and thread interference and memory consistency errors can occur. This stems from the fact that a single line of code is typically many instructions, not just one (Sun Microsystems Inc., 2008a). The solution to this problem is synchronization, which is used to provide an intrinsic lock of objects to a particular thread. Other threads which then attempt to access the particular object are paused until the lock is lifted. There are two methods for this Java; synchronized methods and synchronized statements. A synchronized method causes a lock on the class object to which the method belongs. A synchronized statement locks only a particular object specified in the statement. Synchronized statements offer better granularity, as only the specific object being modified or accessed needs to be locked.

Figure 6.7: Unsafe code

```
1  if(user.events.count == 1) sendResponse(user.events.pop());
```

Figure 6.8: Thread-safe code

```
1 synchronized(user.events) {
2     if(user.events.count == 1) sendResponse(user.events.pop());
3 }
```

In the servlet implementation user objects have to be locked often in order to prevent a multitude of issues which would have otherwise occured. If two threads began the execution of the code shown in figure 6.7 at a very close point in time, the following could occur:

1. Thread A: check user.events.count == 1, true
2. Thread B: check user.events.count == 1, true
3. Thread A: retrieve events.pop()
4. Thread B: retrieve events.pop() causes NoSuchElementException because events.count() == 0

The solution is shown in figure 6.8, which would force the following execution pattern:

1. Thread A: check user.events.count == 1, true
2. Thread A: retrieve events.pop()
3. Thread A: execute sendResponse method
4. Thread B: check user.events.count == 1, false

Though the situation is very unlikely to occur because the instructions take mere milliseconds, it is very important to have correct code; the instructions are likely to be run thousands of times and eventually an error *will* occur.

6.1.6 Faulty Session Identifier Implementation

Identification of user requests is accomplished using sessions; in most cases a user specific data-set stored on the server which is identified using a session identifier (SID). The identifer is stored on the user's computer as a cookie. Sessions and cookies are used by many websites, for example eBay, to identify whether a user is logged in. As such, it is desirable for the session to be active across all tabbed browsers such as Firefox, so that the user can interact with sites with their full privileges.

```
1 private String getCurrentUserId() {
2     return getThreadLocalRequest().getSession().getId();
3 }
```

The getThreadLocalRequest method returns the current thread's (Sun Microsystems Inc., 2008b), and thus the current user's, HttpRequest object (Sun Microsystems Inc., 2007a). If the user does not have a session, one is created, and the SID (JSESSIONID) is added to the response headers of the request (Sun Microsystems Inc., 2007b, p83-88) *(see figure 6.9)*. SIDs should be server-generated, unique and highly random in order to prevent spoofing, especially in the case of websites where confidential information is held on the server.

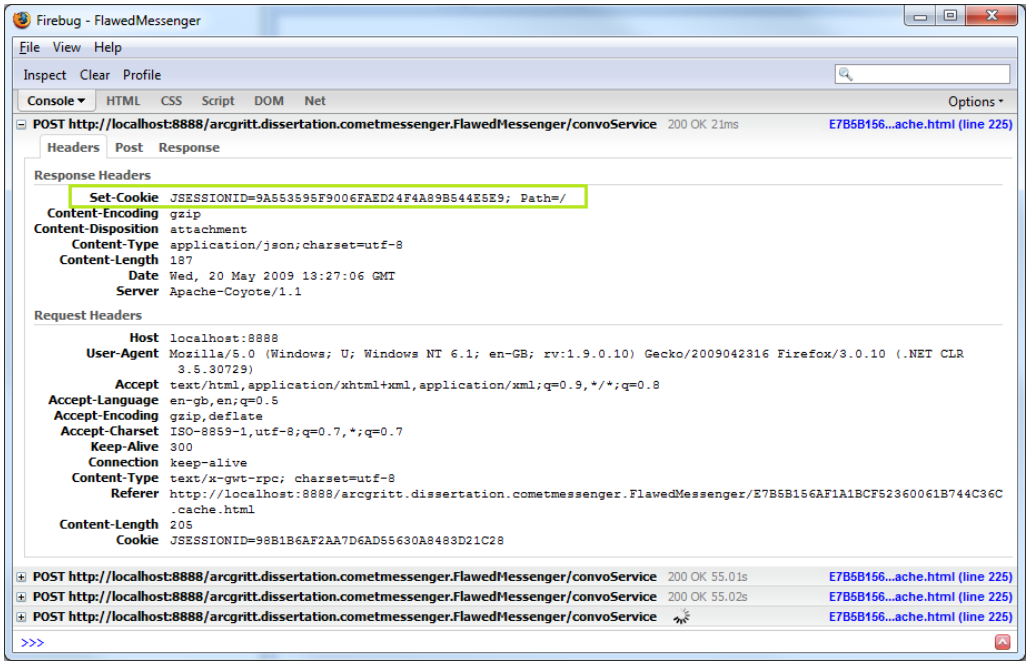


Figure 6.9: Network requests in a flawed version of the Chatroom application

The first prototype of the Chatroom application worked flawelessly with multiple clients (GWT, FF3, IE7), however when the application was loaded twice in the same instance of either Firefox or IE7, the system broke down entirely. Because the JSESSIONID is shared across all browser tabs, each tab is treated as the same user and requests were returned to the wrong tab. There is no solution to this issue, because there are no header attributes which distinguish one tab from another. This forced a reimplementation of the entire SID stack; using JavaScript rather than cookies, a blocking request is made to fetch a unique identifer. The new SID must be transferred in all subsequent requests to the server in order to distinguish separate

tabs in the same browser *(see figure 6.10)*. The `NoSessionIdentifierAvailable` label denotes the time which is wasted due to the blocking `clientOpen` call.

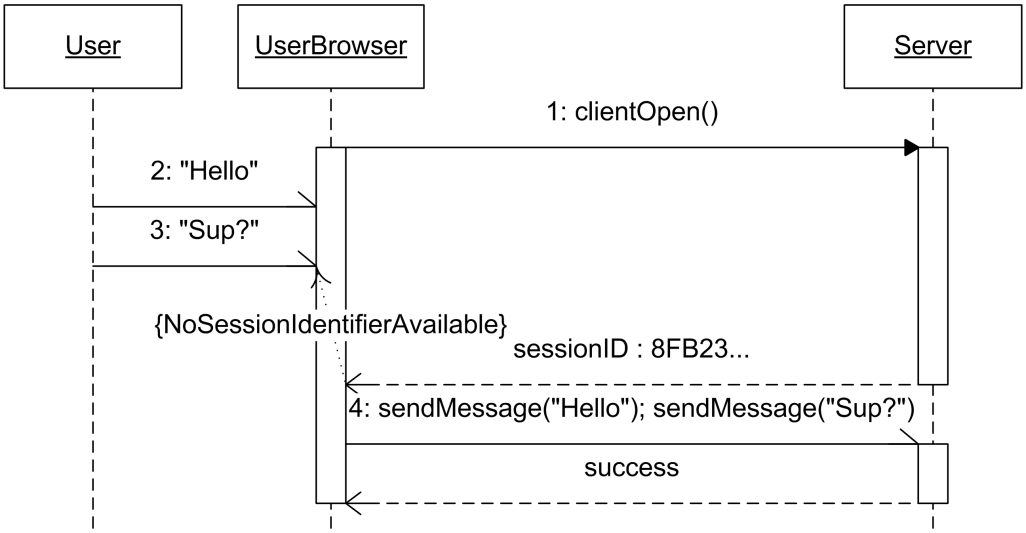


Figure 6.10: Client requests delayed by blocking clientOpen call

The absence of a reimplemented SID stack in the GPokr website 2.3.3 prompted testing it for this issue; the problem *does* in fact exist in the live version of GPokr as of May 2009.

6.2 Second development iteration

6.2.1 Framework codebase separation

Following the successful development of the chatroom's basic functionality, the core Comet functionality was separated. The flexible points of a framework are known as hotspots (Markiewicz & Lucena, 2001); abstract classes and methods which can be customised when they are implemented by the framework user. Care must be taken with method visibility so as not to restrict the functionality of applications using the framework.

The separation process greatly increased the complexity of the codebase; the amount of concrete classes in the application itself remained the same, but new abstract classes were created as part of the framework. It is important that a framework's integral logic and control flow are encapsulated so that they cannot be inadvertently altered. This is accomplished using the `final` keyword in Java, which prevents method and variable overriding by subclasses.

6.2.2 Increasing Event Granularity

Whilst the first development iteration focused on functionality, the second focuses on quality. The three RPC events in the original version of the software *(figure 6.6)* encompass all of the data which will be sent to and from the application during its operation. The use of these, however, would cause a significant amount of redundant data to be sent across the network; for example sending a list of all users each time a new user joins. Having eparated the framework code from that of the chatroom, development becomes much simpler. A new collection of RPC event types was created with an increased granularity; specifically a type of event for every use case 6.11. This not only increases the efficiency of information transport, but gives the application greater control over its response to events.

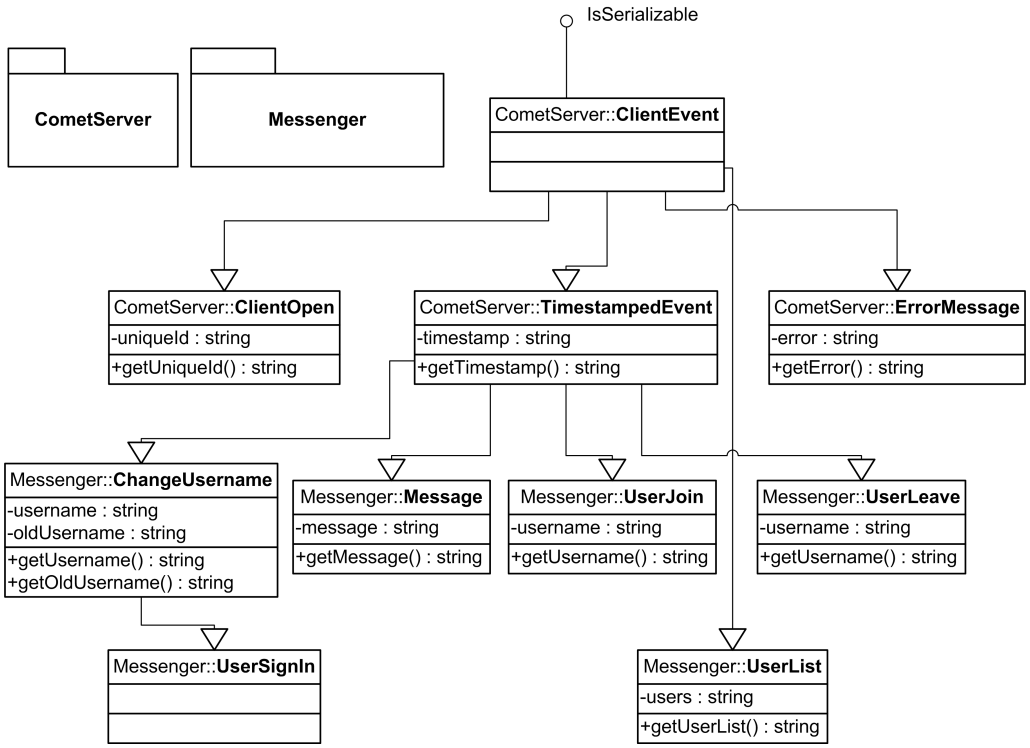


Figure 6.11: Event class hierarchy of the chatroom application after the second development iteration

6.2.3 GWT EventListener Issues

The Tetris and Bomberman artefacts require notification of user keyboard events in order to operate. In GWT, events are handled similarly to Java; classes which implement the SourcesKeyboardEvents interface can have a KeyboardListener added to them. This listener is passed information generated by the event, such as which key was pressed, and the developer can then act upon it. In JavaScript this would normally be implemented using a window or document level listener, however in GWT this is not possible (Google, 2009a); instead events must be sourced from a GWT widget which implements the aforementioned interface. This is very restrictive as it requires the user to give focus to a particular element in their browser. The TextBox field was used for this purpose, as it is always clear to the user whether or not it is in focus.

The GWT development team acknowledges the issue and has flagged a document-level listener for inclusion in a future release (Google, 2009a). This problem is expected to cause usability issues and will be investigated in the user evaluation.

6.2.4 GWT and Cascading Style Sheets

Whilst GWT handles the HTML *and* JavaScript, it helps very little towards cross-browser compatibility with Cascading Style Sheets (CSS). An initial implementation in both the Tetris and Bomberman artifacts used the `background-image` CSS property on table cells in order to display animation. This worked correctly in Firefox but it did not in Internet Explorer (IE). A "hack" had to be made whereby a GWT `Image` widget was added to each table cell which displayed a transparent 1x1 GIF image. The `background-image` property was then assigned to this image, and then the implementation worked correctly in all browsers.

6.3 Final development iteration

6.3.1 Servlet I/O

Having succesfully constructed the framework, efficiency and performance improvements can be examined. The most obvious issue with the original framework is its use of the wait() command; this would cause a thread to be used for each of the connections to the server *(see figure 6.12)*. Whilst this may not be an issue in small applications (the default maxThreads value is 200), in a production environment it would cause severe limitations on the system (Dewsbury, 2008, 470-472).

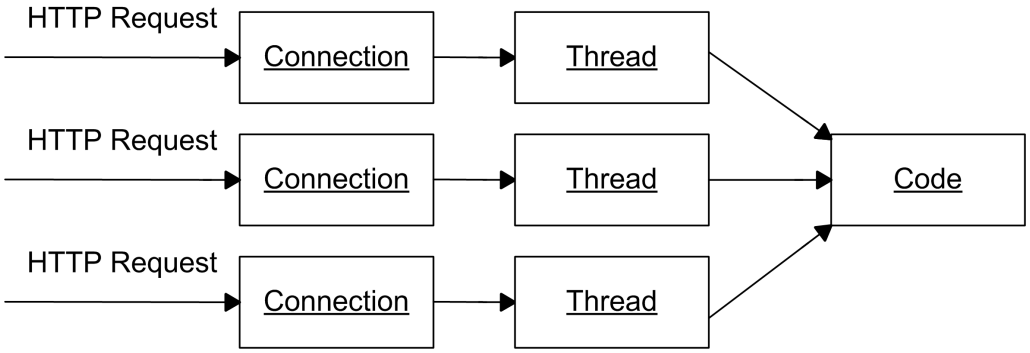


Figure 6.12: Standard HTTP IO in Apache Tomcat using the HTTP connector
Source: Dewsbury (2008)

The solution to this is non-trivial, it involves changes to the HTTP connector itself; fortunately such a connector has already been produced. The non-blocking input/output (NIO) connector *(figure 6.13)* allows a servlet read and write from connections asynchronously, rather than using a blocking read (Apache Software

Foundation, 2008a).

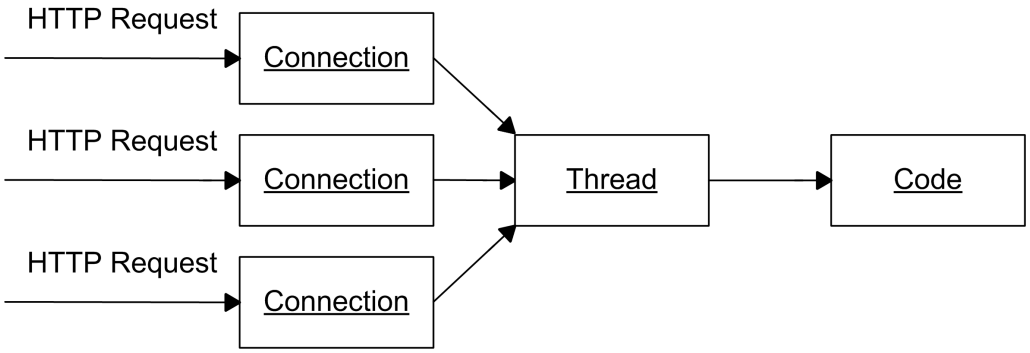


Figure 6.13: Advanced HTTP IO in Apache Tomcat using the NIO connector
Source: Dewsbury (2008)

This enhanced functionality is harnessed on the servlet using the CometProcessor interface supplied with Tomcat (Apache Software Foundation, 2008a,b). The basic `doGet` and `doPost` methods used in Java servlets are replaced by a single `event` method. Uniquely, the event method is called in any of four situations; BEGIN, READ, ERROR and END. A typical request consists of a series of event calls, for example BEGIN -> READ -> READ -> TIMEOUT (Apache Software Foundation, 2008a). This allows the HttpServletRequest and HttpServletResponse objects to be read and written to at any point between the BEGIN and END events.

The CometProcessor interface was implemented only recently, in Tomcat version 6. GWT's hosted mode servlet container is based on Tomcat version 5, and therefore does not have support for the CometProcessor interface. This presents a major obstacle in the development environment; the servlet and client javascript must be deployed to a fully functional server for testing. GWT JavaScript compilation is extemely slow; several minutes on a modern processor even for an application of just 1000 lines of code. A series of minor changes to code which would otherwise take seconds become a tedious process taking up to five minutes. Fortunately, `-noserver` command allows the GWT hosted mode browser to be used with an external servlet container; this allows a developer to continue using the excellent debugging facilities of the GWT hosted mode browser, however it does not solve the issue of having to compile and deploy an application each time it is changed.

6.3.2 Server Side Reflection

Newly in version 1.5, GWT's RPC class hierarchy was made visible to external packages. Serialisation and deserialisation are now pluggable features which can be used by implementations which do not stick to the rigid confines of the GWT RemoteServiceServlet mechanism. This is important because if the classes were not visible, as in GWT 1.4, then any changes to GWT would require reimplementation of servlet code when upgrading (Burnette, 2006, p34). Using the pluggable libraries assures compatibility with later versions.

Figure 6.14: RPC invocation using reflection, in the final version of the framework

```
1 RPCRequest rpcRequest = RPC.decodeRequest(RPCServletUtils.
    readContentAsUtf8(event.getHttpServletRequest()));
2 Method targetMethod = rpcRequest.getMethod();
3
4 try {
5     targetMethod.invoke(cometService, rpcRequest.getParameters());
6 } catch (Exception e) {
7     e.printStackTrace();
8 }
```

Figure 6.14 shows the Java code used to implement the RPC mechanism. The RPCRequest object is the GWT library's container for an RPC call. It holds the method, parameters and serialization policy of the call (Google, 2008d). After processing the UTF8 string supplied by the browser *(figure 6.3)*, line 5 would be equivalent to `cometService.clientOpen();`. Note how the return value of the invoked method is neither stored nor used, this is because the HTTPServletRequest will not be answered yet.

It was decided that best practice would be to return any data using the *oldest* request. In the case of the clientOpen call, it is the first and only request received, and thus will be the same one as returned. In most other situations, a previous request which had been held on the server for a period of time in the `User` object would be returned with the results of the incoming RPC call.

Chapter 7

Evaluation

7.1 Development Evaluation

Whilst there were many issues encountered during development, these were expected, planned for and overcome.

7.1.1 Chatroom

Though placing the least stress on the framework, the Chatroom *(see figure 7.1)* is the most efficient and refined artefact. The granularity of Comet events is very high; the only disadvantage of this is increased development time. Though arguably, too much time was allocated to regression testing and refinement when compared to the other two applications.

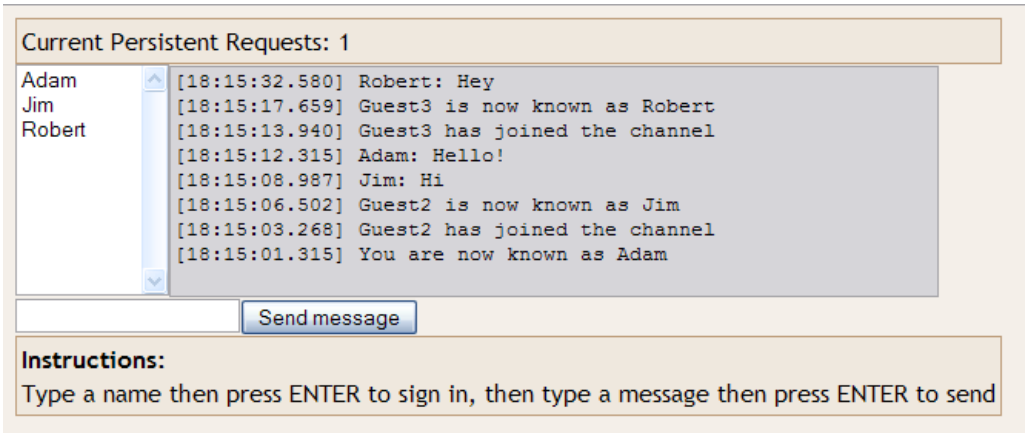


Figure 7.1: The final version of the Chatroom

7.1.2 Tetris

The 10 by 20 grid representing each player's game is stored as a two-dimensional integer array. Each cell is stored as an integer, which is then translated into a string referencing a CSS class. The location of the current block is stored as a set of four x, y coordinates. Only the integer array and the coordinates are sent to the opponent, minimising the bandwidth requirements of the application. Asynchronous timers are run at an interval of 500 milliseconds, and are used to animate the screen. Due to the issue discussed in section 6.2.3, the game suffers from the presence of an extraneous `input` field at the bottom of the screen.

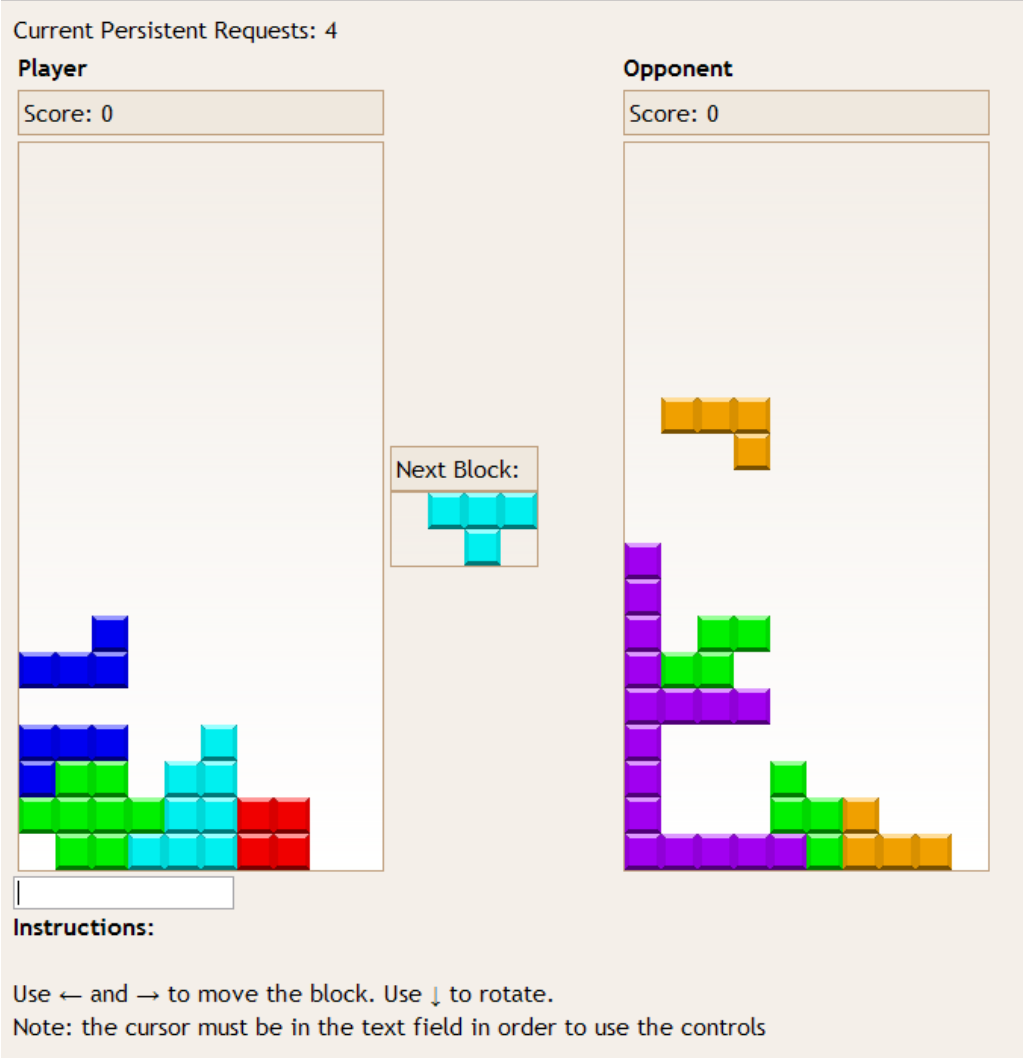


Figure 7.2: The final version of Tetris

7.1.3 Bomberman

Whilst expected to perform worse than the other two artefacts, the Bomberman game is significantly more complex. Due to the shared game environment, all processing must be conducted on the server in order to accurately and fairly distribute updates to each player.

Most importantly, the servlet partitions users into groups of four. Whilst the same chat room is shared between all concurrent users, a new Bomberman *game* is started for every fourth user which joins. Synchronisation is also very important; the server must check that all moves are legal. For example, two bombs cannot be laid on the same location concurrently; however, two clients may simultaneously attempt this in such a short space of time that their models are inaccurate. Rather than the client updating itself when the user performs an action, the client simply sends a *request* to perform that action. The server then checks the validity of the request and pushes revised data to each client; this behaviour is typical of online multiplayer games.

For each bomb which is laid, the servlet creates an asynchronous thread which determines when the bomb will explode. Upon completion, the thread tests which players have been killed, and invokes a server push to send this information. This asynchronous servlet threading operates in much the same was as interaction would in an SOA environment; thread synchronisation becomes increasingly important.

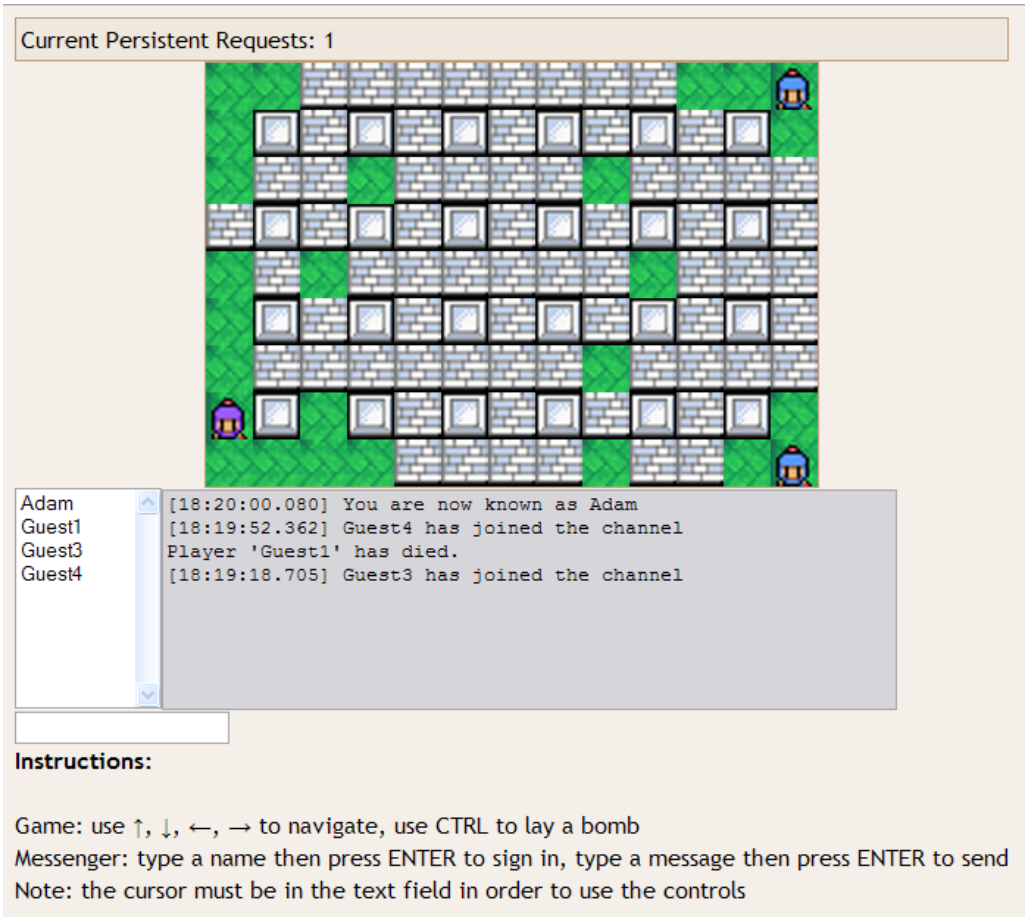


Figure 7.3: The final version of Bomberman

7.1.4 Framework

The framework itself is essentially a set of abstract classes fulfilling the minimum requirements set by the GWT compiler, the GWT API and the Tomcat API *(see appendix D)*. There are, additionally, several template classes such as `TimestampedEvent` and `ErrorMessageEvent`, which are not required in an application but are included as part of the framework to facilitate code reuse. The framework is packaged as a single sealed Java Archive (JAR) which is then shared between all three applications *(see figure 7.4)*. This type of packaging is important in order to maintain accurate revision control in deployed environments.

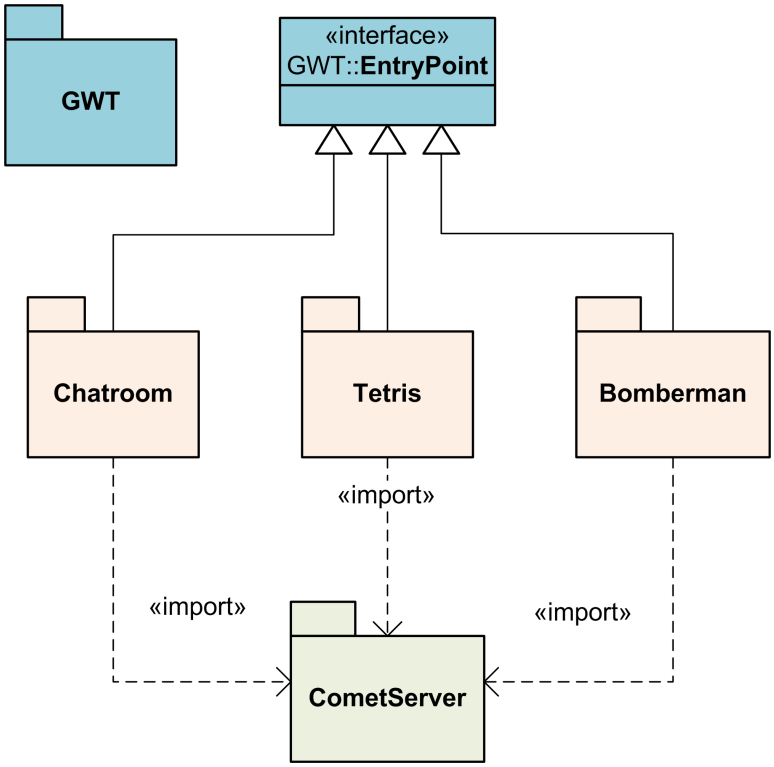


Figure 7.4: A UML package diagram illustrating how applications interact with the framework

7.2 User Evaluation

User evaluation is important in providing an independent and unbiased measurement of software quality. This study is being conducted as an evaluation of the framework, rather than of the software itself. As such, the focus will be on how the framework is utilised rather than how the applications perform.

7.2.1 Questionnaire

Construction

Whilst providing very useful results, the responses to the first questionnaire tended towards neutrality. For this reason, questions in the second questionnaire will be worded much more strongly, in order to illicit a broader range of responses. The quality of the artefacts will be the focal point of the evaluation questionnaire. Converse & Presser (1986) assert that numeric 1-10 scales are ideal for this purpose due to their familiarity with the general population. Users will be asked to evaluate

the responsiveness and performance of the applications, their visual appearance and whether they perform well enough to fulfil their specific roles. The inclusion of a separate question regarding visual appearance implicitly instructs respondents not to take this attribute into account in the other questions. Finally, users will be asked to assess the applications in comparison to existing software.

Due to the nature of the project, the questionnaire required concurrent use of the software by two users; the author supplied this interaction in order to provide a uniform experience to all participants. Identical instructions were provided to all users on the questionnaire page and in applications themselves. The applications were tested by each user over the internet, thus the latency during testing varied at between 50ms and 200ms; highly representative of response times to normal websites.

Results

Unfortunately, due to the duration of the questionnaire only 8 responses were gathered, meaning that data mining cannot be performed as it was for the original questionnaire. The standard deviation of responses was extremely low however, indicative that the results are unlikely to change significantly even if a larger sample were to be collected. Respondents were asked to assess their own expertise with regards to the subject matter *(see table E.15)*; the standard deviation of this value was very high further attesting to the validity of results despite the small sample *(see figure 7.5)*.

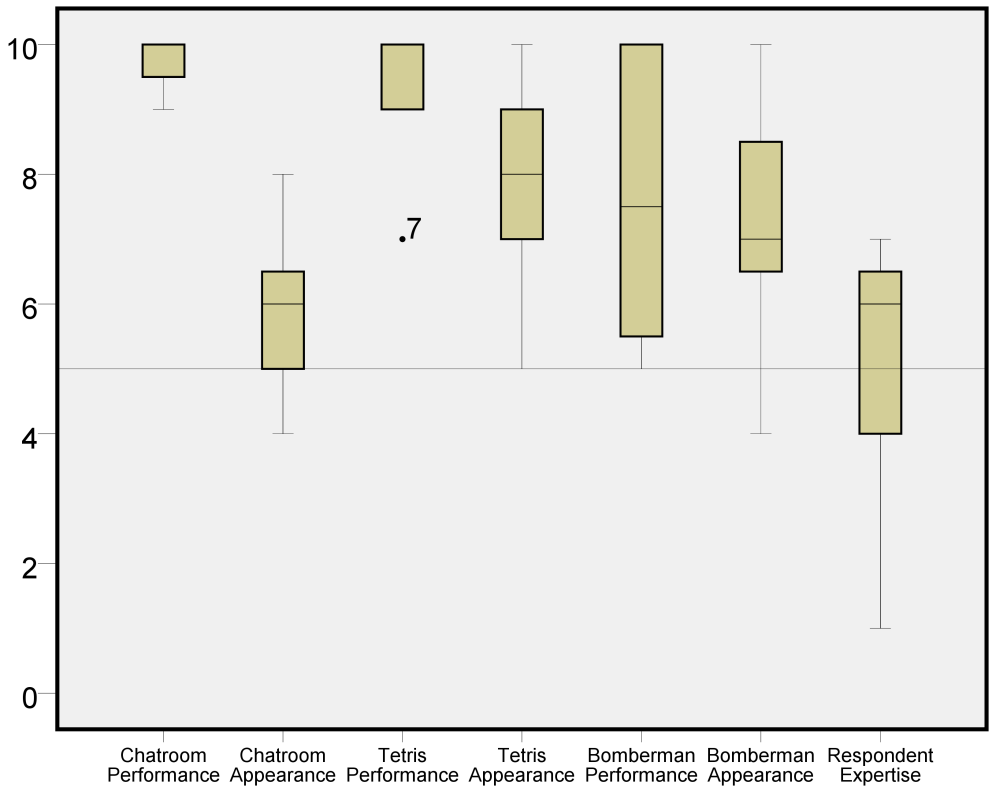


Figure 7.5: Box plots representing the numeric data gathered *(see tables E.1, E.2, E.5, E.6, E.8, E.9, E.15)*

Analysis

The performance of the Chatroom application was rated extremely positively, given 10/10 by all but two respondents, who gave it 9/10 instead *(7.5 & E.1)*. Users clearly agreed that the Chatroom fulfilled its role successfully *(E.4)*. The application was rated quite highly in comparison to existing software *(figure E.2)*; one respondent specifically mentioned a preference to the Chatroom application *(E.1)* over the MSN Web Messenger, which was evaluated during research *(see section 2.3.1)*. Users typically rated the visual appearance as mediocre *(E.2, figure E.1)*, the worst result out of the three. This was also expected as the project focused on functionality rather than appearance. The chatroom was expected to perform successfully, as evidenced by existing Comet implementations *(see sections 2.3.2 & 2.3.3)*.

The Tetris game also performed extremely well, averaging 9.38 on a scale of 1 to 10 measuring responsiveness *(E.5)*. It was rated the highest of the three applications in terms of visual appearance, scoring 7.88 *(E.6)*. As with the Chatroom, there was

general agreement that the Tetris game fulfilled its role *(E.7)*. The KeyboardListener issue investigated during the implementation *(see section 6.2.3)* caused problems for one user *(figure E.2)*, and three out of eight respondents agreed that the control mechanism for the games was 'restrictive' *(E.13)*. These results are very encouraging; they demonstrate the ability of the framework to mirror a dynamic data set in real-time across HTTP. Most information services, such as live sports scores, would not cause this much load on the framework.

As expected due to its networking requirements, the Bomberman game performed worst of the three, but still managed to score 7.63 for responsiveness *(E.8)*. The visual appearance was rated very similarly to that of the Tetris game, though responses varied greatly in this question, resulting in a standard deviation of nearly 2 points *(E.9)*. As with the Tetris game, users agreed that it fulfilled its role succesfully *(E.10)*.

Overall, users rated the games as performing 'quite well' in comparison to existing software *(E.11)*. Most interestingly, 50% of users rated the Chatroom as the best application, 25% the Tetris and 25% the Bomberman game *(E.14)*. The Bomberman game was expected to perform much worse in this test due to its high networking requirements compared to the other applications. This indicates that all three artefacts were successful in fulfilling their roles. That even the Bomberman game was rated by some as the best application may be evidence that the framework can handle even more intensive applications than those constructed during the project.

Surprisingly, these results indicate that the framework exceeds it requirements. Even though it was designed to stress the framework to its limits, the Bomberman game performed admirably. It would seem that the availability of low latency, high bandwidth internet connections facilitates the use of the internet for almost any purpose.

7.3 Technology Evaluation

7.3.1 Google Web Toolkit

Overall GWT performed its role incredibly well. The facilities offered by Eclipse and the GWT hosted mode browser made debugging extremely easy in comparison to normal JavaScript development. Moreover, a single implementation using GWT performed *almost* correctly in all of the browsers officially supported by GWT. The Chatroom application works correctly even on the Apple iPhone, a mobile platform which is not listed under GWT's supported clients, and typically has poor JavaScript

support.

Despite this, there were notable issues. As previously mentioned in section 6.2.3, no document level EventListener is available in GWT; a significant limitation. As mentioned by a user in the evaluation questionnaire *(figure E.1)*, the chatroom message listing scrolls the wrong way. This is due to a bug in GWT whereby scrolling does not occur in Firefox (Google, 2007). Despite being acknowledged by Google, this problem has gone unfixed for over two years.

GWT-RPC

Implementations using GWT-RPC carry a somewhat excessive amount of redundant code; the `Async` interface can be deduced entirely from the standard interface.

```
1 public String myMethod(String s);
```

```
1 public void myMethod(String s, AsyncCallback<String> callback);
```

The return type is always void, and a new parameter is added which is an AsyncCallback object typed as the return type of the original method. It would be better if one of these classes were to be generated automatically, though this would require changes to the development environment (i.e. the GWT 1.6 Eclipse plugin) in order to prevent errors.

GWT 1.6

During the project a new version of GWT was released, version 1.6. This version brings many significant changes, most notably replacing Tomcat with Jetty, introducing an Eclipse plugin and adopting a new EventHandler system. The use of a new servlet container was largely irrelevant due to the issues described in section 6.3.1; an external server was used in order to harness the Comet functionality of Tomcat. Nevertheless, it would have been unfeasible to switch to Jetty because it has a different Comet implementation than Tomcat: *Continuation* (Wilkins, 2008a). The introduction of a new EventHandler system may be evidence of the problems encountered in section 6.2.3, though the problem itself has still not been fixed. As mentioned in section 2.5, the procedure for creating a new project using GWT 1.5 is unintuitive and repetitive, this procedure is vastly improved in 1.6 as a result of the Eclipse plugin.

Most problematic however, was an issue regarding project linking in Eclipse. The framework was developed as a discrete project in order to completely segregate its codebase from that of the applications. As a result of this, the framework's project must be added to the build path of an application; this is not possible in GWT 1.6 as the compilation scripts have been replaced in favour of automatic handling by the Eclipse plugin. This issue was too significant to tolerate, as it would have required exporting the framework to a Java archive and relinking it to each of the three projects, every time the framework's code was changed. As with the aforementioned problems this has been acknowledged by Google, yet the issue remains (Google, 2009c).

CSS

Despite the excellent abstraction GWT provides for JavaScript, it provides no such facility for styling. Standard CSS must be written by the developer and this can cause just as many problems as with the code itself. The Tetris game, when viewed in IE7, does not render the game background correctly; the left-most column is the wrong colour. This problem is caused by the 'hacks' which GWT employs in order to provide cross-browser compatibility. Specifically, a superfluous `colgroup` element is added to the HTML tree; this is used by the JavaScript of GWT's ColumnFormatter class (which, incidentally, is not used in the applications), but serves no presentational purpose.

7.3.2 Apache Tomcat

Tomcat, the servlet container, performed extremely well throughout the project. Its performance was excellent, and through use of the NIO connector exceeded expectations *(see section 6.3.1)*. After implementing NIO, the GWT hosted mode development console could not be used, and so servlets had to be fully deployed before any testing could take place. One criticism, however, is that servlet exceptions are output to one of Tomcat's many text logs, rather than to a console of some kind. These logs have to be individually examined in order to diagnose a problem, something which added up to several hours over the duration of the project.

The incompatibility between Tomcat's `CometProcessor` and Jetty's `Continuation` was a contributory factor in rejecting the use of GWT 1.6. Both of these are non-blocking I/O connectors which do not waste a whole CPU thread per connection *(see figure 6.13)*. These APIs are being superceded by a universal implementation in

the servlet container 3.0 draft specification (Mordani, 2009, p10). This is, perhaps, evidence of the growing importance of Comet functionality.

7.4 Objectives

7.4.1 Server Push

The objective of facilitating server-push over HTTP was entirely successful and several innovative improvements were made over existing implementations. The use of asymmetric response RPC calls improves the efficiency of the implementation by up to 50% compared to existing implementations. The reimplementation of the shared SID stack *(see section 6.1.6)* used by tabbed browsing allows multiple connections from the same browser, something which is otherwise impossible. Unlike the two Comet examples evaluated in section 2.3, the framework does not enforce a timeout on client connections; this saves bandwidth as a single request can be held indefinitely.

7.4.2 Cross-browser Compatibility

Cross-browser compatibility, whilst generally achieved, was the most problematic objective. All three applications operate correctly in Firefox, Opera, Chrome, Safari and IE7+, but fail in IE6. Very few debugging tools exist for IE6, and those that do perform weakly at best; because of this, it was not possible to ascertain the cause of dysfunction in IE6.

In particular, CSS does not work as expected in Internet Explorer *(see section 6.2.4)*, and this is hard to accommodate for because the HTML tree is generated by GWT and not the developer. Ideally GWT should implement CSS properties as widget methods, taking the appropriate action using each of its 6 different JavaScript implementations in order to achieve the same result in each browser.

7.4.3 Code Reuse

The software was successfully developed as a framework, with no limitations placed upon applications which implement it. A complete separation of generic functionality was achieved; the applications only implement their core logic and that which is required by the GWT compiler.

Library Functions

A useful addition would be to develop a set of library functions which some, but not all, applications may have in common *(see figure 7.6)*. There are two major examples of this in the artefacts. Firstly, the Chatroom application is included in its entirety as part of the Bomberman application, and secondly the same 'grid' is used to animate both the Tetris and Bomberman games. Developing these components as part of a library would automatically mirror any improvements and changes in all applications.

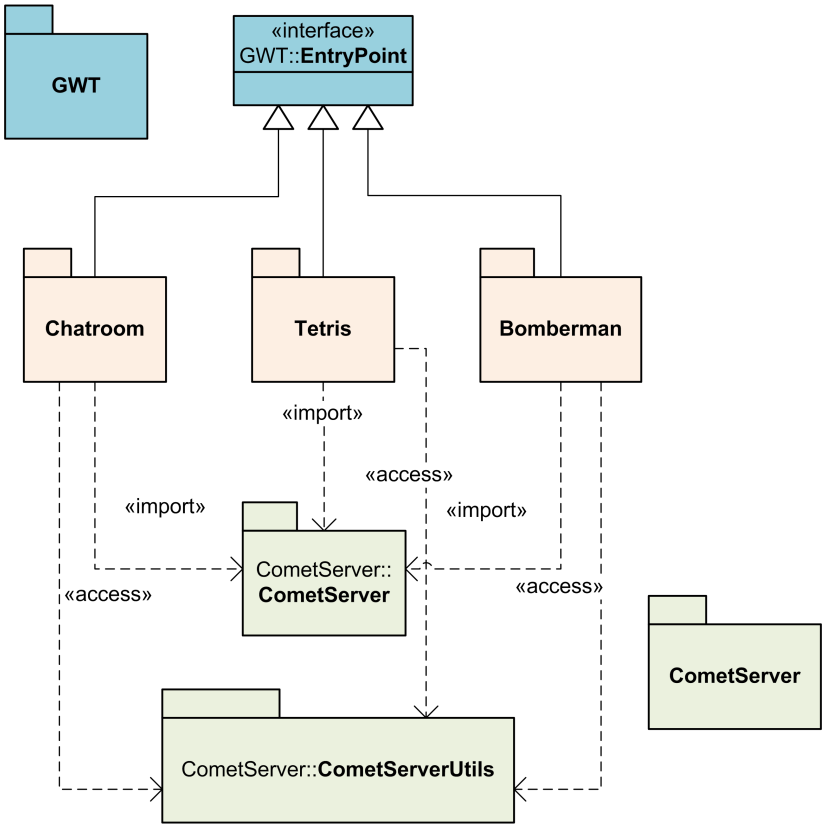


Figure 7.6: A UML package diagram illustrating how applications would interact with the framework and an additional library

7.4.4 Latency

Ideal latency was achieved through the use of Comet for server-push implementation. This minimal latency period is always achieved should enough connections be available to the server *(1)*; however, this is sometimes not the case, as the client must wait for persistent request connections to become available *(2)*.

r request

t timestamp in milliseconds (function)

p number of persistent requests permitted by the browser

l client-server round-trip latency period

a actual communication latency

1. $$\mathrm{t}(r_p) - \mathrm{t}(r_1) > p/l \implies \mathbf{a} = l$$

2. $$\mathrm{t}(r_p) - \mathrm{t}(r_1) < p/l \implies 2l \geq \mathbf{a} > l$$

Additionally, for *any* series $r_{1...p}$ requests, then $\mathbf{a}_{r_p+1} = t(r_p) - t(r_1) + l$.

The framework's Comet mechanism has no limit to the amount of events which can be sent in any one connection. This prevents a backlog of events from building up on the server, and is what prevents the communication latency from ever being more than twice the network latency.

7.4.5 Framework Interaction

Appendix D illustrates the classes which are required for an implementation of the framework. The framework classes (green) provide a layer of abstraction from the GWT and Tomcat APIs (blue and red respectively), reducing the complexity of an application. Whilst it is hard to speculate the ease with which an external developer could utilise the framework, no issues were encountered during the development process. Both the Tetris and Bomberman artefacts were created after the framework had been established, and few subsequent modifications had to be made to the framework in order to facilitate their vastly different functionality.

Chapter 8

Conclusions

8.1 Summary

As defined in the project requirements, the most strenuous application, the Bomberman game, perfomed successfully; ultimately the project was a complete success. The efficiency of the implementation exceeded expectations due to the successful implementation of asymmetric responses and the CometProcessor interface *(see sections and 5.2 and 6.3.1)*. Time management went well generally, however too much time was allocated to the development of the Chatroom in comparison to the other artefacts.

8.2 Artefacts

8.2.1 Instant Messaging

The most successful use of the framework was that of the Chatroom. This could easily be deployed on websites such as Facebook and MSN, which currently permit instant messaging, but only between two users rather than as a chatroom.

8.2.2 Tetris & Bomberman

Whilst the games did not perform quite as well, they demonstrated that the framework itself can handle significantly more load than is utilised by the Chatroom. The performance of the framework is ultimately limited only by a browser's HTTP connection model, which in most cases offers just two connections.

Animation

The most significant limitation of the Tetris and Bomberman artefacts is the quality of animation. Even the relatively simple animation employed can appear sluggish on old hardware and software, particularly in IE6 and the GWT hosted mode browser. Adobe Flash does not suffer from this limitation; because of its closer integration with the operating system, it is able to take advantage of hardware acceleration for 2D graphics. Flash wouldn't suffer from the styling issues which still exist in GWT *(see section 7.3.1)*. With the latest version of Adobe Flex, JavaScript and ActionScript can be seamlessly integrated using the ExternalInterface API (Key, 2009). Similarly, GWT provides the JavaScript Native Interface (JSNI) to extend method visibility to external code, such as native JavaScript or indeed Flash. The obfuscation normally administered by the GWT compiler makes JSNI necessary.

8.3 Limitations

The development of the framework went extremely well, however there are two significant limitations that exist which cannot be overcome. Firstly, the developer can make Ajax connections *outside* of the framework. This could cause issues, as the network architecture depends upon having at least two connections available to it. There doesn't seem to be any way to overcome this issue, aside from modifications to the GWT compiler itself. Secondly, the GWT compiler understands certain class names to have special meaning *(see figure 6.1)*. This still requires a framework user to write a significant amount of redundant code which would ideally be handled by the GWT compiler.

Unfortunately, whilst GWT is an open source, it would not be practical for any organisation to make these type of changes to the GWT compiler. GWT is not an open source *project* per se, as external organisations have no input into its development. Any changes to the GWT source code, such as those mentioned, would be lost when a new version of GWT is released. GWT itself has no mechanism to make these changes at a high level.

8.4 Potential Improvements

8.4.1 Event Chronology

Should a response from the server be lost during transmission, it will be retransmitted by the TCP stack. However, if two separate calls were to be returned soon after each other, and one of these has to be retransmitted, they could arrive out of order. No issues with this have been identified, most likely because packet loss is very low; it could, however, be a problem if the framework were to be used in a production environment. The solution to this is simple - events would be assigned a number chronologically as they are serialised by the server. If they were then received out of order, the client could store them until their intended execution time.

8.4.2 Lobbies

A useful addition to the framework would be a lobby facility which encapsulates applications. This facility would show a list instances of an application which a user could join. For example in the Chatroom application, a list of chatrooms could be shown in place of the contact list; the user would then choose a chatroom to join. Having developed three applications, it seems possible to implement this function generically, whilst maintaining flexibility.

8.4.3 Tabbed Browsing

The shared SID across tabs represented a serious implementation issue which required considerable work to overcome. It would be extremely useful if the HTTP specification were to be updated with regards to tabbed browsing, allowing different tabs on the same client to be differentiated by the server.

As mentioned in section 6.1.2, some clients allow more than the standard connection count of two connections. Unfortunately, this connection limit is shared across all tabs in tabbed browsers. Using the SID *(see section 6.1.6)* the system should be able to prevent a user from attempting to use more than one application at the same time due to the degradation in performance it would cause.

8.4.4 Client Event Buffer

Theoretically, an event 'storm' could occur, whereby events are generated by a client faster than they can be dispatched. The GWT-RPC framework opens a new connection for each RPC made by the client; should the browser already be at its connection

limit, these are transparently buffered until sent. A better implementation would be to create a wrapper for the GWT-RPC client framework which buffers *events* rather than the *connections* which are buffered by the browser. Upon availiability of a single connection, the entire event backlog could be cleared; though as above, this would require the knowledge and certainty of the client's persistent connection limit.

8.4.5 Cleanup

Though the framework implements a cleanup method which is called immediately preceding client close, situations still occur where clients do not disconnect gracefully. A solution to this would be to run an asynchronous servlet thread which checks intermittently for disconnected clients. This level of fault-tolerance would be extremely valuable in a production environment.

8.4.6 Componentisation of Artefacts

Whilst integration of the Chatroom functionality with the Bomberman application did not take long, it could ideally be accomplished using a single line of code. Having attained a comprehensive understanding of GWT, Tomcat and Comet, it now seems possible to package each application as a component. This high level componentisation would drastically simplify the use of the framework by external developers.

8.5 Google Web Toolkit

A multitude of significant issues were encountered with GWT. Whilst these were all surmountable, it raises questions about its suitability in a production environment. Several known issues have gone unfixed in GWT for a very long period, and infrequent release cycles may mean this pattern continues. However, most importantly no issues whatsoever were encountered in GWT-RPC, the crucial component with which the framework is constructed. Utimately, the framework could be used in conjunction with other JavaScript libraries such as those mentioned in section 2.1.1, or indeed another RIA technology such as Flash (Key, 2009). The JavaScript security model prevents access to any data outside the browser 'sandbox'. Whilst necessary, this restriction prevents certain types of applications, such as webcam conversations, from utilising the architecture of the framework. It is on this point that third party plugins, such as Flash and Silverlight, can fill the void; indeed webcam access is already a feature of Flash Player.

8.6 Possible Uses

Aside from the demonstrated uses of instant messaging and games, the framework could be used to replace existing systems based on inferior architecture. Current polling systems such as MSN Web Messenger *(section 2.3.1)* are extraordinarily inefficient, and due to the framework's employment of asymmetric responses it is even superior to existing Comet implementations. Many existing Java applications such as the Formula 1 live timing system (Formula1.com, 2009) could be ported to the framework. This would dramatically inrease the market penetration of such software. Another possible use of this functionality is that of a maintenance interface on a server; the Chatroom application could be easily adapted into a command-line shell.

The most interesting possibility though is that of collaboration. For instance, with two people in different locations talking over a draft document, it would be useful for the page to automatically scroll to the same location as the other viewer. GWT itself provides interesting possibilities for document viewing and editing through its RichTextArea widget. As demonstrated with the Tetris artefact, the mirroring of a dynamic data model is easily accomplished using the framework.

The web is still a rapidly evolving platform, with new technologies being released regularly. Google recently released a beta of O3D, a browser plugin that brings 3D accelerated graphics to JavaScript (Google, 2009b). This framework, in combination with O3D, could allow almost any game currently available to be adapted to the browser platform.

8.7 Final Thoughts

The project has demonstrated that a single framework can be used to develop Comet applications that work on a variety of platforms. With this in mind it seems strange that few implementations of Comet exist on the web. Perhaps it is the complexity of Comet development which makes the cost too high for most businesses. If that is the case, this framework could solve that issue. As for the technologies used, both GWT and the Java servlet specifications are rapidly evolving, and can be expected to show even further improvments in the future.

The high level markup languages used on the web, and user familiarity with the browser, make it an ideal application platform. The limitations of the HTTP protocol seem increasingly outdated though, and perhaps the time has come to revisit it with RIAs in mind.

Appendix A

Market Penetration of RIA Technologies

Source/Technology	Adobe[1]	RIA Statistics[2]	TheCounter.com[3]	W3Schools[4]
Flash	99.0%	96.99%	-	-
Java	81.0%	74.98%	90.9% to 97.7%	-
JavaScript	-	-	93.2%	95%
Silverlight	-	19.92%[5]	-	-

Table A.1: Market penetration statistics of prevalent RIA technologies

It is clear that Flash and JavaScript have the highest market penetration of RIA technologies. It is difficult to attain an accurate comparison, as no sources could be found which directly compared Flash and JavaScript. Whilst Flash may appear the clear winner from the results table, one source (riastats.com, 2009) shows that the latest version of the plugin has a market share of only 51%.

[1] Adobe-Millward Brown survey, September 2008, low sample size (4600), high margin of error (5%) (Adobe, 2008)

[2] RiaStats.com / DreamingWell.com, February 17th 2009, moderate sample size (24000) (riastats.com, 2009)

[3] TheCounter.com, February 2008 February 2009, high sample size (6m+) (TheCounter.com, 2009)

[4] W3Schools, January 2009 (W3Schools.com, 2009)

[5] Interestingly, market penetration of Silverlight significantly increased from 19.92% to 28.78% between 18 February 2009 and 8 May 2009

Appendix B

Background Research Questionnaire

Results are listed in the same order in which the questions were asked.

B.1 General

Table B.1: On average, how long do you spend a day using a web browser?

Value	Count	Percent %
Less than 10 minutes	1	1.28%
10 minutes to 30 minutes	1	1.28%
30 minutes to 1 hour	12	15.38%
1 hour to 3 hours	29	37.18%
More than 3 hours	35	44.87%

Table B.2: Which browsers do you have installed currently?

Value	Count	Percent %
Internet Explorer	58	74.36%
Firefox	58	74.36%
Safari	30	38.46%
Opera	7	8.97%
Chrome	5	6.41%
Cruz	1	1.28%

Table B.3: Which browser do you use most often?

Value	Count	Percent %
Firefox	40	51.28%
Internet Explorer	23	29.49%
Safari	13	16.67%
Chrome	2	2.56%

Table B.4: "If a website does not work correctly in my browser, I usually leave the site rather than use another browser"

Value	Count	Percent %
Strongly Agree	25	32.05%
Agree	30	38.46%
Neutral	9	11.54%
Disagree	13	16.67%
Strongly Disagree	1	1.28%

Table B.5: "Which of the following have you used in the past or use currently?"

Plugin	Currently Installed	Used In Past	Never Used	Never Heard Of
Adobe Flash Player	87.2%	11.5%	1.3%	-
Adobe Shockwave	29.5%	37.2%	21.8%	11.5%
Apple Quicktime	64.1%	21.8%	10.3%	3.8%
Java Browser Plugin	55.1%	16.7%	15.4%	12.8%
Microsoft Silverlight	14.1%	3.8%	17.9%	64.1%
Average	50.0%	18.2%	13.3%	18.5%

Table B.6: "I do not like having to install third party plugins" (such as those listed above)

Value	Count	Percent %
Strongly Agree	8	10.26%
Agree	27	34.62%
Neutral	26	33.33%
Disagree	11	14.10%
Strongly Disagree	6	7.69%

Table B.7: "I install third party plugins because I have no choice"

Value	Count	Percent %
Strongly Agree	17	21.79%
Agree	44	56.41%
Neutral	11	14.10%
Disagree	6	7.69%
Strongly Disagree	0	0.00%

Table B.8: Which of the following do you prefer?

Value	Count	Absolute %	Adjusted %
AJAX Websites	26	33.33%	49.06%
Don't Know	25	32.05%	N/A
Flash Websites	11	14.10%	20.75%
Plain HTML websites	16	20.51%	30.19%

Table B.9: "I have problems using Back/Forward/Refresh/Bookmarking on AJAX Websites"

Value	Count	Percent %
Strongly Agree	2	2.56%
Agree	6	7.69%
Neutral	7	8.97%
Disagree	20	25.64%
Strongly Disagree	4	5.13%
Don't Know	39	50.00%

B.2 Instant Messaging

Table B.10: In general, how do you rate desktop based messengers? (such as Windows Live Messenger, Skype)

Value	Count	Percent %
Very Good	19	24.36%
Good	44	56.41%
Average	12	15.38%
Bad	1	1.28%
Very Bad	0	0.00%
Never used them	2	2.56%

Table B.11: In general, how do you rate browser based messengers? (such as Facebook Chat, MSN Web Messenger, Google Chat)

Value	Count	Percent %
Very Good	6	7.69%
Good	23	29.49%
Average	27	34.62%
Bad	20	25.64%
Very Bad	1	1.28%
Never used them	1	1.28%

Table B.12: "The performance of browser based messengers is acceptable."

Value	Count	Percent %
Strongly Agree	3	3.85%
Agree	38	48.72%
Neutral	21	26.92%
Disagree	13	16.67%
Strongly Disagree	0	0.00%
Never used them	3	3.85%

Table B.13: "I only use browser based messengers when I cannot use desktop based messengers" (such as on a shared computer)

Value	Count	Percent %
Strongly Agree	14	17.95%
Agree	16	20.51%
Neutral	14	17.95%
Disagree	25	32.05%
Strongly Disagree	4	5.13%
Never used them	5	6.41%

Table B.14: "I do not like having to keep my browser open in order to continue using browser based messengers"

Value	Count	Percent %
Strongly Agree	8	10.26%
Agree	29	37.18%
Neutral	15	19.23%
Disagree	21	26.92%
Strongly Disagree	1	1.28%
Never used them	4	5.13%

B.3 Browser-Based Games

Table B.15: Have you ever played a browser based game? (such as a Flash game)

Value	Count	Percent %
Yes	56	71.79%
No	22	28.21%

Table B.16: In general, how do you rate browser based games?

Value	Count	Percent %
Very Good	2	2.56%
Good	29	37.18%
Average	28	35.90%
Bad	1	1.28%
Very Bad	0	0.00%
Never played one	18	23.08%

Table B.17: "Browsers are only good enough for very simple games"

Value	Count	Percent %
Strongly Agree	4	5.13%
Agree	28	35.90%
Neutral	12	15.38%
Disagree	11	14.10%
Strongly Disagree	2	2.56%
Don't Know	21	26.92%

Table B.18: Have you ever played a "real-time" multiplayer browser game? (such as Poker or Chess)

Value	Count	Percent %
Yes	26	33.33%
No	52	66.67%

Table B.19: How did you rate that game? (or the most recent)

Value	Count	Percent %
Very Good	2	2.56%
Good	19	24.36%
Average	7	8.97%
Bad	0	0.00%
Very Bad	0	0.00%
Never played a multiplayer browser game	50	64.10%

Table B.20: What technology did that game use? (or the most recent one)

Value	Count	Percent %
Never played one	42	53.85%
Don't Know	19	24.36%
Flash	12	15.38%
Java	2	2.56%
Javascript	2	2.56%
AJAX	1	1.28%

B.4 Other Browser-Based Applications

Table B.21: Do you use any *other* browser based "desktop replacements", such as Google Documents, on a *regular* basis?

Value	Count	Percent %
Yes	13	16.67%
No	65	83.33%

Table B.22: How would you rate the performance (speed, responsiveness) of those applications?

Value	Count	Percent %
Very Good	1	1.28%
Good	11	14.10%
Average	6	7.69%
Bad	2	2.56%
Very Bad	0	0.00%
Don't use them	58	74.36%

Table B.23: "Browser based 'desktop replacements' are missing important features, preventing me from using them."

Value	Count	Percent %
Strongly Agree	1	1.28%
Agree	6	7.69%
Neutral	6	7.69%
Disagree	8	10.26%
Strongly Disagree	2	2.56%
Don't Know	55	70.51%

B.5 Demographic

Table B.24: What is your gender?

Value	Count	Percent %
Male	26	33.33%
Female	52	66.67%

Table B.25: What is your age?

Value	Count	Percent %
0-17	2	2.56%
18-20	35	44.87%
21-23	36	46.15%
24-26	2	2.56%
27+	3	3.85%

Appendix C

Questionnaire Data Mining Results

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     result
Instances:    78
Attributes:   33
              browserusage
              hasInternetExplorer
              hasFirefox
              hasSafari
              hasChrome
              hasOpera
              primaryBrowser
              websiteCorrectly
              hasFlash
              hasShockwave
              hasQuicktime
              hasJava
              hasSilverlight
              like3PP
              nochoice3PP
              favType
              ajaxProblems
              desktopMessenger
              browserMessenger
              browserMessengerAcceptable
              browservsDesktop
```

```
              browserWindowOpen
              playedBrowserGame
              browsergameRating
              browserSimpleGame
              realtimeBrowserGame
              realtimeRating
              realtimeTech
              browserApps
              browserAppsRating
              browserFeatures
              gender
              age
Test mode:    5-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

hasFlash = Currently Installed
|   hasJava = Currently Installed
|   |   browserMessenger = 1: Male (1.0)
|   |   browserMessenger = 2: Female (13.0/4.0)
|   |   browserMessenger = 3: Male (16.0/3.0)
|   |   browserMessenger = 4: Female (7.0/1.0)
|   |   browserMessenger = 5: Female (3.0/1.0)
|   |   browserMessenger = Never used them: Male (1.0)
|   hasJava = Used In Past
|   |   realtimeBrowserGame = Yes: Male (3.0)
|   |   realtimeBrowserGame = No: Female (8.0/2.0)
|   hasJava = Never Used: Female (9.0)
|   hasJava = Never Heard Of: Female (7.0)
hasFlash = Used In Past: Female (9.0)
hasFlash = Never Used: Female (1.0)
hasFlash = Never Heard Of: Female (0.0)

Number of Leaves  :  13

Size of the tree :  17


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
```

```
=== Summary ===

Correctly Classified Instances          51               65.3846 %
Incorrectly Classified Instances        27               34.6154 %
Kappa statistic                          0.1474
Mean absolute error                      0.371
Root mean squared error                  0.5169
Relative absolute error                 83.1058 %
Root relative squared error            109.5615 %
Total Number of Instances               78

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.308     0.173      0.471     0.308     0.372      0.575    Male
                 0.827     0.692      0.705     0.827     0.761      0.575    Female
Weighted Avg.    0.654     0.519      0.627     0.654     0.631      0.575

=== Confusion Matrix ===

  a  b   <-- classified as
  8 18 |  a = Male
  9 43 |  b = Female
```

Appendix D

Framework Class Diagram

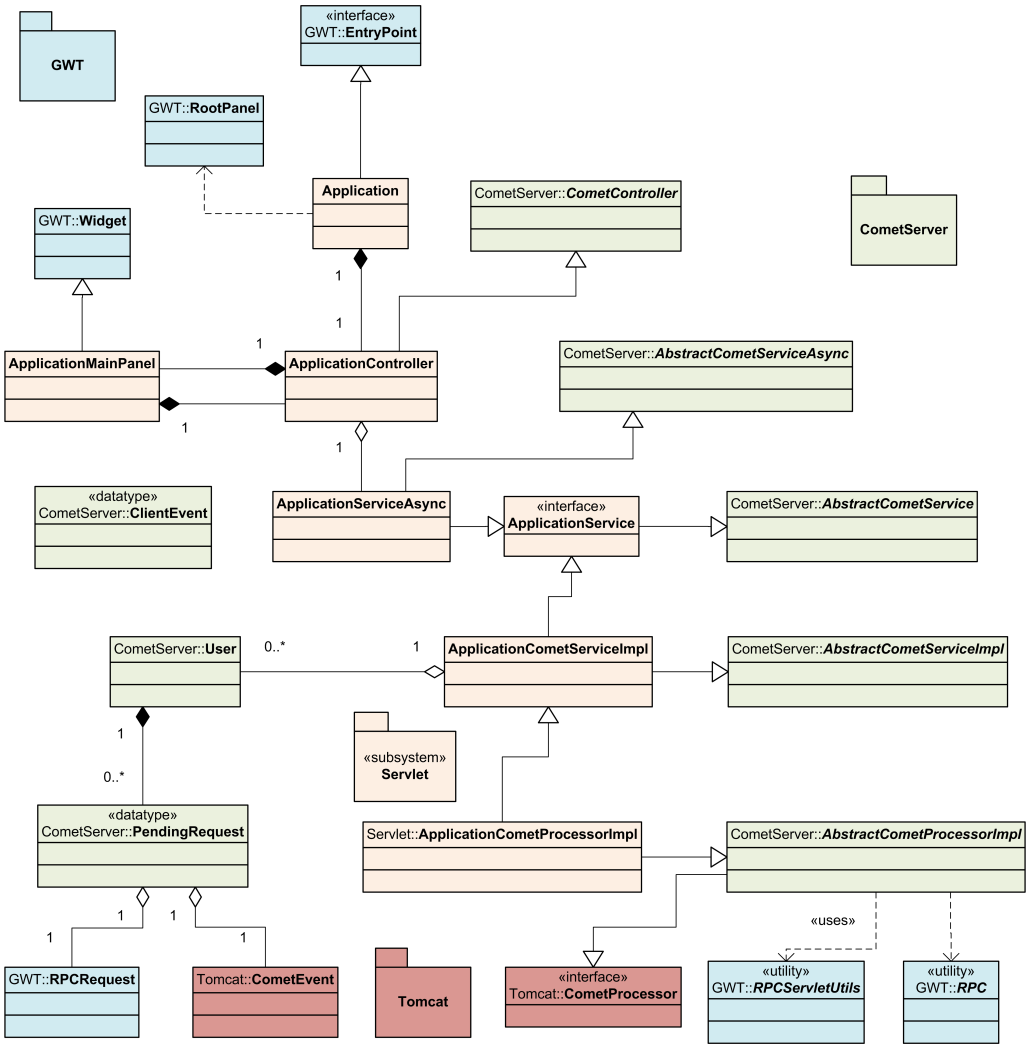


Figure D.1: A generic implementation of the framework. Hotspots are shown in orange.

Appendix E

User Evaluation Questionnaire

Results are listed in the same order in which the questions were asked.

E.1 Chatroom

Table E.1: How do you rate the responsiveness (latency) and general performance of the Chatroom?

Value	Votes	Proportion
10 (Best)	6	75.00%
9	2	25.00%
8	0	0.00%
7	0	0.00%
6	0	0.00%
5	0	0.00%
4	0	0.00%
3	0	0.00%
2	0	0.00%
1 (Worst)	0	0.00%

Average: 9.75 / 10.00

Standard Deviation: 0.463

Table E.2: How do you rate the visual appearance of the Chatroom?

Value	Votes	Proportion
10 (Best)	0	0.00%
9	0	0.00%
8	1	12.50%
7	1	12.50%
6	3	37.50%
5	2	25.00%
4	1	12.50%
3	0	0.00%
2	0	0.00%
1 (Worst)	0	0.00%

Average: 5.88 / 10.00

Standard Deviation: 1.246

Table E.3: How does the Chatroom perform in comparison to other browser-based instant messengers you have used?

Value	Count	Percent %
Very Well	3	37.50%
Quite Well	3	37.50%
Average	2	25.00%
Quite Bad	0	0.00%
Very Bad	0	0.00%

Table E.4: "The Chatroom performs well enough to have meaningful conversations."

Value	Count	Percent %
Strongly Agree	4	50.00%
Agree	4	50.00%
Neutral	0	0.00%
Disagree	0	0.00%
Strongly Disagree	0	0.00%

Figure E.1: Please offer any further comments regarding the Chatroom.

User A "Very impressive to see real-time response demonstrated. Would like a more engaging interface and visual appearance."

User B "Text box where you type the message could be bigger"

User C "New messages could do with appearing from bottom, not top. Events should stand out from messages possibly."

User D "better than webmessenger by msn"

E.2 Tetris game

Table E.5: How do you rate the responsiveness (latency) and general performance of the Tetris game?

Value	Votes	Proportion
10 (Best)	5	62.50%
9	2	25.00%
8	0	0.00%
7	1	12.50%
6	0	0.00%
5	0	0.00%
4	0	0.00%
3	0	0.00%
2	0	0.00%
1 (Worst)	0	0.00%

Average: 9.38 / 10.00

Standard Deviation: 1.061

Table E.6: How do you rate the visual appearance and animation of the Tetris game?

Value	Votes	Proportion
10 (Best)	2	25.00%
9	0	0.00%
8	3	37.50%
7	2	25.00%
6	0	0.00%
5	1	12.50%
4	0	0.00%
3	0	0.00%
2	0	0.00%
1 (Worst)	0	0.00%

Average: 7.88 / 10.00

Standard Deviation: 1.642

Table E.7: "The Tetris game performs well enough to have meaningful matches."

Value	Count	Percent %
Strongly Agree	3	37.50%
Agree	3	37.50%
Neutral	1	12.50%
Disagree	1	12.50%
Strongly Disagree	0	0.00%

Figure E.2: Please offer any further comments regarding the Tetris game.

User A "Again, like the messenger this is very responsive with zero latency demonstrated."

User B "Having cursor in text box is annoying."

User E "nice game!"

E.3 Bomberman game

Table E.8: How do you rate the responsiveness (latency) and general performance of the Bomberman game?

Value	Votes	Proportion
10 (Best)	3	37.50%
9	0	0.00%
8	1	12.50%
7	1	12.50%
6	1	12.50%
5	2	25.00%
4	0	0.00%
3	0	0.00%
2	0	0.00%
1 (Worst)	0	0.00%

Average: 7.63 / 10.00

Standard Deviation: 2.200

Table E.9: How do you rate the visual appearance and animation of the Bomberman game?

Value	Votes	Proportion
10 (Best)	2	25.00%
9	0	0.00%
8	0	0.00%
7	4	50.00%
6	1	12.50%
5	0	0.00%
4	1	12.50%
3	0	0.00%
2	0	0.00%
1 (Worst)	0	0.00%

Average: 7.25 / 10.00

Standard Deviation: 1.982

Table E.10: "The Bomberman game performs well enough to have meaningful matches."

Value	Count	Percent %
Strongly Agree	3	37.50%
Agree	3	37.50%
Neutral	1	12.50%
Disagree	1	12.50%
Strongly Disagree	0	0.00%

Figure E.3: Please offer any further comments regarding the Bomberman game.

User A "While you could have proper matches on this one some latency was evident although not enough to detract from the experience."

User B "Having cursor in text box gets a bit annoying"

User C "Just a bit too slow I'm afraid"

User E "A fun game!"

User F "slightly laggy"

E.4 General

Table E.11: How do the games perform in comparison to other browser-based multiplayer games you have played?

Value	Count	Percent %
Very Well	0	0.00%
Quite Well	4	50.00%
Average	1	12.50%
Quite Bad	1	12.50%
Very Bad	0	0.00%
Not Played Any	2	25.00%

Table E.12: "The applications are let down by their visual appearance."

Value	Count	Percent %
Strongly Agree	0	0.00%
Agree	1	12.50%
Neutral	1	12.50%
Disagree	5	62.50%
Strongly Disagree	1	12.50%

Table E.13: "The control mechanism for the games is restrictive."

Value	Count	Percent %
Strongly Agree	0	0.00%
Agree	3	37.50%
Neutral	0	0.00%
Disagree	5	62.50%
Strongly Disagree	0	0.00%

Table E.14: Which application performs its role most effectively?

Value	Count	Percent %
Chatroom	4	50.00%
Tetris	2	25.00%
Bomberman	2	25.00%

Table E.15: How would you rate your expertise with regard to browser-based games?

Value	Votes	Proportion
10 (Best)	0	0.00%
9	0	0.00%
8	0	0.00%
7	2	25.00%
6	3	37.50%
5	0	0.00%
4	2	25.00%
3	0	0.00%
2	0	0.00%
1 (Worst)	1	12.50%

Average: 5.00 / 10.00

Standard Deviation: 2.160

Figure E.4: Any further comments?

User A "Very impressive Messnger app because of the lack of latency, the two games can be improved upon with some extra attention to the control mechanisms (clicking in boxes etc.). Overall however, this was very interesting to see and compares favourably to all browser-based games I've experienced."

Appendix F

Documentation

This document was compiled with LaTeX using the `hyperref` package; in the digital version internal references (such sections in the contents, and inline citations) are clickable hyperlinks.

F.1 Directory Structure

Software Off-the-peg software used to run the applications. The software is all free and redistribution is permitted.

Artefacts The compiled applications which run on the Tomcat server.

Workspace The Eclipse workspace; contains the source code and GWT compile batch files.

F.2 Software

All of the software used during the project is cross-platform, however only the Windows version were used and tested with the software. Versions of the software for other operating systems are included in the `Software` directory.

Name	Win32
Apache Tomcat	apache-tomcat-6.0.18.exe
Google Web Toolkit (GWT)	gwt-windows-1.5.3.zip
Eclipse IDE for Java Developers	eclipse-java-ganymede-SR2-win32.zip
Java Development Kit	jdk-6u13-windows-i586-p.exe
WEKA	weka-3-6-0.exe

F.2.1 Installation Instructions

The Java JDK must be installed in order to run Eclipse, GWT and Tomcat. Version 6 should be used, though it is possible that version 5 would work. Version 6u13 was used during development.

Apache Tomcat should be installed using the "native" option; this gives support for the APR Connector. GWT and Eclipse can simply be extracted to a directory, no installation is required.

The `Workspace` directory should be copied to a local drive as write access is required.

WEKA was used for the link analysis shown in appendix C. It is not required in order to test the artefacts.

Tomcat Configuration

The `gwt-servlet.jar` file, included with GWT (also located in `Software/Apache`) should be added to the `Tomcat 6.0/lib` directory in your Tomcat installation.

The `CometServer_v3.0.jar` file, the central artefact of the project should also be added to the `Tomcat 6.0/lib` directory.

The `Tomcat 6.0/conf/server.xml` file must be edited. Add the following anywhere inside the <`Service name="Catalina"`> element:

```
<Connector
   connectionTimeout="20000"
   port="8081"
   protocol="org.apache.coyote.http11.Http11NioProtocol"
   maxThreads="1"
   acceptorThreadCount="2"
   redirectPort="8443"
   socket.directBuffer="false" />
```

Alternatively, replace your config file with the one located in `Software/Apache`.

F.2.2 Deploying the Artefacts

Delete the existing `ROOT` directory from the `webapps` folder in your Tomcat installation directory, then simply copy the contents of `Artefacts/webapps` to it.

F.2.3 Running The Software

GWT Hosted Mode

The `AjaxTester` (section 6.1.2 of the report) and `FlawedMessenger` (section 6.1.6) must be run in GWT Hosted Mode. Run Eclipse and select the following:

1. Run menu
2. Run Configurations...
3. AjaxTester / FlawedMessenger
4. Run

Due to limitations in GWT, libraries must be linked statically to the IDE. It may be necessary to update the project references to the GWT JARs, which can be found in the GWT installation package.

F.2.4 The Main Artefacts

To access the main artefacts, visit the following URL:

http://localhost:8081/

Note: the port **must** match the one used in the NIO connector listed above.

If you are testing using a browser with tabs, please close all other tabs that are accessing the server. This issue is discussed extensively in the report.

F.2.5 Compilation

If for any reason you need to recompile the artefacts, run the appropriate compile script, e.g.:

`Workspace/CometServerMessenger/CometServerMessenger-compile.cmd`

Note: you will need to update the paths in this batch file; they must be specified as an absolute path due to GWT limitations.

Bibliography

References

Adobe. 2008 (December). *Flash Player Penetration*. `http://www.adobe.com/products/player_census/flashplayer/`. [Accessed 18 February 2009].

Adobe. 2009 (February). *Adobe Labs: Flash Player 10*. `http://labs.adobe.com/downloads/flashplayer10.html`.

Apache Software Foundation. 2008a. *Apache Tomcat 6.0 - Advanced IO and Tomcat*. `http://www.mbaworld.com/docs/aio.html`.

Apache Software Foundation. 2008b. *CometProcessor (Tomcat API Documentation)*. `http://tomcat.apache.org/tomcat-6.0-doc/api/org/apache/catalina/CometProcessor.html`.

Beck, Kent, Beedle, Mike, van Bennekum, Arie, Cockburn, Alistair, Cunningham, Ward, Fowler, Martin, Grenning, James, Highsmith, Jim, Hunt, Andrew, Jeffries, Ron, Kern, Jon, Marick, Brian, Martin, Robert C., Mellor, Steve, Schwaber, Ken, Sutherland, Jeff, & Thomas, Dave. 2001 (February). `http://agilemanifesto.org/`.

Boehm, Barry W. 1988. A Spiral Model of Software Development and Enhancement. *Computer*, **21**(5), 61–72.

Burnette, Ed. 2006. *Google Web Toolkit: Taking the pain out of Ajax*. Raleigh, North Carolina: The Pragmatic Bookshelf.

Cerf, Vinton, Dalal, Yogen, & Sunshine, Carl. *RFC 675 - Specification of Internet Transmission Control Program*. `http://tools.ietf.org/html/rfc675`.

Chaganti, Prabhakar. 2007. *Google Web Toolkit: GWT Java AJAX Programming*. Birmingham: PACKT.

Cole, Jeffrey. 2009 (April). *2009 Digital Future Project Highlights*. http://www.digitalcenter.org/pdf/2009_Digital_Future_Project_Release_Highlights.pdf.

Converse, Jean M., & Presser, Stanley. 1986. *Survey Questions: Handcrafting the Standardized Questionnaire*. Thousand Oaks, London: Sage Publications.

Crane, Dave, Pascarello, Eric, & James, Darren. 2006. *Ajax In Action*. Greenwich: Manning.

Deitel, P., & Deitel, H. 2008. *AJAX, Rich Internet Applications, and Web Development For Programmers*. Upper Saddle River: Prentice Hall.

Dekker, Marcel Douwe. 2008 (October). http://en.wikipedia.org/wiki/File:DSDM_Development_Process.svg.

Dewsbury, Ryan. 2008. *Google Web Toolkit Applications*. Upper Saddle River: Prentice Hall.

Fain, Yakov, Rasputnis, Dr. Victor, Tartakovsky, Anatole, & Eckel, Bruce. 2007. *Rich Internet Applications with Adobe Flex & Java*. Woodcliff Lake: SYS-CON Books.

Fielding, Roy, Irvine, UC, Gettys, J., Compaq/W3C, Mogul, J., Compaq, Frystyk, H., W3C/MIT, Masinter, L., Xerox, Leach, P., Microsoft, Berners-Lee, Tim, & W3C/MIT. 1999 (June). *Hypertext Transfer Protocol – HTTP/1.1*. http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html.

Formula1.com. 2009. *Formula 1 Live Timing*. http://www.formula1.com/services/live_timing/live_timing.html.

Gay, Jonathan. *The History of Flash*. http://www.adobe.com/macromedia/events/john_gay/.

Google. 2007. *Issue 906: TextArea setCursorPos does not work in Firefox*. http://code.google.com/p/google-web-toolkit/issues/detail?id=906&can=1&q=textarea.

Google. 2008a. *Creating a GWT Project*. http://code.google.com/webtoolkit/tutorials/1.5/create.html. [Accessed 6 April 2009].

Google. 2008b. *JRE Emulation Reference - Google Web Toolkit 1.5*. http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=RefJreEmulation. [Accessed 6 April 2009].

Google. 2008c. *RPC Plumbing Diagram - Google Web Toolkit 1.5*. http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=DevGuidePlumbingDiagram.

Google. 2008d. *RPCRequest (Google Web Toolkit Javadoc)*. http://google-web-toolkit.googlecode.com/svn/javadoc/1.5/com/google/gwt/user/server/rpc/RPCRequest.html.

Google. 2008e. *Serializable Types - Google Web Toolkit 1.5*. http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=DevGuideSerializableTypes.

Google. 2008f. *Tutorial - Google Web Toolkit 1.5*. http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=GettingStartedTutorial.

Google. 2009a. *Issue 1442: Add document level keyboard listener*. http://code.google.com/p/google-web-toolkit/issues/detail?id=1442.

Google. 2009b. *O3D API*. http://code.google.com/apis/o3d/.

Google. 2009c. *Project Linking Issue in GWT 1.6*. http://groups.google.com/group/Google-Web-Toolkit/browse_thread/thread/951499c5773693c9.

GPokr.com. 2006. http://www.gpokr.com.

Harmon, James E. 2008. *Dojo: Using the Dojo JavaScript Library to Build Ajax Applications*. Addison-Wesley Professional.

Heller, Martin. 2009 (February). *AJAX and RIA 2009: More Choices, Tough Decisions*. AjaxWorld Magazine http://ajax.sys-con.com/node/739931.

Hickson, Ian, & Hyatt, David. 2009. *HTML 5*. http://dev.w3.org/html5/spec/Overview.html.

Hollar, Ashby Brooks. 2006. *Cowboy: An Agile Programming Methodology for a Solo Programmer*. M.Phil. thesis, Virginia Commonwealth University. http://hdl.handle.net/10156/1400.

Jacobs, Ian, & Walsh, Norman. 2004 (December). *Architecture of the World Wide Web, Volume One*. http://www.w3.org/TR/2004/REC-webarch-20041215/.

Key, Charlie. 2009. *Building a simple interaction between Flex and JavaScript using the ExternalInterface API*. http://www.adobe.com/devnet/flex/articles/flex_javascript.html.

Koch, Alan S. 2005. *Agile Software Development: Evaluating the Methods for Your Organization*. Artech House.

Markiewicz, Marcus Eduardo, & Lucena, Carlos J.P. 2001. *Object Oriented Framework Development*.

McConnell, Steve. 1996. *Rapid Development: Taming Wild Software Schedules*. Redmond, WA, USA: Microsoft Press.

Mordani, Rajiv. 2009 (April). *Java Servlet Specification: Version 3.0 (Proposed Final Draft)*. http://jcp.org/aboutJava/communityprocess/pfd/jsr315/index.html.

MozillaZine. 2008 (March). *Network.http.max-persistent-connections-per-server*. http://kb.mozillazine.org/index.php?title=Network.http.max-persistent-connections-per-server&oldid=36055.

NetApplications. 2009 (April). *Browser Version Market Share*. http://marketshare.hitslink.com/browser-market-share.aspx?qprid=2&qpmr=40&qpdt=1&qpct=3&qptimeframe=M&qpsp=123. [Accessed 16 May 2009].

Nicholls, Michael E.R., Orr, Cathernine A., Okubo, Matin, & Loftus, Anread. 2006. Satisfaction Guaranteed: The Effect of Spatial Biases on Responses to Likert Scales. *Psychological Science*, **17**(12), 1027–1028.

Nutschan, Conrad. 2008 (April). http://en.wikipedia.org/wiki/File:Spiral_model_(Boehm,_1988).png.

Oppenheim, Abraham Naftali. 1992. *Questionnaire Design, Interviewing and Attitude Measurement*. Continuum International Publishing.

Raggett, Dave. 2005. *The Ubiquitous Web*. http://www.w3.org/2005/Talks/0621-dsr-ubiweb/.

Reiff, Frank. *Migrating from character-based interfaces to web-based interfaces*. http://www.comp.lancs.ac.uk/computing/research/cseg/projects/Intranet/FormstoWeb.pdf.

riastats.com. 2009 (February). *Rich Internet Application Statistics*. http://riastats.com/. [Accessed 18 February 2009].

Riehle, Dirk. 2000. *Framework Design: A Role Modeling Approach*. Ph.D. thesis, Swiss Federal Institute of Technology, Zurich. http://www.riehle.org/computer-science/research/dissertation/diss-a4.pdf.

Russell, Alex. 2006 (March). *Comet: Low Latency Data for the Browser*. http://alex.dojotoolkit.org/2006/03/comet-low-latency-data-for-the-browser/.

Smith, Paul. 2009 (March). http://en.wikipedia.org/wiki/File:Waterfall_model.svg.

Smith, Ric. 2007 (August). *AJAX, Flash, Silverlight, or JavaFX: Must We Choose?* AjaxWorld Magazine http://ajax.sys-con.com/node/417624.

Snook, Jonathan, Gustafson, Aaron, Langridge, Stuart, & Webb, Dan. 2007. *Accelerated DOM Scripting with Ajax, APIs and Libraries*. Berkely, CA, USA: Apress.

Sun Microsystems Inc. 2007a. *HttpServletRequest (Java EE 5)*. http://java.sun.com/javaee/5/docs/api/javax/servlet/http/HttpServletRequest.html.

Sun Microsystems Inc. 2007b. *Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications*. http://dlc.sun.com/pdf/819-2634/819-2634.pdf.

Sun Microsystems Inc. 2008a. *Thread Interference*. http://java.sun.com/docs/books/tutorial/essential/concurrency/interfere.html.

Sun Microsystems Inc. 2008b. *ThreadLocal (Java Platform SE 6)*. http://java.sun.com/javase/6/docs/api/java/lang/ThreadLocal.html.

TheCounter.com. 2009 (February). *JavaScript Stats*. http://www.thecounter.com/stats/2009/February/javas.php. [Accessed 18 February 2009].

W3Schools.com. 2009 (February). *Browser Statistics*. http://www.w3schools.com/browsers/browsers_stats.asp. [Accessed 18 February 2009].

Wilkins, Greg. 2008a. *Continuations - Jetty*. http://docs.codehaus.org/display/JETTY/Continuations.

Wilkins, Greg. 2008b (May). *Jetty vs Tomcat: A Comparative Analysis*. http://www.webtide.com/choose/jetty.jsp.

Additional Reading

McConnell, Steve. 2004. *Code Complete, Second Edition*. Redmond, WA, USA: Microsoft Press.

NetApplications. 2009b (February). *Top Browser Share Trend*. `http://marketshare.hitslink.com/report.aspx?qprid=1&qpdt=1&qpct=4&qptimeframe=M&qpsp=117&qpnp=24`. [Accessed 27 February 2009].

Pilone, Dan. 2006. *UML 2.0 Pocket Reference (Pocket Reference (O'Reilly))*. O'Reilly Media, Inc.

Wake, William C. 2000. *Extreme Programming Explored*. Addison-Wesley Professional.