VIRTUAL SPORT @ BBC
"VISUALISING LIVE SPORT EVENTS ON THE WEB"

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2013

By
Stefan Klikovits
School of Computer Science

Contents

Word Count: 19645

List of Tables

List of Figures

Abstract

VIRTUAL SPORT @ BBC
"VISUALISING LIVE SPORT EVENTS ON THE WEB"
Stefan Klikovits
A dissertation submitted to the University of Manchester
for the degree of Master of Science, 2013

There is a growing interest in the analysis of sports data. The reasons for these developments are various and range from improvement of training strategies to calculations for the sports betting industry. The BBC has television rights for a large number of sports events. For some events (e.g. certain football games) they not only receive the video and audio data, but also have access to a data stream, which is logging events on the pitch. This projects' goal was to utilise the data and evaluate possibilities for its presentation on computers and mobile devices, and the feasibility of such an application's implementation. In the course of this project a prototype called "VirtualSport" was developed as a proof-of-concept. This dissertation will first show that the data quality is high enough for the creation of appealing data visualisations that can be used to augment the user's live sports experience. An implementation will then provide an example for efficient data integration using data stream management principles. The dissertation further describes the usage of an event driven architecture to ensure server performance and HTML web sockets to implement "push"-technology for real-time updates of the clients using HTML5. To assure the simplified addition of new functionality, this thesis describes the implemented plugin architecture and proposes strategies for assuring the system's stability and correctness. As the application will have more than ten thousand users for individual sports events (*C10k problem*), an important part of the prototype is the evaluation of performance and optimisation strategies. This dissertation introduces details of these problems and provides information to overcome these challenges. Further discussion on accessing other data sources in the form of server-side mashups and addition of external information using proxy-style mashup technology is provided to show how the software can include other data sources.

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank the BBC Future Media department for this great opportunity.
I enjoyed working in an open, dynamic environment.

I would also like to thank Dr. Keith Mitchell and Matthew Clark for their support.
You have contributed a lot to the success of this project.

Further, I would like to express my gratitude to Timothy Furmston, who provided some of the required server infrastructure and generously helped setting it up.

A special thank you goes to my supervisor, Dr. Duncan Hull.
I am deeply grateful for all your help throughout the entire project.
I am convinced that without your assistance the project would not have been as successful.

Chapter 1

Introduction

1.1 Project background and involved parties

This project was a cooperation between the BBC Future Media department and the University of Manchester. Its aim was to evaluate the possibility of parsing live sport data streams (e.g. football games) and displaying extracted information such as current player positions, player heat maps or statistics tables for large numbers of concurrent users.

1.2 Project description

The subject of this project was the visualisation of live sports events on end user devices. The software created was an application that uses data of sports events the BBC has access to and displays informative and attractive visualisations. The application could be used to not only provide users with the video broadcast via television, web stream, BBC iPlayer[1], etc. but also to offer them a second screen[2] (e.g. a computer, tablet or smartphone) experience to access further information (i.e. statistics) about the current game.

The gathered information was represented in various forms. The most significant use case was the provision of detailed information on specific players including its visual representation (such as player heat maps, pass maps, shot maps, etc.).

After verification of the data quality and the amount of the available position data,

[1] more information on the *BBC iPlayer* can be found in [Koz08], on general *IPTV* in [XDZ+07] and [CRC+08]

[2] see Section 2.4 for details about second screens

it was also desirable to provide a live football experience showing where the individual players are in real-time and to display events like passes and shots. In this use case, the positions of players and the ball had to be transferred and updated in real time.

Although the primary data source was a live data stream (i.e. the Opta Sports Data [Opt13a] event stream), an interface for the usage of internal BBC player tracking tools[3] as an extension to the main data source has been provided. The system is also extensible to other (live) data providers, sources with static information (such as player profile pages), statistics from other domains (e.g. BBC sports data) and other external sources (e.g. Facebook pages/Twitter feeds).

A football pitch displaying the squads as team formations was another application of the gathered data. The users could click on e.g. a player icon if they wanted more information about this individual player or other buttons for game/team statistics. Another use case was to provide the representation of a game to people with very low bandwidths. Especially in bandwidth ranges, where it is not possible to watch the video data since the required data transfer amount is too high. For these users it is possible to e.g. listen to the radio stream and still be able follow the game visually using this application.

1.3 Research question

The research goal of this project was to evaluate the feasibility of an application that is accepted and used by customers for an augmented video experience.

This prototype project was carried out to provide answers to the following research tasks. The first one was regarding the quality and amount of available data. To fully support an application such as the one mentioned before, the data has to meet certain minimum requirements. Hence an upfront analysis of the data sources was necessary.

After this evaluation the main task was to extract different types of useful information and to implement appropriate visualisations for the obtained data in an application called "VirtualSport"[4]. Furthermore the possibility of using additional data sources such as social media or other BBC sources should be evaluated to provide and display more information.

This was followed by the problem of designing an efficient architecture that was flexible and modular enough to exchange and upgrade individual components easily.

[3] currently developed by the BBC R&D department

[4] In the course of this dissertation, the "VirtualSport" software and/or parts of it will also be referred to as "the application", "the prototype", "the software", "the program" or similar.

Apart from these targets, the project also investigated reliable web technologies that are capable of serving large numbers of concurrent users (i.e. more than ten thousand simultaneous connections; *C10k problem*: see Section 2.7). The difficulty for an application like this lies in providing a high transmission performance for the timely data. The challenge was to minimize the latency between arrival of the live data at the server and display at the client. Thus the processing and transferral of the data needed to be executed in an efficient way, since a big offset (∼7-10 seconds) would have reduced the user experience dramatically and an even higher offset (∼15-20 seconds) would have rendered this tool unusable.

Another requirement was the usability of the application on different devices and platforms. The application had to run on computers, smartphones and tablet computers likewise, regardless of differences of the underlying environment.

1.4 Motivation

There were several reasons for this pilot project. The most obvious were the usage of available data and providing the customer with additional information. However there were also other reasons, such as the statistical analysis of sports and individual players to draw conclusions from their performances and finding information about areas for improvement[5]. Another reason was the increasing number of mobile devices and tablet computers ("second screens", see Section 2.4). These devices had recently been targeted by many applications and many customers use them for multitasking while watching television. This application is, following those principles, extending the possibilities of the original television experience, but also offering appealing data and visualisations as standalone software.

1.5 Structure

The report is structured in the following way: Chapter 2 will give detailed information about the background of this project, including related work and publications that were used as a foundation for this project. The following chapter (Chapter 3) analyses the available data in detail. This includes a detailed evaluation of the primary data source (Opta Sports data) as well as an investigation of other possible data sources. The next

[5] see Section 2.2 and [AS13] for more information about statistical analysis in sports

part (Chapter 4) reveals details about the software's system architecture and justifications for the design decisions made. Chapters 5 and 6 summarise the findings of the technology evaluation and reveal details of the implementation process of the client and server components, respectively. The Chapter *Testing & Evaluation of the Prototype System* (Chapter 7) provides detailed insight of the performance analysis and the multi-platform tests, alongside results of an user evaluation of the software.

The thesis is ended with the conclusions that have been drawn and an outlook into future work and further research questions that came up during this project.

Please note: A major part of this project involved the development of user interface and visualisation prototypes. The user interface uses colours, shades, opacity, etc. to represent different object states and properties (e.g. different colours for successful/unsuccessful passes, different teams, etc.). The reader is advised to keep this in mind when considering to print this dissertation, especially if printed in black and white.

A digital version of this dissertation can be found at
`http://stefan-klikovits.github.io/MasterOfScience-Dissertation/`
The on-screen version contains all figures in high resolution and can therefore be used for a better view of the printed figures.

Chapter 2

Background

VirtualSport is an application that will primarily follow the client-server model. The server is responsible for the data acquisition and processing. The client's task is the display of the data and the handling of user input. The use cases mentioned in Chapter 1 demanded that the application was usable as a standalone application as well as a software that was used while following a video broadcast. The latter usage form is commonly referred to as "second screen" usage. Regardless of the use case, the user interface provides both execution modes and the user can choose whether to display statistics, live views or both. The client employs web standards and technologies to provide flexibility in terms of execution environment.

Especially in the scenario, where VirtualSport is used as a replacement for the TV broadcast, the timeliness of the data is of high importance. Information that is older than a couple of seconds can be regarded as outdated and can only be used for the calculation of statistical data. To assure that the information is transferred in real-time, the server uses "push" technology, to eliminate the need for client's data requests. Pushing of data in real-time and Internet technology have been controversial philosophies, as the original implementation of web technologies demanded client requests and did not foresee the active sending of information from the server to the client.

The requirement of having thousands of concurrent users demands high performance server technology, and the concept of having lasting connections between server and client adds even more efficiency constraints. Traditionally, content providers could overcome the problem of peak requirements by simply adding fast cache servers, that would store copies of the generated content and serve these copies to the client. As VirtualSport does not generate the content pages on the server side, scaling of the server infrastructure represents a different kind of problem. Additionally to the higher

resource requirements due to standing client-server connections, the application evaluates possibilities to incorporate third-party data. Some of this content has to be transferred via the VirtualSport server, which also increases the server task load.

This chapter will place the project into its scientific background and give information about research areas that are related to VirtualSport. The first two sections will provide an overview of the BBC's work and its data sources (Section 2.1), followed by a short discussion of the value of statistics in sport (Section 2.2). This is followed by a background analysis of video data extraction research (Section 2.3), before Section 2.4 introduces the concept of *second screens*. The second half of the chapter provides information about current research on real-time applications (Section 2.5), data push technology (Section 2.6) and scalability issues for web applications (Section 2.7). The chapter concludes with a strategy evaluation for the inclusion of third-party data sources (Section 2.8) and an overview of the software engineering process used in this project (Section 2.9).

2.1 BBC, BBC sport and live data

The British Broadcasting Corporation (short: *BBC*) [Bri13b] is a public service broadcaster. Its purpose is to provide radio, television and internet programmes to the public "to enrich people's lives" [Bri13c]. The BBC currently maintains ten national radio and television stations respectively, dozens of local radio stations and offers several news and information services in 27 languages and many countries around the globe. It also maintains the corporation's online platform `http://www.bbc.co.uk`.

A large part of the BBC's focus is dedicated to sports. Apart from radio and television news broadcasting and coverage of live sport events, the BBC provides extensive information on various sports on its internet platform `http://www.bbc.co.uk/sport/` [Bri13a]. Next to "traditional" reporting, i.e. textual articles, enhanced with photos and short video clips, the BBC also uses technologies such as news tickers to accelerate the delivery of the content and information to the user.

The BBC has the rights to live cast certain sports events on its online, television and radio programmes[1]. Next to these TV rights, the corporation also receives access to data stream information describing the events on the pitch. This information is provided by external sources (other companies) and composed from the automatically produced extracts of the live video footage. However, this source of information has

[1] such as the *Football Association Challenge Cup* [Foo13] or *FIFA* [FIF13] world cup games

not been fully explored yet. The aim of this project was to create an application that gathers interesting data and displays this information to the user in real-time.

2.2 Sport and statistics

In recent years many of the big football clubs have moved to digital football analysis. What started as watching recordings of previously played games (both, their own and the opponent's) has moved to more detailed statistics. As in almost any activity, information is the key to success. It is essential to know on which side of the goal a player usually shoots the penalty kick or how often per game a defender fails to win aerial duels. This information can be used to compensate for deficits and exploit opponents' weaknesses.

Live football coverage on television employed systems that display statistics for some time and it is not unusual to see information about a player's covered distance, duel win-loss-ratio or the teams' ball possession percentages.

In the last couple of years individual football clubs started using these systems for analysis of games and training. Nowadays, one of the biggest providers for analytical systems is Opta [Opt13a]. Opta uses automated systems to extract information from video footage and provides this data to the clubs. The clubs employ analysts that use the information to draw conclusions for the team's training and game strategies.

Further information on football data statistics is described in [Hud13] and [AS13].

2.3 Data extraction from video footage

The extraction of data from live video streams is a topic that has been researched in many different areas. Video analysis is nowadays used alongside static image recognition for security systems (e.g. face and retina scans), in learning/teaching environments [GY06] and even for recognition of harmful insect plagues [BMTB10].

Recently automated sports events analysis systems have started to gain research interest. The findings and similarities between team sports like football and the area of automated video surveillance, e.g. for security reasons, draw attention towards each other. In the past years, a research group at the *Institute of Intelligent Systems for Automation National Research Council* in Bari, Italy has produced a large number of publications in the area of football[2] player identification and tracking. Examples of

[2] soccer

their work are referenced in [DLM+09] and [MSLO10].

Next to the publications of this research group there are also other concepts of automated sports data collection from video footage, such as the automatic identification of highlights [ABC+03] and the classification of game events [ZC04]. All these publications share football games and data as a research subject. The findings however are applicable to a large number of sports and other video analysis areas.

2.4 The popularity of second screens

With the spread of internet-connected, mobile devices such as smartphones, tablets and laptops, entertainment has shifted from a television-only environment to a multi-device setup [CD12], where it is always possible to lookup information online and exchange thoughts on social networks (e.g. Facebook [Fac13] or Twitter [Twi13a]).

Recent studies have shown that the topic specific activity on Twitter during live television events (e.g. talent casting shows) is growing and people use the platform as a means to expressing their opinion. [LC11, LC12]

Although it has not been directly researched, this project assumes that the behaviour is similar for any kind of enthusiastic viewer, regardless of the nature of the television programme. This matches the recent trend of TV shows to leverage Twitter for extended viewer outreach, described by Lochrie and Coulton in [LC11], leading to the belief that dedicated fans would share their views for live sports events in the same way.

2.5 The concept of real-time

From an abstract point of view the VirtualSport application is a piece of software that reads input data (e.g. data streams), processes this data and responds to certain information by carrying out actions. The whole data flow should happen with minimal or zero time delays. This concept matches the definition of real-time systems as “computer systems that monitor, respond to, or control an external environment” [Sha01a]. Real-time systems have been subject of many different research projects and there is a broad amount of publications giving overview of this topic. Early discussions of this topic can be found in [Sha01b] and [TC77].

Software that adapts to changes in real-time underlies different requirements than ‘standard software’. The most significant difference is that the dominating quality

factor lies in the timeliness of the data. Information needs to be parsed, processed and transferred quickly to assure that the data is displayed to the user while it is still valid. In many cases outdated information can lead to wrong decisions, causing threats to financial or personal well-being of companies and individuals. In most cases, the data that has been displayed to the user, can be discarded, since it is not of use anymore. These concepts make performance an important factor of the overall software and in many cases real-time systems use different design paradigms to ensure the fulfilment of the time requirements. Introductions into the architecture of real-time systems can be found in publications of Levi and Agrawala [LA90] and Laplante [Lap97]. Recent summaries of real-time systems design have been discussed in [Fow10] and [KAH08].

2.5.1 Real-time systems on the web

The concepts and publications discussed in the previous section look at real-time computer systems from a general point of view. The VirtualSport application will extablish the client-server connections using web technologies. These systems differ from the systems above in the fact that between the steps of obtaining, processing and displaying the data, the information has to be transmitted over a computer network[3] and the displaying technology is in most cases a web browser [Gar04].

This data transfer is a new challenge to real-time systems as network failures and limitations or fluctuations in bandwidth introduce new points of failure. Recent advances such as [Mat09] and [BL10] propose the use of web services[4] for real-time systems.

Further research has been undertaken on the subject of web services in combination with event-driven architectures. In these systems the general information passing happens by broadcasting and listening to events. This concept has been described in detail as the *Observer pattern* by the "Gang of Four" [GHJV95]. A more detailed definition of the architecture and an introduction of its advantages can be found in [Mic06].

[3] in some applications the data source and the processing component are at the same site, so that the 'expensive' transfer only happens between the processing and the displaying unit

[4] see [W3C04b] for a definition and [W3C04a] for an introduction of web services; extensive discussion of web services can be found in [ACKM04]

2.6 Pushing data

"Pushing" information to clients refers to a technology, that actively sends messages from a server to a client. These message transfers happens without specific requests and are initiated by the server. An early description of the push technology and its different work principles has been introduced by Franklin and Zdonik in [FZ98a] and [FZ98b]. Recently push technology became more popular with the spread of smartphone devices and social media (e.g. Twitter [Twi13a], Facebook [Fac13], etc.). The technology removes the user's effort to actively check (poll the server) for updates. Instead the device uses the technology to establish one or multiple consecutive connections to the server and informs the user when new information is available.

This technology has also been introduced into web applications. The usage of AJAX (*Asynchronous JavaScript and XML*) [Gar05] for the retrieval of information offers the possibility to establish a background connection to the server and poll for data[5]. Figure 2.1 displays the working principle of two different polling techniques. The first technology (*polling technique*) issues a data request to the server, asking for new data. This request is answered immediately. When the client receives the data, it waits for a certain time span and then emits another data request. The second technique (*long polling*), which has been commonly used in combination with AJAX, uses a slightly different methodology. The server does not respond to the request right away (unless there is new data available immediately when the request arrives). If there is no new information, the server keeps the request open until it reaches a certain maximum timeout or new data arrives and responds with the message stating the situation, respectively[6]. Immediately after the client receives a response, it sends a new data request to the server. This request is again delayed by the server until there is new data available. This principle assures that the data is sent to the client as soon as it is available. As you can see long polling, as the name suggests, technically is not a pure push-technology, but has been adapted for this purpose. [BvD08] and [BDK+02] provide a deep analysis of the usage of AJAX for push/pull messaging.

Similar to AJAX, there is another possibility to asynchronously fetch data from the server. `JSON-P` [Sim13] is a technology that exploits HTML's weakness for JavaScript [Int13] source file loading. The working principle for push updates is then similar to AJAX. The big advantage is that JSON-P allows loading data from other site domains

[5] see Section 2.6.1 for more information on asynchronous website technology

[6] Some implementations do not send empty responses messages, but wait for the client to receive a request-timeout. This removes server effort for initialising an empty message.

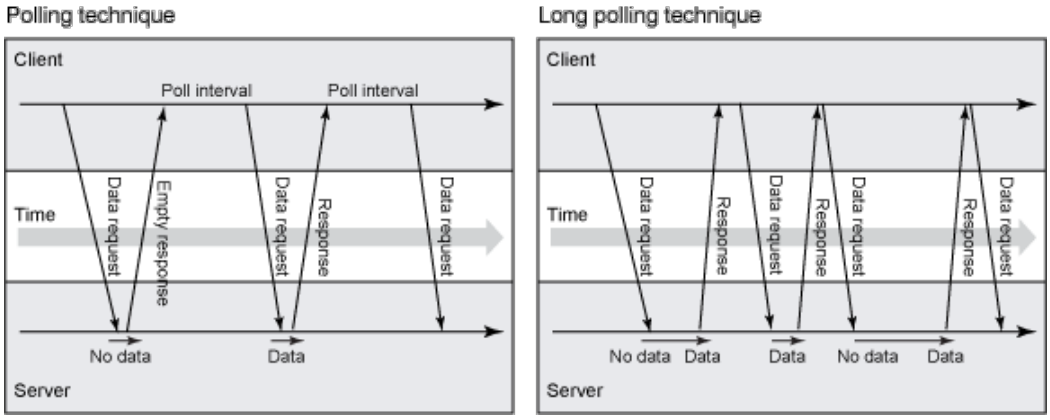


Figure 2.1: Conceptual time graph of *Polling* and *Long Polling*

Image source: Amir Jalilifarad - Real Time Commenting using SignalR and XSLT [Jal13]

and is therefore not restricted by the *Same-Origin Policy* [W3C13a].

A recently introduced technology is *HTML 5 web sockets* [Jav10]. This technology has been newly specified in version five of HTML[7]. It allows the client and server to keep up a lasting connection, which facilitates the transfer of data. An evaluation of this technology and a comparison to other push implementations, including HTTP polling, has been performed by Agarwal in [Aga10]. Deeper research has been performed by Lubbers in [Lub10] and Lubbers and Greco in [LG], who describe the usage and performance of HTML 5 web sockets in combination with proxy servers [HJ98]. Agarwal provides a detailed discussion about HTML5 technologies in [Aga12], where he compares the trade-offs of using *HTTP*[8] technologies over TCP[9] to the advantages, such as compatibility with security firewalls and HTTP Proxies [HJ98].

Next to these frequently used technologies, there are several other technologies that can be used to establish a background connection to the server. One is called *Forever Iframe*. This technique uses invisible Iframes [W3C13b] to transfer messages to and receive them from the server. Another possibility is to use Adobe Flash [Ado13] multimedia streaming technology to establish a durable connection between client and server. This connection is then used to transfer data. This however, requires the client to have the Adobe Flashplayer plugin installed in the browser. Lately the number of non-Flash-compatible devices, such as certain smartphones and tablet computers, is increasing due to manufacturer choices [Job10].

[7] *Hypertext Markup Language* [W3C12]
[8] *Hyptertext Transfer Protocol* [W3C99]
[9] *Transmission Control Protocol* [U. 81]

2.6.1 Asynchronous data loading

In website technology the terms *asynchronous content loading* and *AJAX* are usually used interchangeably. The term asynchronous indicates that a webpage fetches data from the server after the initial web page has already been rendered and displayed. Although there are other forms of asynchronous data retrieval (see previous section), AJAX is the most common form. In contrast to the "traditional" way, where a client requests a resource (i.e. a web page) from the server, waits for exactly one response and finally parses and displays this resource, asynchronous web pages can issue follow-up data requests to the server, fetching further data which can then be processed upon arrival. Hence "asynchronous" in this scenario refers to web pages that issue more than one data request, rather than referring to unanswered messages (as in i.e. UML 2.0).

Asynchronous web technology was initially used with `xmlHttpRequests` (short: *XHR*) in combination with JavaScript. JavaScript is essential for the usage of this technology, since native HTML only supports the display of information, not its modification.

The big advantage of using this technology is that parts of the web page/data can be updated without the need to refresh the entire web page. Another advantage is, that some parts of the web page can be sent directly in the response to the HTTP request, whereas less important parts (i.e. large multimedia files) are retrieved after the prioritised data has been loaded and displayed.

A disadvantage of this technology is that asynchronous data loading cannot implement a "standing" connection between the client and the server. As can easily be observed in Figure 2.1, there are regular intervals where the server does not have the possibility to transfer the data to the client. While these time spans tend to be in the area of milliseconds, the delays can increase during times of high server load.

An introduction into general asynchronous I/O, rather than asynchronous content loading on the web, can be found in [Jon13].

2.7 Scalability issues on the web

Scalability of web services has been the topic of many discussions and there are a countless number of proposals on how to improve it. An overview of the most common techniques can be found in [Hen06]. Apart from the various ways of improving the scalability by improving or increasing the quality and amount of hardware (called

"*scaling out*"), there is also the possibility to improve the way requests are handled (called "*scaling up*"). The challenge of handling a large number of simultaneous connections on standard hardware is commonly known as the *C10k problem*. This scenario has been defined by Kegel as the task to "handle ten thousand clients simultaneously" [Keg11].

Recently two server implementations have drawn attention towards them, both succeeding at this challenge. There is `node.js` [Joy13] on the one hand and `nginx` (pronounced "*engine x*") [Ngi13a] on the other. Node.js is a web server implementation, that uses JavaScript as scripting language and runs on Google Chrome's V8 JavaScript Engine [Goo13]. It employs the observer pattern [GHJV95] to react in an event-driven fashion to requests. Nginx is a lightweight server implementation that is highly resource efficient and has minimal memory requirements. Currently 23.8 % of the top 100.000 web pages and 32.1 % of the top 1.000 pages on the web use nginx as a web server [W3T13], including 'big players' such as Facebook, Netflix and Soundcloud [Ngi13b].

2.7.1 Scaling out efficiently

Performance improvements on a *scale up* basis eventually reach limits. At the point where a program's efficiency cannot be increased anymore, a *scale out* approach is necessary. Scaling out means to add performance by adding more and/or better resources. There are several approaches to do so. The first one is to run the program on faster and better hardware. This however is limited by the current state-of-the-art technologies and the resources (i.e. money, space, etc.) available. A different concept is to run multiple instances of the program at the same time and split the requests between those instances. The most trivial way is to employ multi-processor/multi-core computer systems that execute *fork*[10] commands (or similar) to multiply the processes ran and split the taskload between the individual threads and processors/cores.

A more complex proposal is to run the program on several independent computers. The difficulty here lies in the need of a dedicated front end that serves as a proxy[11], mapping each incoming request to one of the multiple computer systems and thereby reducing the individual task loads. This approach is widely used for static and slowly changing content in the form of server-side cache machines. However, proxies can also be used to distribute requests between servers directly and thereby offering scale

[10] "fork() creates a new process by duplicating the calling process" [die13]

[11] also "*web accelerator*" or "*reverse proxy*", see [CQRD03] for an explanation of web proxy systems

out for fast changing and generated content.

The multi-machine and multi-processor/multi-core system concepts can be combined, which leads to a highly efficient, but also highly complex system setup.

In any implementation that runs a program redundantly, it is necessary to pay close attention to the way the information is spread between the individual instances. Both approaches require an efficient information update policy across the entire system to avoid the propagation of wrong (outdated) information.

2.8 Data sources

The fully implemented VirtualSport system uses different data sources to implement all functionalities. The main data source is acquired in the form of XML data streams[12] [W3C08]. When it comes to querying data streams there are several problems to be considered. One of them is that the query results can change whenever new data is arrives. A second problem is, that the amount of data to be parsed is usually uncertain and it cannot be determined upfront at which rate the data will be flooding in. In a worst case scenario, there is too much data to store. This means that some of the data has to be deleted and the results are non-deterministic. Advanced problems include the modification and deletion of events at a later point. The problem here is that the original data could have been compressed[13] in order to store more data. It is then not possible to update the query result reliably. Another question is whether to use all of the obtained data for the query or to use sliding "windows" on the data, that only look at the most recent data. These windows can be defined on e.g. a time or tuple number basis. A final research problem is the concept of dropping data in the case of too high input data rates. The question here is which data to discard, without losing any important information. Detailed research about the problems of continuous queries and data stream management systems have been researched by Babu and Widom [BW01] and Abadi et. al. [ACc+03].

The subject of processing data streams has been thoroughly researched by the database community. Further introductions into this topic can be found in [BBD+02] and an extensive discussion about the research is given in [Mut05]. Publications that are directly related to analysing XML based data sources can be found in [OFB04] and [CDZ06].

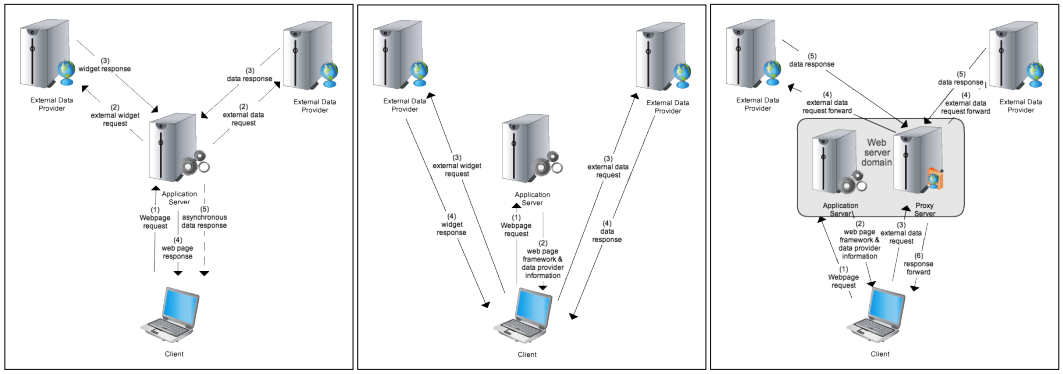
[12] "data stream": A sequence of digitally encoded signals used to represent information in transmission" [Adm96]

[13] compression happens by computing aggregate values such as mean, sum, count, etc. for a dataset

2.8.1 Mashups

Recently there has been extensive research to combine multiple data sources and display the results to the user as one system. This combination of systems behind a homogeneous user interface is commonly referred to as "*mashup*".[14]

Depending on the architectural unit that requests the external resources, there are three different forms of mashups: server side, proxy-style and client-side mashups. Figure 2.2 shows simplified concepts of the three working principles[15]. The principle of server-side mashups (Figure 2.2a) is the following. Following a client request, the server fetches the required data from the individual data sources and responds to the client with a single processed web page, that has already been parsed. This technology requires increased processing and more resources on the server side.



(a) Server-side mashup (b) Client-side mashup (c) Proxy-style mashup

Figure 2.2: Different forms of mashups

In contrast to this method, proxy-style mashups (Figure 2.2c) and client-side mashups (Figure 2.2b) retrieve only the basic layout and information about the different data sources from a server. Once this information is loaded, the client sends requests to collect the required data from the different data sources and the initially loaded system (i.e. the HTML web page) takes care of its appropriate display. The speciality of the proxy-style technique is that the requests have to be sent to the domain of the original request. This is due to the fact that most browsers only allow AJAX requests to the domain of the originally loaded site, to prevent loading of malicious code. The solution to this problem is to implement a proxy server at the server domain (hence *proxy-style*) to give the client the possibility to fetch the data.

[14] A short introduction into the working principles of mashups can be found at [OBB].

[15] The digital version of the dissertation provides Figures in high resolution, allowing you to zoom. See `http://stefan-klikovits.github.io/MasterOfScience-Dissertation/`

The third category are client-side mashups. For the reason mentioned before this group of systems is not very common and therefore has not been subject of deep research work. For detailed information about client-side mashups, the problems of this strategy and on how to overcome them, the reader is referred to [SMNT10].

Further information about the development, the classification and the technical and social challenges of mashups can be found in [Mer06] and [CH07].

To maximise performance and simplify access, the VirtualSport application will use a combination of both techniques. The final software will merge information of data sources with quick response times on the server side (sports event data, BBC live ticker, BBC internal databases) and transfer the data to the client. This mashup will then be extended on the client with proxy-style mashup content from slower or external sources such as social media platforms or BBC web radio audio data.

There are some further challenges that need to be overcome in the implementation of the VirtualSport application in an production environment. Although applications based on the combination of different data sources resemble the well known *broker business pattern* [RS04], the situation becomes more complicated when facing requirements such as real-time processing and display on mobile devices. The first problem was the research topic of Liu and Deters [LD08], who propose - amongst other strategies - asynchronous processing as a solution. The latter problem was discussed by Salo et.al. in [SAM11]. Their suggestion is to use MapReduce [DG08] in combination with RESTful web services[16] [Fie00] to overcome problems.

2.9 Software engineering process

The software development process used in this project is based on Kanban [And03]. Kanban is an agile software engineering framework, offering even more flexibility than Scrum [DBLV12]. Scrum's goal is to finish as many work packages as possible from a backlog that has been defined at the beginning of the sprint. The sprint length is usually constant over several iterations. Kanban however aims to minimise the work-in-progress and increase the velocity of work packages. This means that once a work package has been started it should be finished as soon as possible. The sprints can have variable lengths and instead of a rigid sprint backlog, there is a loose collection of tasks in the "To Do"-column of the Kanban board, sorted by priority.

[16] *REpresentation State Transfer*; *RESTful* refers to REST architectures that follow the six basic constraints: *Client-Server*, *Stateless*, *Cacheable*, *Layered system*, *Code on demand* and *Uniform interface* [RR07]

One of the most important features of Kanban is the quick and informal feedback process. This allows the users to retrospect sprints and draw conclusions for the next iteration. The review cycles in this project were set to the length of one month. At the end of every review cycle it was necessary to look back and adjust the process if necessary. The standard sprint length was set for a duration of two weeks. However, if for example a certain event (e.g. an exam) required a project pause for some days (e.g. one week) it was intended to have only one sprint over three weeks in the review cycle rather than a very short sprint with very little progress. It was essential, that after each sprint a working application was available and that the existing features were still functioning.

Kanban software development usually uses visual progress tracking. This means that the Kanban board (see Figure 2.3) including the backlog are displayed in a place where every project stakeholder has access to. The difference between a standard Scrum board and the Kanban board is that work-in-progress (abbreviated as *WIP*) can be limited. This WIP limitation makes sure that there is a maximum number of Kanban cards in each column. If the maximum is reached, no other work package can be added to this state and the team (the developer) is forced to work on one of the tasks that are currently in a *blocked* state.

Experiments and case studies have measured increased productivity after switching to Kanban process management (see [SJS12] and [ACL+12] for details). A thorough description of the Kanban process used in this project is explained in Appendix D.

2.10 Background summary

This chapter placed the concept of the VirtualSport application in a wider research context. It was shown that the individual requirements to the application can be connected to known problems. A major goal of this project is to fulfil the real-time requirements of the intended application. The chapter provides references to related work that faced similar challenges and introduced further background information on the use of real-time systems in an online/web based environment and the latest advances for push update technologies. The last part of this chapter summarized the current state of research regarding scaling of web applications, data stream processing and the various forms of mashups, which resemble a large part of the VirtualSport application. The chapter is concluded with an introduction of Kanban, the software engineering process, that has been used in this project.

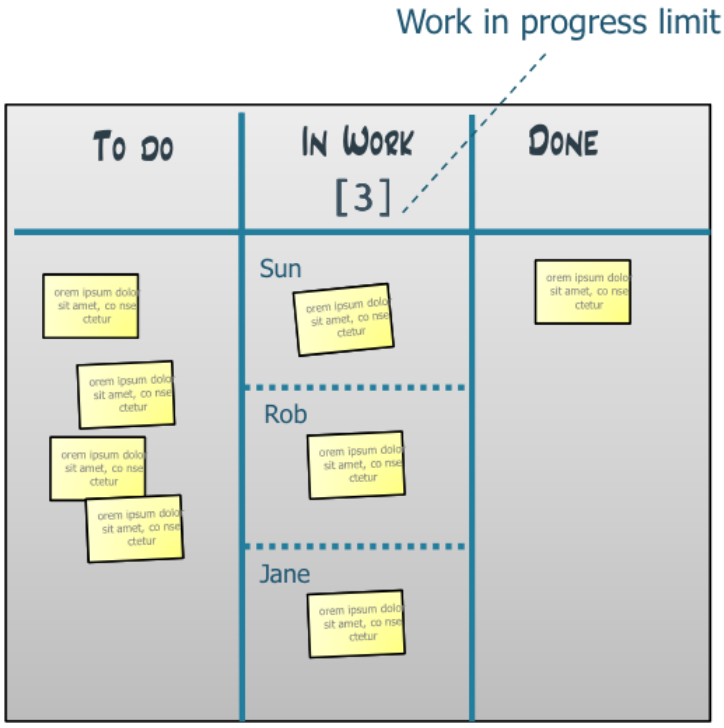


Figure 2.3: Example Kanban board

Image source: Sathees Kumar.K - The Art of Project Management
http://satheespractice.blogspot.co.uk/2012/05/kanban-board-for-agile-projects.html

VirtualSport combines the concepts that have been described in this chapter and establishes a lasting client-server connection to push real-time information to the user interface. Although extensive background research has been undertaken, it was not possible to find scientific evaluations of the employed technologies. To the best of our knowledge it appears that this was the first scientific attempt to combine and analyse these technologies. There are for example many web pages providing information on the usage of event-driven server architecture to effectively serve extremely high numbers of client requests and web socket connections, but there are no publications that objectively analyse the scalability of this technology and provide information on response times at peak capacity. One of this project's goals was to fill these gaps and provide objective information that can be used by follow up projects and other research work likewise.

Chapter 3

Analysis of available Data Sources

High data quality is the key to success for many applications. An upfront analysis of the data provided by the BBC was essential for the use cases the prototype would fulfil, as well as the further design and implementation phases. The provided files had to be evaluated and the structure and format of the information be determined. Further interest went to the amount of available data.

After the data format had been structurally analysed, the contextual information needed to be evaluated. In this project, the data represented football games. This means that the information had to be broken down to see the distribution of the events for the individual players and teams. It was also necessary to identify how often each event type is occurring.

The delays between the players' events had to be evaluated as well. Some visualizations, e.g. a full player tracking could only be implemented if there was enough position data available. Other visualisations require the data to occur in a certain frequency (e.g. touch maps or player heat maps).

This chapter is structured in the following fashion: First Section 3.1 will explain the format and structure of the data, followed by an analysis of the contextual information of the data source files. The second part of this chapter (Section 3.2) is an evaluation of other data sources and their forms of implementation. These information providers could be integrated in a possible follow-up project.

3.1 Analysis of BBC data files

The following section describes the data that has been provided by the BBC. The datasets describe an entire football game each and contain all the events recorded during the game. The BBC provided two data files, describing the games of *Queens Park Rangers vs. Bolton Wanderers* on 13^{th} August, 2011, and *Tottenham Hotspur vs. West Ham United* on 25^{th} November, 2012. This data is normally provided as a live data stream but has been stored in files for analytical usage. A typical game file contains up to 2000 entries [Opt13b] recording the events on the pitch.

3.1.1 Data format and quality

The Opta data files use the XML format. The root tag is the `games` element, which is parent to exactly one `game` tag. This `game` tag stores basic information about the football game in its attributes. Examples for these details are team names, name of the competition, match day and season.

Appendix A displays a short extract from one of the Opta data files. The `game` node is the parent of numerous `event` elements. Each `event` has a timestamp attribute and x/y coordinates that indicate the position on the pitch. Further attributes specify its type (pass, goal, shot, etc.), the game time, the involved player (with player ID) and/or the outcome (success/fail) of the event.

Events also have qualifier child nodes (`Q`) storing the information in form of qualifier type and value attribute pairs. These qualifiers provide additional details about the event like the type of a pass (e.g. cross, flick-on), team lineup information (players and shirt numbers or foul details (e.g. hand ball, dangerous play, etc.). The number and type of the qualifiers varies between event types and also the events itself. This means that not all the events have all the possible `Q` tags.

3.1.1.1 Correctness of the data

From the given data set, it is not possible to verify the correctness of the data. Due to the nature of the data, there is no possibility to check whether given data really represents the events on the pitch. It is also impossible to determine how precise the information is. The prototype application will be developed under the assumption that the given information is correct and accurate.

3.1.2 Analysis of the contextual information

Apart from the structural analysis, the data has been evaluated for event frequencies and other data distributions to evaluate the context of the data.

3.1.2.1 Event frequencies

The two datasets `game1` (Queens Park Rangers vs. Bolton Wanderers; 13.08.2011) and `game2` (Tottenham Hotspur vs. West Ham United; 25.11.2012) have been analysed for their statistical data before starting with detailed planning and implementation.

The tables B.1 and B.2 in Appendix B show the event frequencies by event type in `game1` and `game2`, respectively.

`Game1` contains a total of 1673 events, of which 51% are passes, 8.7% are out events, 6.5% are ball recoveries, 5% are clearances and 3.8% aerial duels. All other event types amount to less than 3% each. This means that the most common event type accounts for more than half of the events and the five most common event types make up for nearly three quarters (74.96%) of the events.

`Game2` has 1608 events, 54.7% passes, 8.6% out events, 5.35% aerial duels, 5.3% ball recoveries and 4.41% clearances. Other events amount to less than 3% each. In this dataset the passes even have a slightly higher ratio and the five most common event types amount to more than three quarters (78.36%) of the events.

3.1.2.2 Events per player

The events distributions for the individual players are displayed in Appendix B tables B.3 and B.4. The tables show that in mean average, each player has 56 and 58.5 events in `game1` and `game2`, respectively. This means that in both games the individual players have information updates in average every $\frac{5400}{56} \approx \frac{5400}{58.5} \approx 92.3$ seconds. The maximum number of events associated with a certain `player_id` is 104. This leads to a minimum average of one update per $\frac{5400}{104} \approx 51.3$ seconds.

3.1.2.3 Time delays between player updates

A standard football game lasts 90 minutes (= 5400 seconds)[1]. In both datasets this results in an average of events happening every $\frac{5400}{1673} \approx \frac{5400}{1608} \approx 3.3$ seconds, which means slightly over 0.3 events per second.

[1] continuous time, without extra time

Since the BBC was only able to provide two datasets, it is not possible to say whether the similarities in the data distributions are coincidental or observable in other records as well.

Deeper analysis showed that the average data rate of 0.3 events/sec. does not allow a full player tracking. The most significant problem is due to the fact that the majority of the events require some sort of action on the ball. Players that are not involved in some ball event and also do not provoke any other event like substitutions, cards, fouls, etc. will not show up in the data stream. The general trend suggests that the position of midfield players is updated more frequently than the position of defenders or the goal keeper. A simple, self-written analysis tool revealed that some players are not involved in any event for more than 10 minutes. The data set `game2` even contains a player that as a maximum timespan of over 21 minutes between two events with his `player_id`. This makes an effective player tracking impossible.

The problem can only be overcome by obtaining more player position data, especially for players that are not involved in any current ball actions. The research and development department at BBC is in the process of creating a set applications, which could provide data like this. It is not possible to say, if or when and to what extent these programs will be available for usage and integration into the VirtualSport project.

3.2 Feasibility of other data source integrations

In addition to the Opta data stream and the intended connection to the BBC internal tracking tools, there are a number of other information providers that could be integrated into this application. Possibilities vary from the integration of BBC live commentary (web radio, live tickers, etc.), to display of related Twitter feeds and social media content, such as status updates and messages on the teams' and players' Facebook fan page walls. Another possibility would be to include user generated multimedia information into the application. This can be done by displaying photos and videos that have been recorded by users in the stadium and uploaded to certain multimedia platforms such as Youtube, Instagram, etc.

3.2.1 Forms of integration

Most of the big data providers offer application programming interfaces (short *API*s) to provide developers with the possibility to include their data in other applications. The

access of the data is usually provided via REST APIs [RR07] and integration occurs on the server side where the server application fetches the desired data before the client request is answered[2]. Although this setup involves added effort on the server-side it has become de-facto standard and is widely used. Modern web technologies, such as *Node.js*[3], provide a growing number of plugins to access REST services in general (see [Nod13b] for an example), but also offer application specific extensions for large content providers such as Facebook [Eng13] and Twitter [McM13].

It is important to keep in mind that REST APIs evolve and change and this can lead to non-functioning data integration. Twitter for example reworked their entire API during the change between their API versions 1 and 1.1 and included a new authentication process (OAuth [OAu13]). This not only required developers to adapt their program but also remodel certain parts of their applications to adapt the new authentication requirements. See [Twi13b] for information about the Twitter API v1 retirement.

As REST and SOAP [Dai11] represent the dominating standards for cross domain data access, only a few alternatives are being developed. However, Twitter recently introduced so-called *widgets*. Widgets allow account holders to integrate Tweets and Twitter timelines based on usernames, trending *hashtags*[4] or combined lists of usernames and hashtags. The advantage of these widgets is the easy integration into a web page. The only requirement to the developer is to copy & paste two html-tags into the application. At run-time the integrated JavaScript code executes a server call, fetches search results and takes care of the common Twitter style and display guidelines. On the downside, these widgets are not dynamically configurable. To adjust the widget settings, the developer has to manually modify the settings on the Twitter account page.

Facebook is choosing a different path and offers client-side APIs so that the traffic does not have to be routed over the developer's server. The API then gives limited access to the currently logged in user's data as well as virtually any public data of any public profile (personal as well as business profiles). This approach allows developers to quickly request data sources without the need to manually modify settings (see paragraph about Twitter above). Facebook further offers powerful options in terms of user interaction, as not only the teams' Facebook information is available, but also the current users' profile data. This allows to adjust the displayed information. An example

[2]This strategy represents the server-side mashup introduced in Section 2.8.1

[3]see Section 6.2.4 for more information about Node.js

[4]hashtags are keywords that tag posts and allow other users to search for tweets containing these hashtags

would be to search through the user's news feed and display his friends' posts that are related to the current game. Combined with functionality that offers the user to post status updates from within the application, this would create a highly interactive user experience and increase the user activity.

3.3 Conclusion

This chapter explained the necessity of high quality data and the importance of having a reasonably large amount of data. As the VirtualSport application's purpose lies in the appealing display of information, it would render the software useless, if the data amount or quality was too low.

In the course of this chapter it was shown that the data provided by the BBC is rich enough to support the display of certain visualisations and the extraction of statistics. However, since the data does not include continuous position information of all players, certain visualisations (such as a full player tracking) cannot be offered. In the course of this chapter, it is also explained how a live-application could leverage other data sources for its own benefit. Especially the integration of social media could provide users with interactivity and tie consumers to the application for a longer time period.

Chapter 4

Design and Concept of the System Architecture

Although VirtualSport might resemble a classical client-server architecture, the system concept was not thoroughly planned before the start of the implementation. The resulting software is the product of the dynamic software engineering and evolved from a process that included frequent refactoring.

This chapter will provide details about the system architecture and the software engineering process of the application. First Section 4.1 will give some introductory explanations about the iterative design concept and its relation to the Kanban methodology. The next Section (Section 4.2) will introduce the overall architecture of the final version of the software, before Section 4.3 takes a closer look at the individual components and provide a brief outline of those architectural units.

4.1 System design process with Kanban

The system design is closely coupled to the Kanban process used in this project[1]. The architectural concept of the system components was not specified upfront, but evolved iteratively.

Following the YAGNI-principle (for "You Ain't Gonna Need It"), only the parts that were necessary for the implementation of the features were designed and implemented. Although, this lead to some refactoring overhead throughout the project, it

[1] See Section 2.9 in Chapter 2 and Appendix D for more details about Kanban and the application in this specific project.

reduced the number of unneeded features to a minimum. Furthermore, the system design was reviewed constantly and refactoring helped to keep the code base clear and simple.

The resulting system concept is described in the following sections.

4.2 System architecture

Figure 4.1 shows a simplified UML component diagram. This diagram describes the working principle of the software in a generic way and shows the simplified information flow through the system.

As it is easily observable, the system will be divided into three components. The first one, the *Data parser* is the interface between the system and the data source(s) and converts the different data types into internal format. The second server component (*Server*) is responsible for the communication with the client and for pushing information to the user interface. The third component is the client itself. It serves as display and user interface likewise.

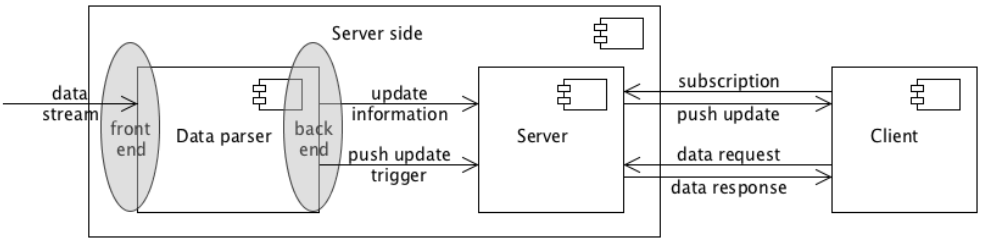


Figure 4.1: Architectural concept of the VirtualSport application

4.3 Architectural Components

The prototype consists of multiple components interacting with each other, where each component serves as a template implementation for a possible future application that is used in a live environment.

4.3.1 Data parser

The first component is a data stream parser, that accesses and parses incoming data streams, handles the data and triggers the data processing. Since the main data source is an XML data stream, the parser is geared towards analysing such data efficiently. Apart from this data parser, the application provides a generic interface for the integration of other data sources. The parser itself consists of two subcomponents. One is the frontend, which is necessary to gather the data either via pull requests or by accepting push updates (incoming port) from the data sources. The other one is the backend, responsible for evaluating the incoming information and generating/updating the current game representation, updating statistics and informing the server component of the arrival of new data.

4.3.2 Server

The transfer of the data to the user is handled by a server component. The server is responsible for answering the clients' requests, accepting clients for information update subscriptions and pushing information updates to subscribed clients.

4.3.3 Client

The client is the user interface of the application. Its purpose is to subscribe itself to updates and present data on arrival. Since not all of the data is transferred via push updates, the client is also responsible for translating user input into server calls and thereby requesting further information. The client is platform independent and exclusively uses web technologies such as HTML(5), CSS and JavaScript to be usable on as many devices as possible.

4.4 Summary

Through the application of Kanban and refactoring, the system evolved into a classical client-server architecture with three basic components. The two components on the server side are responsible for the acquisition and processing of the data, where the client component serves as a user interface and display unit. The system concept implements flexible component based principles, where the individual architectural

units can be independently exchanged and updated. The following two chapters provide detailed explanations about the design and development of the client and server components, respectively.

Chapter 5

Design and Implementation of the Client

The user interface is one of the crucial components of the application. Chapter 3 explained the importance of the data quality. However, the user interface is the part that will present this information to the user. The involved technologies have to be reliable and well-established to serve a wide user base. It is essential to keep in mind that the most interesting data and the best statistics will not be used if they are not presented in an appealing form.

This chapter explains the design and development process of the client component in detail. First Section 5.1 explains the reasoning behind the selection of the client technologies. This is followed by Section 5.2 and Section 5.3, that provide information about the development of the prototype and the final user interface implementation, respectively. The chapter is concluded with an overview of the data visualisations that have been implemented as examples for the prototype.

5.1 Technology evaluation

Although there was no restriction by the BBC, it was the wish of all stakeholders to use web technologies. Since these technologies are designed to serve on the Internet, they have been developed to run in many different environments in a similar way.

The following section describes the available user interface (short: *UI*) technologies and explains their advantages and disadvantages, respectively.

5.1.1 WebGL

The *Web Graphics Library* [Khr13b] is a JavaScript API that offers functionality similar to *OpenGL* [Khr13a]. The aim of this library is to provide a similar API to OpenGL ES 2.0 in the HTML context [Khr]. This means that it is possible to create 2D and 3D models of objects and modify them natively in the browser. As WebGL is currently released in version 1.0, it is highly likely that the functionality and API will change. This could lead to big effort in terms of adaptation to new versions. Another disadvantage of such a 'new' technology (WebGL 1.0 was released in March 2011) is that some web browsers, especially Microsoft's Internet Explorer and several mobile web browsers, do not support this technology yet. Further challenges are that the browsers manufacturers often add extensions to their WebGL implementations, that provide additional functionality. These extensions are not standardised which may result in the need for redundant implementations for different web browsers.

As a conclusion, the use of this technology would probably result in the need to redundantly implement some parts of the user interface for certain browsers and also have no support at all for other web browsers. The large benefit would be the 3D support. As this project is a feasibility study and the display of 3D objects is not a requirement, the use of WebGL would come with a lot of effort compared to relatively little benefit.

5.1.2 Adobe Flash

Adobe Flash (often abbreviated as *Flash*) is a platform that allows the display multimedia content like graphics, sound and video. It has been a well-known technology for the creation of browser games and web pages. Since 2008 Adobe also provides the Adobe Integrated Runtime (short: *AIR*) that supports, in combination with Apache Flex, the creation of Rich Internet Platforms. However, Flash has lately been refused by mobile operating system manufacturers. The main reason for this, amongst others, is that Flash drains a lot of energy during its execution and thereby decreases the devices' battery run times [Job10].

Although Adobe provides functionality to compile Flash into HTML 5 and AIR code, that can be also used on mobile devices, the usage of this technology would result in overhead for adapting the Flash code for the technologies to compile for.

5.1.3 HTML 5, CSS, JavaScript

HTML 5 is the latest version of the *Hypertext Markup Language*. In combination with CSS (*Cascading Stylesheets*) and JavaScript, HTML has been the main form of content display on the web. One of the main features of HTML 5 compared to previous versions is the `<canvas>` tag. This tag supports the drawing and modification of shapes and texts via a JavaScript API.[1]

The big advantage of HTML 5 is that the standards are well defined and there is a large number of frameworks and libraries that facilitate its usage. Since HTML has been used and improved for over 20 years, there is a large community that supports this technology and HTML 5 has been accepted widely. HTML and HTML 5 are supported on all of the major operating systems, browsers and devices, which makes it an ideal technology for the user interface.

A disadvantage of HTML 5 is that the native JavaScript API can require a lot of programming effort (e.g. modifying the content of a canvas-tag). Therefore it is common to use one of the many libraries that simplify this task. `JQuery` for example evolved to a de-facto standard. However, the use of a third-party library involves certain risks such as discontinuance of the development, lack of support or wrong implementations that cannot be solved. There is a large number of similar libraries that solve these problems though and it is in general easy to switch from one library to another.

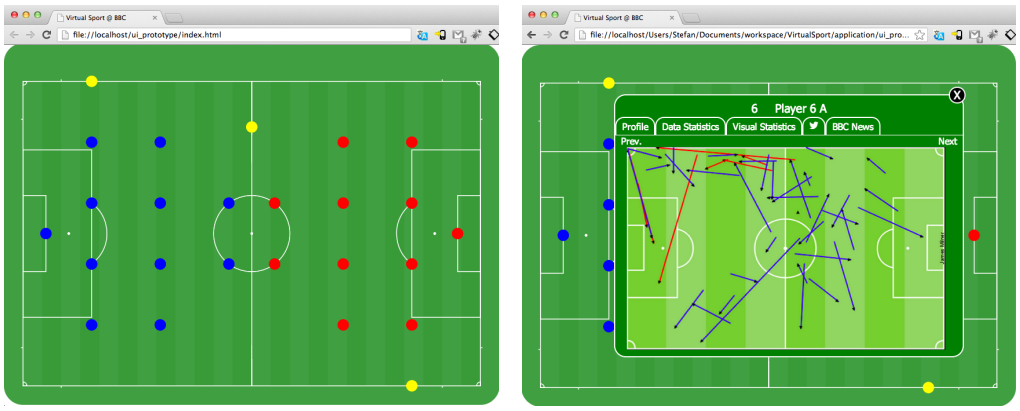
5.2 User interface prototype

Following the decision to use standard HTML 5 web technologies in combination with JavaScript libraries, the development of the user interface prototype started. Figure 5.1 shows screen shots of the prototype that has been agreed on as basic design. The team line up view is shown in 5.1a. Additional visualisations and more data is displayed in popup information boxes that open after clicking on the players. (See Figure 5.1b)

5.3 Implementation

After approval of the prototype, the user interface functionality has been improved and extended. Figure 5.2 shows some of the new features. The settings can be adjusted via JavaScript parameters and are stored in cookies to remember a user's settings. In

[1] the canvas tag is also used for WebGL, described in section 5.1.1



(a) The UI prototype, displaying the two team formations on a generated pitch

(b) A visualisation of the popup container that displays further information

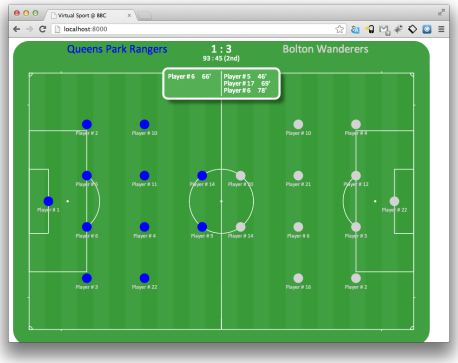
Figure 5.1: Screen shots of the user interface prototype (used static data and images as mock-ups)

Figure 5.2a the new team formation pitch is displayed. Attention should be given to the team names, the player names[2], the game time and score, and the `onMouseOver` score board display. Figure 5.2b shows the info area, which is same sized as the other pitch. This feature allows to see the visualisations on a bigger panel. Another feature is displayed in Figure 5.2c. It shows the "vertical pitch" display functionality, that was requested by the BBC after the evaluation of the user interface prototype. Figure 5.2d introduces possibilities of the combination of the two functionalities mentioned above. It is possible to have the pitch with the formation info on the left and the information pitch on the right side next to each other. This setup allows a quick navigation between the player information and therefore easier comparison of the players.

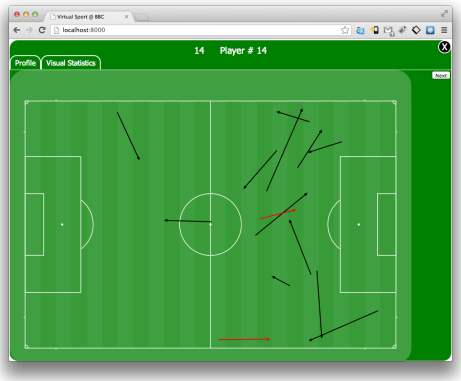
5.3.1 Libraries

The libraries that have been used for the user interface are `jQuery`, `Kinetic.js` and `smoke.js`. These third-party libraries have been extended with own libraries to facilitate code re-use and functionality that is not provided by the manufacturer.

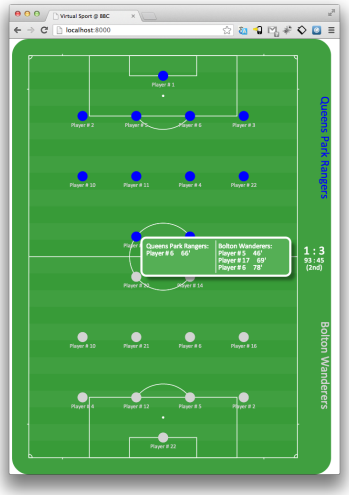
[2]generated from the player's shirt number, as the necessary data files were not available



(a) The enhanced pitch design



(b) Horizontal full screen info area



(c) Vertical pitch design

(d) Vertical pitch (l.) with info area (r.) side-by-side

Figure 5.2: Screen shots of the enhanced user interface

5.4 Visualisation features

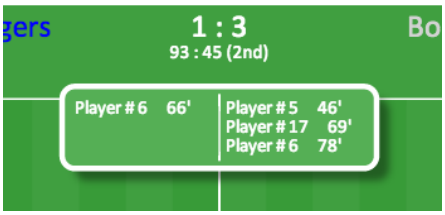
The application implements examples of data visualisations. To assure that the implementations meet the stakeholders' requirements, regular meetings were held to discuss implementations and further display opportunities.

The following section describes the visualisations that have been implemented as sample visualisations for the prototype application.

5.4.1 Team lineup pitch

The `team lineup pitch` displayed in Figure 5.3a shows each player that is currently on the pitch. It also displays further information such as substitutions or delays. It is possible to click on the individual player icons and thereby open a panel that shows further information.

(a) Team lineup pitch substitution information



(b) Current game state information display

Figure 5.3: The "team lineup pitch"

5.4.1.1 Score and game time information

The team line up pitch also displays information about the current state of the game. Figure 5.3b shows a screen shot of the information that is shown apart from the team names and player positions. This information displays the current game time and the score. The table of goal scorers is shown as an `onMouseOver` effect when the user places the cursor over the score-text.

5.4.2 Live pitch

A second representation shows the `live pitch`. This visualisation of a football pitch displays the events currently happening on the pitch. Each event (e.g. a pass, shot or foul) is shown for a few seconds and disappears after a specified time out. Although not all game details are displayed[3] it is possible to follow the basic gameplay. Similar to the `team lineup pitch`, this pitch informs about substitutions and delays as well. Figure 5.4 shows an example live pitch display during a football game.

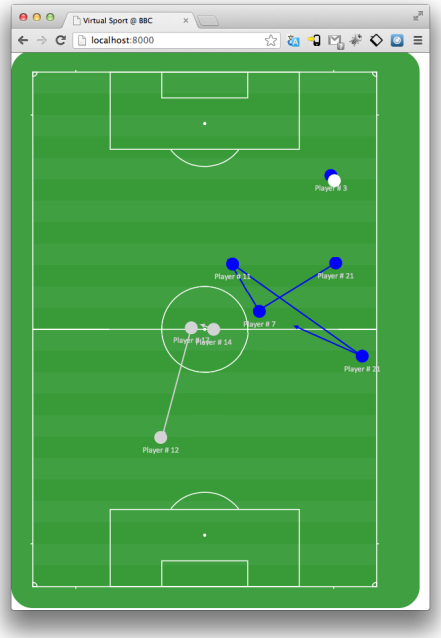
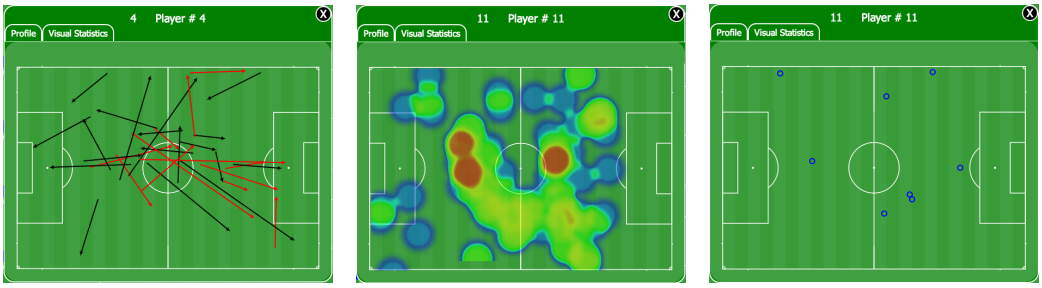


Figure 5.4: A screenshot of the live pitch during the game

[3] such as positions of players that are not involved in any events

5.4.3 Popup / info area statistics

This part of the user interface displays details about individual players, such as heat maps and pass maps. The number of visualisations is limited, as they only represent examples. A live-application should aim to increase the number of displays by adding information from other data sources as well.



(a) An example for a player pass map

(b) An example for a player heat map

(c) An example for a player touch map

Figure 5.5: Data visualisations for individual players in the info area

5.4.3.1 Pass map

A touch map, such as the one displayed in Figure 5.5a shows each pass on a virtual pitch. The passes are represented as arrows. A successful pass is drawn in black, an unsuccessful pass (offside pass, pass to opponent) in red. This information can help determining whether a player is playing offensively or defensively by looking at the start and end coordinates and the direction of his passes.

5.4.3.2 Heat map

To determine where on the pitch a player has the most activity, a heat map visualisation has been implemented. A heat map is a layer placed on top of the pitch that displays areas with high activity as in red and areas with low activity in blue. The values in between are coloured - in ascending activity frequency - in a gradient colours from blue over green, yellow and orange to red. Areas without activity are not coloured. An example for a player heat map is displayed in Figure 5.5b.

5.4.3.3 Touch map

A touch map (see Figure 5.5c) information is similar to a heat map. The differences to a heat map are, that a touch map only displays ball contacts and the icon is a small

circle symbol. Compared to the heat map, this visualisation only displays a player's active involvement in the game.

5.5 Conclusion

In the course of this chapter the entire implementation of the client component has been explained. The beginning of the chapter describes the technology evaluation and provides justification for choosing HTML 5, CSS and JavaScript as client technologies. The second part describes the creation of a user interface prototype. The chapter is concluded with a detailed focus on the final version of the prototype client. The individual parts of the user interface are presented and example implementations for data visualisations are illustrated and explained.

Chapter 6

Design and Implementation of the Server

As with the client, the VirtualSport server was implemented with technologies that allow flexibility regarding the used production environment. Therefore this component employs technologies that are used by a broad user base and are likely to be supported in the future. This however does not mean that experimental tools, libraries and patterns could not have been used as well, as long as the main application stayed functional in case the support for these features ended.

The technology for the communication had to be suited to connect the server with a large number of clients. The framework does not only permit the polling of data, but also pushing data to the client. This assures that the client receives the information as soon as it is available, and also saves resources on the server side, which would otherwise have had to be spent for sending empty "no new information" messages to the clients.

A backend requirement for the server was the communication with the data sources. Since it is not known yet, which data sources (other than the Opta data stream) will be used in the future software, the server has to be flexible enough to implement additional data providers.

It was important that the communication as well as the overall server technology were efficient in terms of resources, as it could be expected that an implementation of the application in a production environment would easily attract tens of thousands of users simultaneously, especially for sports events such as cup finals, games between teams of the same city (derbies) or close league standings at the end of the season.

The performance of the overall system had to be scalable and the server implementation needed to respond to requests fast and push new data to the clients in (near) real time.

6.1 System architecture

The server component consists of a number of subcomponents that are interacting with each other. These subcomponents offer well-defined interfaces to access their functionality.

Figure 6.1 displays the system architecture of the *core system*. The core system consists of the components that are necessary to read, process and send data to the client. The plugins and the plugin system do not belong to the core and resemble extensions to the system.

6.1.1 Data parser

The data parser implements the connection of the system to the data source via `Source Connectors`. These source connectors implement different ways of obtaining the data. This might be via a push-based (for incoming data streams), pull-based (polling of certain web-resources) or file-based (polling a file for changes).

During the development phase, a file-based source connector was implemented, since there was no access to the original data source. There was also the need to create a data source that simulated the continuous data flow into a file. Later the mock up data source was extended to also provide push technology to an incoming port.

The events obtained by the source connectors are transformed into an internal event format and published by the `EventEmitter` component. This component's purpose is solely to offer the two functionalities of publishing events to be picked up by other components, and offering an interface where other components can subscribe to the event stream.

Between parsing and publishing of the events, the data is converted into internal event format by a `Wrapper` component. Such a wrapper provides a simple interface that implements a single *convert(originalEvent)* method to transform the events and return the data in internal format.

Figure 6.2 shows the UML component diagram of the parser implementation and the connections to other components.

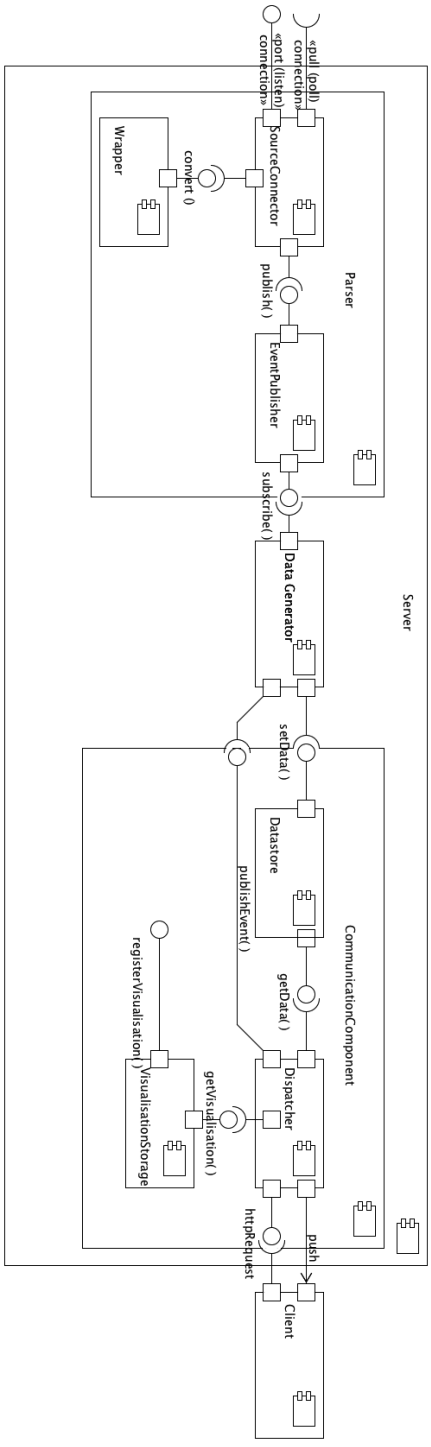


Figure 6.1: UML component diagram of the server system architecture

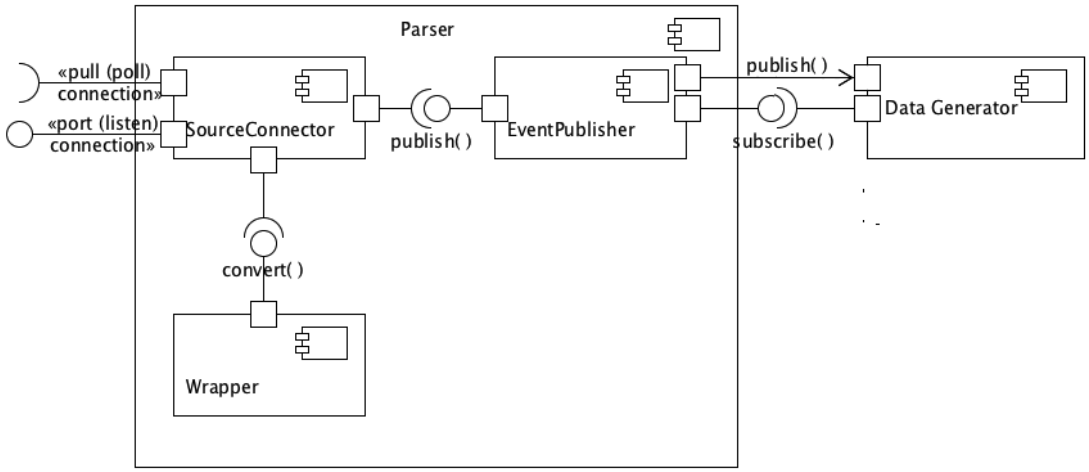


Figure 6.2: UML component diagram of the `parser` component.

6.1.2 Data generators

A `data generator` is a component that subscribes itself to certain events and performs actions when these events are observed. The 'standard' actions were to add the data to a certain data store or to update some information (e.g. current game time). However, other actions like calculation of key numbers or logging of information could be implemented too. Optionally the data generator can trigger an information push to the client.

6.1.3 Communication component

The communication component is responsible for handling the information exchange with the client. For this purpose the component maintains a `Datastore` component, which handles the storage and fetching of the data. The data store holds two types of data: on the one hand the `Gamestate` data, which stores the current information about teams, players, score and game time, and a second storage part for information that can be fetched for other client representations. The second part, called `Statistics data` is a key-value storage and every component is ought to have a different key, so that there is no interference of the data. The data format of the storage is JSON [Cro13, Cro06]. This data format facilitates the storage, retrieval and update of primitive types, lists and dictionaries and can be converted natively between string and object format.

Another subcomponent is the `Visualisation storage`. This class maintains the

set of available visualisations (names of visualisations, data store keys, callback methods). If a visualisation is requested, the dispatcher component first fetches the data from the `Datastore` and afterwards the callback function, that is executed when the data arrives at the client. There is further information sent to the client, such as the previous and next visualisation in the list, so a quick navigation between the individual visualisations is possible.

The data transfer to the client can happen in two different ways. The first one is a standard HTTP request-response handling that happens when the client first calls up the web page. The server responds by sending the *index.html* file content to the client. The other option is to use asynchronous techniques for requests and push updates. This means that a client sends off an asynchronous call[1], requesting the server to transfer data to the specific client. Next to the server request, the client also installs a callback function that is executed when the response data arrives. While the client is waiting, the server fetches the required information and answers with an asynchronous response message to the client. It is important to remember that the server can also push information to the client by publishing events that have not been requested. The trigger for this is usually the availability of new `Gamestate` data (goals, current time update, substitutions, etc.).

Figure 6.3 displays the UML component diagram of the communication component and its interfaces to other parts of the application.

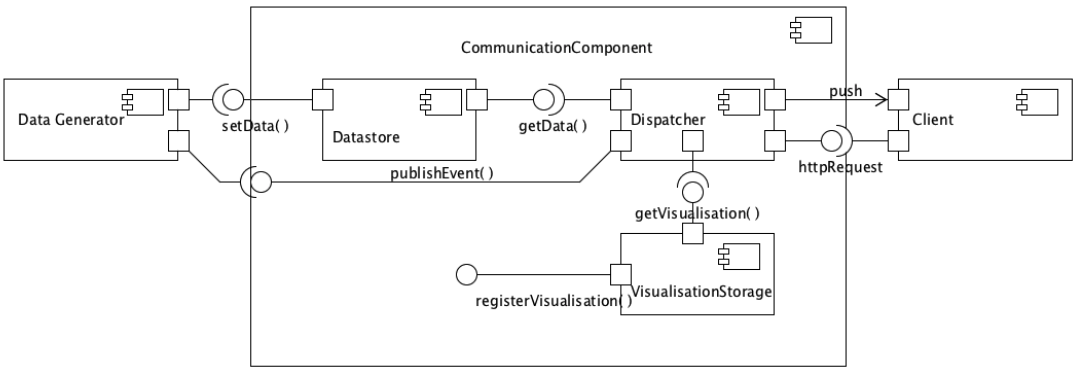


Figure 6.3: UML component diagram of the `communication component`.

[1] see Section 2.6.1 for more information on asynchronous technology

6.2 Technology Evaluation

The following section explains the evaluation of different technologies and web server implementations. The type of web server ultimately also affected the programming language for the server development. The descriptions list the information that has been valuable during the decision phase. All evaluated web servers are open source implementations and available for free.

6.2.1 XAMPP (Apache HTTP Server)

XAMPP is a software bundle consisting of Apache HTTP Server, MySQL, PHP and Perl, and some further, smaller tools and modules. Its purpose is to provide a web server that is easy to set up and use on different operating systems.

XAMPP and PHP are both well-known and widespread on the internet and serve many different applications. However, this project will result in an application that is client bound and does not require the dynamic generation of web content, that is featured by PHP. Another disadvantage is that XAMPP is a synchronous implementation and did not succeed in solving the C10k-problem. This means that the performance decreases and/or the server crashes when trying to handle handle more than ten thousand simultaneous connections. This makes the server infeasible for our needs.

6.2.2 Apache Tomcat

Apache Tomcat is a web server and web container that is capable of executing Java Servlets and JSP applications. This makes it appealing, since the Java programming language, that is mostly used in desktop applications, can also be used to create the server side of a web application. Thereby the object-oriented concepts and the large number of libraries and frameworks are accessible as well. However, in terms of performance, the Tomcat web server faces the same problems as XAMPP.

6.2.3 nginx

Nginx (pronounced "*engine x*") is a high performance server that has been developed with the idea to counter the C10k-problem. It is extremely resource efficient and easily scalable. Many of the large web platforms use nginx to serve thousands of simultaneous connections. Detailed information about nginx's performance in terms of load balancing is evaluated in [CLNW12].

A big disadvantage of nginx in the available version was, that it did not support web sockets. This technology would have allowed lasting connections between server and client and therefore could have been used for push updates.

6.2.4 node.js

Node.js is a server side technology that uses JavaScript as the programming language. It is widely used for web applications. Node.js is, like nginx, one of the server implementations that successfully overcame the C10k problem. There are many different modules available to extend the functionality of the core application. In contrast to nginx node.js supports the use of web sockets (after installation of the socket.io package). Another advantage is the usage of JavaScript. This feature provided the possibility to re-use the same code (with minimal adaptions) on the server and client, hence facilitating the development.

We decided to use node.js in our application for the two latter reasons. Nginx was still being seen as a candidate that could, if configured correctly, run faster and serve more simultaneous connections. This was of course under the condition that a future version of nginx would support web sockets or a similar technology to push information to the client.

6.3 Communication technology

The communication between server and client is performed using the `socket.io` library [Rau12]. This library establishes communication between a server and a client and supports bidirectional message passing. Thereby it does not matter whether the client sent a request or not. Socket.io supports the technologies of Adobe Flash Socket, web sockets, AJAX long polls, forever Iframes, and JSON-P Polling. The library has a node.js package and can easily be integrated into the client with a simple JavaScript API.

The big advantage of this library is that it establishes the best communication path on its own. After establishing the connection between server and client, the library automatically determines the most efficient technology and uses it for future communication. It further supports features such as `heartbeats`, `timeouts` and `disconnection management`, making it easy to use and extremely stable.

6.4 Plugin system

To assure the extensibility of the system, the application provides an interface where plugins can be easily added by simply providing a set of classes and configuration files in a certain location. It was important that these extensions were independent from each other and could only access and modify 'their own' data.

6.4.1 Plugin parser

The basic working principle of the `Plugin parser` is to search through a specified directory for plugins. A plugin is a subfolder of the directory that contains a special configuration file. This configuration file stores details about the plugin, such as information about its name, its data generator class and visualisations that should be added. The information is stored in JSON-format. Listing 6.1 shows an example code listing for the `heatmap`-plugin.

Listing 6.1: An example for a plugin configuration file

```
{"name": "heatmap",
  "datagenerators": [
     {"lib": "HeatMapListener", "classname": "HeatMapListener"}
  ],
  "visualisation": {
        "type": "visual", "callback": "file:uiCallback.js"
  }
}
```

The above listing specifies one data generator class `HeatMapListener` that is defined in a library of the same name. The HeatMapListener also features a visualisation of type `visual` (an identifier used for grouping of visualisations and navigation inside the popup/info area panels). The `callback` is a routine that is executed when the visualisation is fetched from the server. It can be a JavaScript function or, as in this case, an entire JavaScript file (see "file:" flag). On client request, this file is parsed and transferred to the client for execution. It is possible to reference further resources inside the `Callback.js`.

To ensure the safety and stability of the overall system, the registration of visualisations and data generators, as well as calls to plugins have to be surrounded with `try/catch` statements and checked for possible errors during execution. The same strategy is applied when it comes to storing and retrieving information from the data stores.

6.5 Summary

This chapter introduced the server architecture that has been developed for this project. Each component (*Data parser*, *Data generators*, *Communication component*) is described thoroughly including implementation details and purposes. The section about the technology selection gave insight about the reasoning process between four possible server technologies (XAMPP, Apache Tomcat, nginx and node.js) and justified the selection of *node.js* for the implementation. The last two sections explained the communication principles (using socket.io) and described the concept of the *plugin system* that was developed to support an easy, quick and safe extension of the system.

Chapter 7

Testing & Evaluation of the Prototype System

The system evaluation was an essential part of the project. This chapter provides information about the evaluation process. At the end of the implementation phase, the performance of the implementation was analysed (see Section 7.1) and compared to the original requirements (more than ten thousand concurrent connections). Furthermore a *scale out* strategy (see Section 2.7 in Chapter *Background*) was implemented and the performance compared to the original implementation.

To ensure the correct display of the user interface in many environments, the client was executed on a number of different browsers and operating systems (Section 7.2) and verified for its correct display.

The last part of the chapter summarises the results of a questionnaire that has been given to a test user group at the end of the implementation phase (Section 7.3). This test was conducted to gather opinions and provide user feedback to the BBC.

7.1 Performance testing

The BBC targets a broad user base and its web and smart phone applications serve millions of user requests daily. This fact leads to the assumption, that a live version of this application will most likely have several thousands of users too. It is crucial to the success of the software to verify its effectiveness based on these performance measurements. While this is a prototype project, and hence maximum performance is not a direct requirement, an investigation of possibilities to increase effectiveness had to be conducted to provide a guideline for possible improvements in future projects.

To evaluate the performance of the application, it was necessary to determine the parts of the application that would become bottlenecks when the number of users is increased. In the case of this software it is the *CommunicationComponent* (see Section 6.1.3) that provides the interface to the client side and therefore is most likely to encounter peak performances when many clients try to connect.

7.1.1 HTTP performance

A short analysis of the information flow reveals, that the clients' HTTP requests are always answered with the same content. The server responds with an HTML file and a handful of CSS files and JavaScript libraries.

Upon receipt of this data, the client displays the HTML and executes the JavaScript. The JavaScript code initiates the connection to the server and requests the sports data. This data is rendered by the browser on the client side.

This leads to the conclusion that the HTTP server plays a minor role in the setup. This part can easily be scaled out by placing one (or more) web caches between the server and the client. This approach is well known and has already been topic of extensive research. An overview over web caches can be found in [CQRD03].

7.1.2 Websocket/Socket.io performance

A more significant problem for the application is keeping thousands of websocket connections alive and responding to requests with minimal delay. As the definition of the *C10k problem* states the requirement of supporting ten thousand connections on standard hardware, the initial tests were performed on a five year old (*'Late 2008'*) Apple MacBook running on a Intel Core 2 Duo CPU @ 2.0 GHz with 4 GB 1067 MHz DDR3 memory and OS X 10.8.4 operating system.

7.1.2.1 Test setup

To conduct the performance tests, the following setup was chosen. The tests were executed in the School of Computer Science's "MSc Lab". A computer cluster of 20 to 30 Linux machines was created using SSH connections and each computer executed the same client-program. Each of these clients[1] requested a new connection every 50/100 milliseconds until a specified limit was reached. Figure 7.1 displays a visualisation of the used test setup.

[1] the node.js library *socket.io-client* was used to perform headless client tests

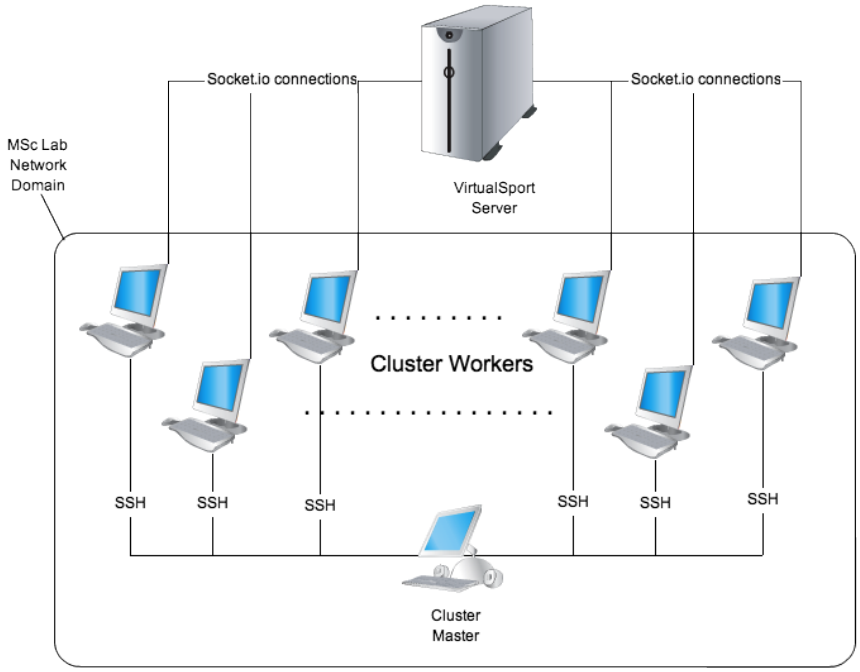


Figure 7.1: Test setup in the MSc Lab.

The cluster master triggered the execution of the client programs on the 20 to 30 worker machines via SSH connection.
Each worker established a specified number connections to the VirtualSport server.
The VirtualSport server also executed a client program, responsible for taking the measurements and increasing the workers' number of connections.
Hence, the worker programs provided additional work load for the server, but did not perform any measurements themselves.

The VirtualSport server was extended to accept the clients and broadcast incoming messages to all connected clients. Furthermore the server included the total number of active connections in its broadcasts. Apart from the application, the server machine also hosted a client program. Every second this special (measurement-)client emitted a timestamped messages. On arrival of the server's broadcast of this message, the client calculated the difference between arrival time and the time in the message data and printed this value, together with the total number of connections on the server and allocated CPU and memory to a file.[2] At regular intervals (15, 30 or 60 seconds) the measuring client emitted a certain *increase-connections* message which triggered an increase of the connections-limit in each cluster machine. To assure that the connections-limit in each worker machine did not increase indefinitely,

[2]The reason for having measurement-client and server on the same machine was to avoid measurement of the network/transport delay.

the measurement-client was configured to send only a certain number of increase messages.

Example: *A typical test run increased the connections limit in each cluster-client every 30 seconds by 100. Assuming there were 20 machines in the cluster, the server had to deal with 2000 new connections every 30 seconds.*

The measurement client was configured to send 6 increase messages. This means that every cluster client received six messages specifying an increase of 100 clients each. This lead to 20 clients having 600 connections each, reaching a total maximum of 12000 connections.

Unfortunately the computer cluster setup was not always stable during the tests and sometimes the workers' SSH-connections broke and all the machine's connections to the server were lost. In the following diagrams this behaviour is displayed as a nearly vertical drop of the connections graph.

7.1.2.2 Performance of the original implementation

Figure 7.2 shows three typical test runs on the test system described in Section 7.1.2. The curves represent the server's total number of active client connections ("Run X[3] connections"), the response time ("Run X RT"; measured by the *measurement-client*) and the CPU-usage in per cent ("Run X CPU").

The test runs in Figure 7.2 show the first increases in client numbers without any long delays. The number of connections rises rapidly and no connections are dropped. Interesting however are the peaks in the application's CPU usage during the connection number increases.This behaviour is due to the establishment of 2000 new connections, while having the workload of keeping the existing connections alive and handling the broadcast of the measurement message in one-second intervals. The increased activity can also be observed in the higher response times during connection increases. While the majority of response times measurements are under 10 milliseconds in the first 2 minutes and below 50 ms for the rest of the test run, during the CPU usage peaks individual messages take up to 700 ms for processing and broadcast.

In all test runs the allocated memory was under 20 MByte. As this value is very small when compared to modern server machines' resources[4], this value does not limit the performance and will not be further discussed.

Test run 1 (red) displays near perfect execution for the first 120 seconds. The

[3] X is a place holder for the number identifying a certain run

[4] modern servers tend to have dozens of GByte of RAM

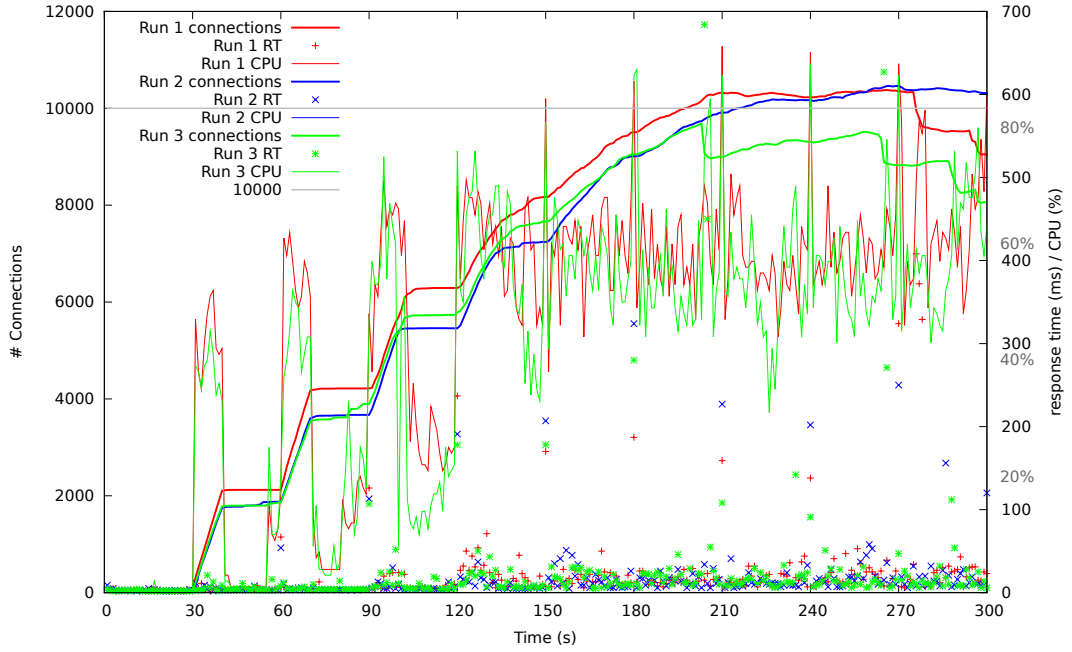


Figure 7.2: Test runs on a *Late 2008 MacBook* (2.0 GHz, Core 2 Duo)

The (bold) graphs show the total number of connections over time while the number of connections is increased in 30 seconds intervals.
The thin graphs show the CPU usage, the points mark the response times of the system.

slightly higher number of connections (>2000 after the first increase) is attributed to the 'maintenance connections' that are used to conduct the tests and trigger the increases. As one can see, the typical spike in CPU usage happens at every increase. At about 120 seconds the graph starts to flatten and loses the typical step-like shape. This happened because the number of existing connections required too many resources and the growth in connection numbers slows down. The test run reaches its maximum number of active connections after 200 seconds, where the server handles about 10500 connections. This value is kept constant (with minor fluctuations) until one of the cluster-client machines' SSH connection breaks (around 270 sec.) and the number of connections decreases. A second client is lost shortly before the end of measurements.

Test run 2 (green) displays a similar behaviour as *Test run 1*. As this test was conducted on a cluster with only 19 workers, naturally the total number of connections grows slower. The graph, having the same typical shape as *Test run 1* reaches a total maximum after 270 seconds ($\sim$ 10500 connections) but already exceeds the ten thousand connection border after around 220 seconds. The test run ends after 5 minutes relatively stable at $\sim$ 10400 connections.

Test run 3 (blue) confirms the observations in the first two datasets. However, the SSH-connections of three cluster machines drop after 190, 255 and 285 seconds, respectively, leaving the maximum number of connections of this test run at around 9600. It is safe to assume, that the curve of this test run would follow a similar shape as the one of *Test run 2*, if the SSH-connections were not broken.

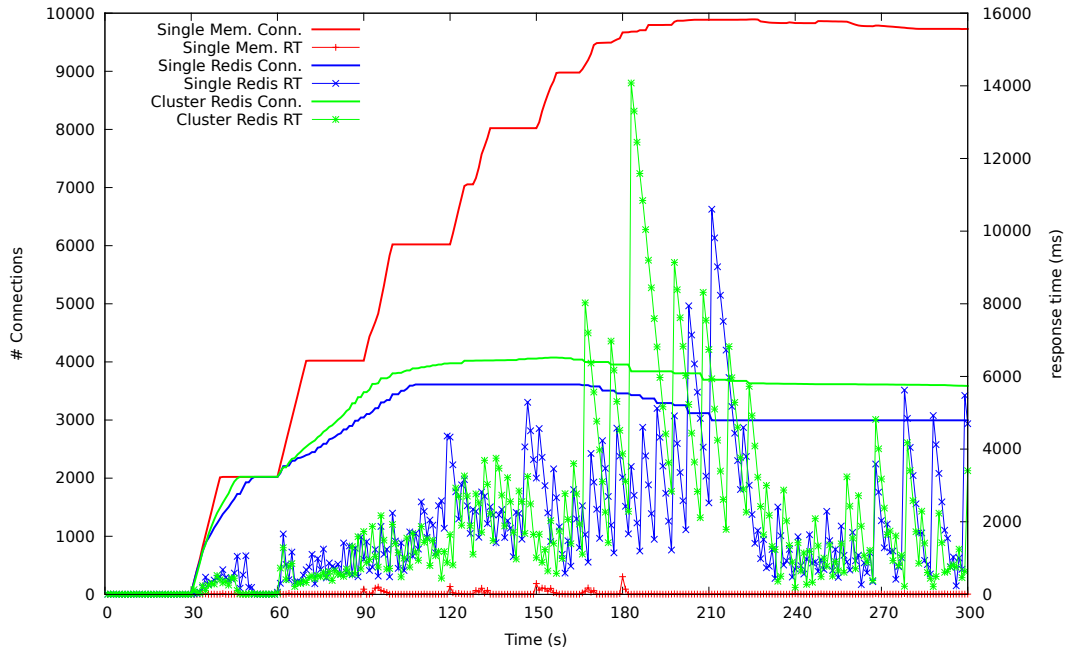


Figure 7.3: Comparison of single and cluster (multi thread) execution on a *Late 2008 MacBook* (2.0 GHz, Core 2 Duo)

The plain graphs show the total number of connections over time.
The graphs with points mark the response times of the different forms of execution.

7.1.2.3 Increase the number of connections using clusters

There are several ways to increase the number of connections served. The approach evaluated in this project is the usage of the node.js internal *cluster* library [Nod13a]. This library allows the forking of the process and to run several equivalent process instances. Whereas this is a straight-forward implementation for standard HTTP-requests, sharing Socket.io connections between different threads is not as simple, as Node.js does not offer any shared memory variables between the threads of a process. The solution proposed by the Socket.io team is to use a Redis database server[5], that

[5]Redis is an asynchronous key-value data store

stores information about the connections. Everytime a Socket.io server instance receives a message from a client, it looks up the connection in Redis and verifies its validity, client-handshake and status. Naturally, the communication with the Redis server takes longer than the previous in-memory data-lookup.

Figure 7.3 displays three graphs that represent the different execution types of the adapted application. The red line displays the number of connections during "normal" (single threaded, in-memory data store) execution. The slightly lower number of connections is explained by the fact that during this test run the Redis server was running in the background to provide the same execution environment for all test runs.

The blue graph represents the system behaviour when the in-memory data store is replaced with the Redis data store for the connection management. As previously described, the connection handling slows down and broadcasts times increase from under 100 ms to between 0.5 to 6 seconds, with occasional peaks of up to 11 seconds. The maximum number of connections peaks around 100 seconds at $\sim$ 3500 and falls back to 3000 after 3 minutes.

The green graph represents the two-threaded cluster execution. As in the previous test run, the Socket.io connections are handled in the Redis data store. The interesting part is the increased number of total connections. This number peaks at $\sim$ 4000 and drops slightly towards the end of the test run. This behaviour is explained by the split workload. The computers' cores are evenly used and the number of possible active connections is higher. However, the server machine only has two CPU cores and the execution of two VirtualSport server threads, the Redis server and the measurement-client application removes the possibility of an even higher connection count.

7.1.2.4 Execution on a dedicated multi-core server

To draw conclusions from the observations of the cluster system, the tests were run on a dedicated server machine. The dedicated server was a virtual machine run on the School of Computer Science's VM host. The VM has 8 cores, running at 2.67 GHz each[6].

Figure 7.4 displays the test results on this system. As the red line shows, the single-threaded execution with in-memory data store peaks slightly below 16000 connections and continues stable thereafter. The blue graph, representing the single-threaded test

[6]The VM is a CentOS 6.4 Linux system, hosted on a Dell PowerEdge R710 with 2 Intel Xeon E5530 @ 2.4GHz (provides 16 virtual cores) and 24GB memory

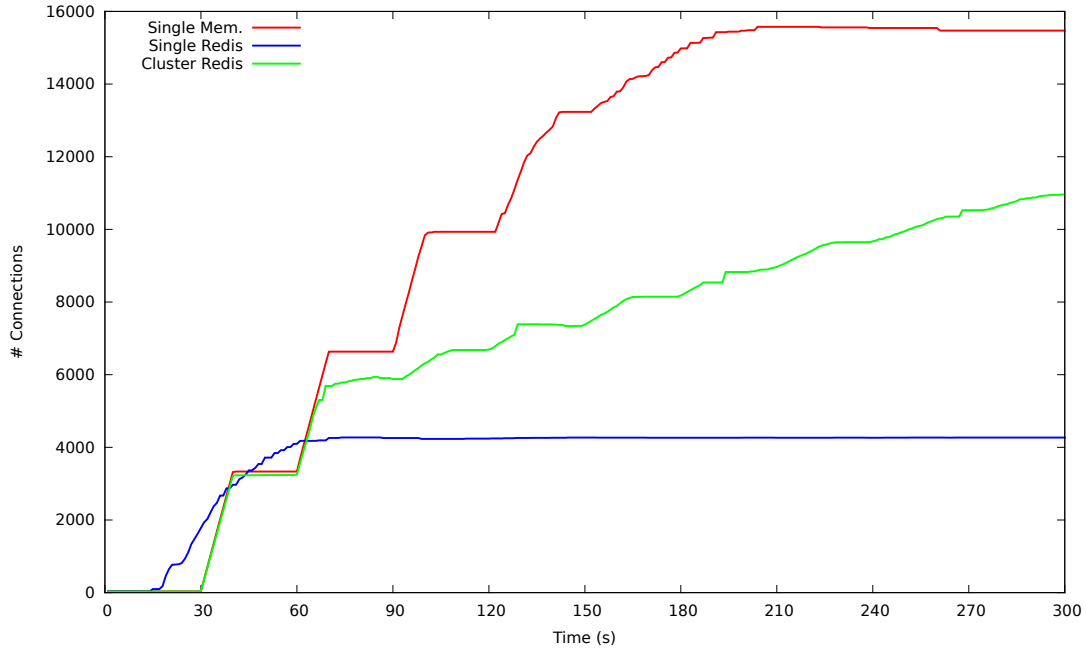


Figure 7.4: Comparison of single and multi thread execution on a Virtual Server machine (8 cores @ 2.67 GHz)

The graphs show the server's total number of connections over time, while the number of connection requests is increased incrementally.

run with Redis data store shows a similar (slightly better) behaviour as in the execution on the slower test machine (MacBook). The big difference between the run on the MacBook and on the VM is the cluster-execution (green graph). Due to the eight available cores, the task load is split evenly between eight threads and uses all the available CPU resources. The test run ends after 300 seconds where the cluster successfully served just under 11000 connections. Interestingly though, it takes the cluster execution longer to rise to a high number of client-server-links. An explanation for this behaviour is the increased time for the registering of the new connection in the Redis data store.

The response times of the VM system are similar to the ones on the MacBook and have not been plotted to allow a better view of the connection number graphs.

7.1.2.5 Connection number conclusion

The conclusions drawn from these experiments were that Socket.io applications are well scalable and given enough resources available, the number of client connections

is expected to be raised to a multitude of the single-threaded execution. One problem faced when executing applications on clusters was the necessity of dedicated data stores and the thereby highly increased response time of the system. Unfortunately this system setup lead to much (up to 100 times) slower response times. This topic, while still being in the boundaries of the project definition (<15 - 20 seconds) will definitely serve as an interesting challenge for an eventual future application in a production environment.

As this system can be configured to run on two independent machines (rather than two CPU-cores), the application can assumingly also be scaled behind a proxy server across a large number of machines. This would remove the necessity for having uniform CPU cores but introduce new load balancing issues.

7.2 Client execution in different environments

One of the research goals was to develop an application for as many runtime systems as possible and to support devices impartial from their operating systems. All client technologies (JavaScript, HTML5, CSS) chosen for the prototype are supported by the major browser manufacturers. The JavaScript libraries (jQuery, Kinetic.js and smoke.js) and the Socket.io communication library have been created and tested in different environments to support mobile and desktop environments likewise.

The application was developed on a MacBook running OS X 10.8 Mountain Lion and first tested on Google's Chrome browser. To assure compatibility with other browsers, the application was regularly tested on Mozilla Firefox and Apple's Safari browser. As execution in mobile environments is one of the requirements specified, the client was also executed in mobile environment such as Apple's Safari browser on iOS (running on an iPad mini). The usage of third party software libraries that have been developed to be executed on all these runtime systems greatly facilitated the development process.

After implementation, the application was executed in as many environments as possible to ensure maximum compatibility. The following paragraph describes the key findings of the evaluation.

All targeted browsers in desktop environments and operating systems support the application without any major problems. Table 7.1 displays a matrix of the browsers and operating systems tested for compatibility. The result was that all tested browsers fully support the application.

Browser / OS	(Mac) OS X (10.8)	Linux (Fedora)	Windows 7
Google Chrome (v. 28)	verified	verified	verified
Mozilla Firefox (v. 22)	verified	verified	verified
Apple Safari (v. 6)	verified	not available	not available

Table 7.1: Browser - Operating system compatibility test matrix.

The table shows the browsers and operating systems on which the correct functionality has been verified.

In mobile environments the main target platforms were *Android* (version 4.2) and Apple's *iOS* (version 7). Both systems have been tested in their latest version with their standard browser. The first test performed was on an Android device with its standard browser Chrome. To not have any trouble with interfering software, the application was run on a Nexus 10 tablet computer with the latest stock Android version 4.3 and the latest version of Chrome without any plugins. The test was successful and the application was fully supported[7].

For a second test, the application was executed on older, weaker hardware. Unfortunately this test was less successful, as the Samsung Galaxy Nexus[8] smart phone's hardware was not strong enough to support the chosen implementation. The device was capable of displaying static graphics of the football pitch but could not cope with the frequent updates of the HTML canvas-object which lead to browser crashes.

The next mobile test was conducted on an Apple iPad mini running a Safari web browser. This test was very successful and the user interface updates were handled at normal speed. Merely when the server was set to push the game updates at ten times the normal speed, the browser performance started to slow down and eventually resulted in a crash when the speed was increased to the twentyfold of normal.

7.2.1 Conclusion of multi-browser evaluation

While all desktop browser systems fully support the application and there were no observable issues, the mobile execution occasionally seemed to produce problems, especially on weak, outdated hardware. Therefore it is necessary to reduce the requirements of frequently redrawing the display. In a possible future version of this software the client side should be optimised to only update visible objects, while hidden and

[7] During the first test some minor problems with the touch support occurred. These problems were related to differences with Kinetic.js's 'click' and 'tap' event handlers

[8] running the same software as the Nexus 10 tablet

background objects should be disconnected from server updates.

Summarising, the tests were a success, since the main requirements ("mobile execution" and "environment independence") were fulfilled. The minor problems concerning the user interface performance issues are not highly important as this project served primarily as a feasibility study and prototype project.

7.3 User evaluation

A small user group was invited and their opinions of the current state of the software were evaluated and analysed. The focus of these evaluations was based on the general concepts, such as usability, the visualisation of live data and the "interestingness" of statistical visualisations. Of further interest were the users' thoughts on when and where they would use the software and their expectations of a future version of the software. It was important to steer the tests towards these questions, as at the time of user evaluation the application was in a prototype state and therefore showed deficits in terms of 'completeness' and 'cleanness' of the user interface.

7.3.1 Evaluation setup

The test group, consisting of five computer-literate sports fans was invited to a guided introduction of the application, followed by a request to share their opinion in an anonymous web survey. The setup was intentionally kept informal and simple to motivate the users to share their honest opinion. The group was informed that the program was a prototype application and that their opinion would effectively influence future developments.

The test group was asked into the "MSc laboratory" in the Kilburn building, where the tests were conducted. After a short introduction about the aim and outline of the project, the users were introduced into the functionality of the application. This was followed by some 'unguided' time (about 10 minutes), where the users could try using the software by themselves. Directly after this unguided time span, the users were asked to share their opinion by filling out an anonymous web form.

Although it was not officially invited, some users started a short discussion after they filled out the questionnaire and gave oral feedback.

The anonymous questionnaire results can be found in Appendix C.

7.3.2 Summary of evaluation results

Environment of usage The first question asked the users whether they would 'try this application', use it 'occasionally', 'regularly' or 'as often as they could'. This was followed by a question where they could justify their answer. Most of the users stated that they would 'give the software a try' if they had the chance to, where one user stated that he would use it on a regular basis. The justification for using the application was the possibility to access statistical information that is "usually displayed only during half time break or after the game ends". The users stated that the reason for not using it on a regular basis is that the live pitch only visualised "a small part of the action (only around the ball)". When asked in which situations the users would use the app, all of the users answered that they would use it mobile. Some answered that they would also use it in addition to watching the television broadcast or in public places such as restaurants and bars.

Visualisations feedback Regarding the visualisations, the test group had a split opinion, whether the live pitch or the statistical visualisations were most appealing. None of them chose the formations pitch as the most interesting part. The reasons for favouring the live pitch were the possibility follow the game. Justifications for choosing the statistical visualisations included the access of information that is not available on traditional television broadcasts (or only at certain times). All of the users stated that they understood the individual visualisations. One user mentioned that some were not as self explanatory as they could be, another user requested the feature to filter the events for each half of the game separately.

Social media and other data providers Four of the five testers stated that social media integration is a good idea in their opinion. The reason is the possibility to have discussions about the current game, specific plays and players, since "That is what we do when watch a real game! (sic)". Only one tester indicated to be indifferent to this. When asked which data sources they would like to have integrated, all of them chose the BBC web radio. Most of the users also indicated that data from 'ESPN (or similar sports platforms)' and the 'BBC sports live ticker' should be integrated.

Likes, dislikes & feature requests The last three questions were open questions, where the users were asked to state things they liked or disliked. They were also invited to share their ideas about features they would like to see in a future version of

the software.

The negative responses were statements on obvious prototype problems, such as non-optimized memory usage of the client, no information during reconnects and unclear button symbols. Other answers included complaints about the shortage of position data ("the ball seemed to jump from the its last location"; "live broadcast[9] need more data to be realistic").

The users indicated to like the live pitch representation of the game and the concept of accessing statistical data at any point during the game. Interestingly, one user mentioned to like the integration of the Twitter widget, although none of the users chose Twitter in the previous question about data providers, that should be integrated.

The last question asked for the users' feature wishes. Most users expressed in the questionnaire and also in the discussion afterwards the request to display more data and the positions of all players on the pitch. Further wished features included radio commentary, information about *benched players*, "traditional statistics"[10] and an *event history*[11], as well as a *chat*-function to exchange opinions with others.

7.3.3 Conclusion of user evaluation

The prototype in the current form seems to attract user attention. The users stated that the integrated features were appealing and interesting, but that the amount of presented information was too little. Some of these issues can be solved by integrating more data sources (radio, third-party web services, social media) and displaying statistical data in forms of tables and legends. Other features are not that easy to implement, as the available data does not allow full player tracking, etc. It appears that especially the availability of statistical data at any point of the game is interesting to the users, which indicates that the test group would use the application on a second screen while watching the game on television. However, all of the users stated that they would use the application at times when they could not follow the video broadcast. It seems that a future version of the application should continue to include both parts of the application (live pitch and statistics), as depending on the situation (mobile or second screen usage) the focus shifts between these visualisation forms. It is important to keep in mind, that the test group size was deliberately chosen to be small and therefore the results may not represent the opinions of a large audience. Hence, further user evaluations should

[9] i.e. the live pitch
[10] i.e. numbers and tables displaying counts and statistics in percentages
[11] a list that keeps a log of the events

be carried out, after more functionality has been implemented. (See Section *Future work* in Chapter 8)

7.4 Summary

The previous chapter describes the performance analysis of the server implementation, as well as the changes made to support cluster-execution of the program. It is shown that the software can overcome the *C10k problem* even when the server is run on an outdated personal computer. The second part of the performance analysis displays clearly that a parallel execution of multiple server instances proves to be less efficient on a small scale, but allows the continuous growth through easy scaling possibilities.

The next section describes the process of testing the client application in various environments throughout the implementation phase, as well as after implementation stop. A summary of the test lists the target browsers and operating systems and explains the problems that occasionally arise, especially on weak hardware.

The last section summarises the findings of the user evaluation and describes the users' opinions on existing features and wishes for new ones. An analysis of the questionnaires states that the program is likely to be used and, provided it displays interesting facts and data, will gain the users' acceptance.

The conclusion drawn from these tests is that the prototype is a successful first implementation with interesting and appealing visualisations that can be executed in many browsers, on different operating systems and devices and that the server implementation offers good performance and scalability options.

Chapter 8

Conclusions and Future work

The following chapter summarises the key aspects of the dissertation. Section 8.1 summarises the conclusions of this project. The last section (Section 8.2) gives an outline of possible future projects and describes research questions and problems that arose during the project.

8.1 Conclusions

Analysis of live sports data has recently gained growing interest and is likely to continue to become more important. This project has proven the feasibility of VirtualSport, a web application that offers visual information about live sports events.

We have shown that the task of creating software such as VirtualSport requires the combination of numerous research areas, that have been explored before. For the implementation of the use cases, we had to analyse existing solutions in the fields of event-driven and mashup architectures, data stream querying, and real-time and push communication.

In the course of this dissertation it was shown that the Opta sports data, which is available to the BBC, provides enough data quality and quantity to extract interesting information. A thorough analysis reveals that the amount of data is even high enough for the display of graphical live information that permits the user to follow the game in real time. Further display options include statistical information and game data, such as a player's activity areas or passes.

A prototype implementation of the software serves as a proof-of-concept and shows a small number of visualisations (including the live pitch). This dissertation documents the features of the user interface and explains the functionality of the visualisations to

provide usage examples. The technologies for the implementation of the client had to be portable and flexible. It is shown that HTML5 and CSS in combination with JavaScript and frameworks such as JQuery and Kinetic.js prove to be reliable choices for this task. Executions of the client on various browsers and platforms have shown that the user interface is compatible with all targeted execution environments, namely Chrome, Firefox and Safari on (Mac) OS X, Linux, Windows, Android and iOS.

Based on the background research and data analysis we proposed an architecture that was capable of employing data stream and event-driven architectures to handle large amounts of data efficiently. A detailed discussion of possible server technologies was provided, followed by an explanation of the server architecture. We used Node.js for the implementation of the prototype-server and illustrated its capability of serving more than ten thousand clients using Socket.io communication technology. These tests have been performed on slightly outdated, off-the-shelf personal computer hardware and it was shown that an execution on modern server machines leads to even higher efficiency.

Alongside the evaluation of the original implementation, the results of a cluster execution are revealed. The tests show good scalability and suggest that this form of execution offers the possibility of serving client numbers that are at least one magnitude larger than the original software.

A user evaluation verifies that the software is appealing to consumers and provides feedback on the prototype for the BBC. The results state that the prototype is interesting to them and the users asked for more features. The user group expressed interest in a future version of the software, with some caveats. This suggests that further work on this software could lead to a program that is widely accepted as a second screen application or even a mobile replacement where the television broadcast is unavailable.

This project obtained a valuable objective analysis of the employed technologies and provided the groundwork for further research in this area.

8.2 Future work

In the course of this project several interesting research topics have been discovered that would serve as possibilities for follow-up projects.

Scalability Probably the most challenging topic lies in the area of scaling out. Although this project included some basic information and experiments about the scaling

of the server-technology, these ideas could be further explored, tested and verified to ensure stability when it comes to serving larger numbers (more than one hundred thousand) of clients. The challenge is to optimise the large-scale architecture to have redundant servers and data storages that provide reliability on the one hand, while still offering the same responsiveness and performance on the other.

Time correctness During the performance analysis, tests have shown that the response time of the server can vary between as low as 100 milliseconds up to 10 seconds when using data stores such as Redis. This means that eventually the data could arrive in "bursts" which would reduce the correctness of the live pitch visualisations. For a future application like this, it is necessary to investigate mechanisms that allow the data to arrive with constant delay times to make sure that the visualisations on the pitch are veritable.

Other data sources and more visualisations The visualisations in the prototype serve as examples of the capabilities and functionality. However, further research could investigate other visualisation opportunities to provide a reasonably large set of statistics for each player. To exploit the maximum amount of information from the data, visualisations could also include statistics for the entire team, benched players and perhaps even managers and referees. Other visualisations could investigate the possibilities of displaying the goals and the end point of a shot to show where a player places his attempts. A visualisation that has been requested during the user evaluation asks for the list of events and the possibility to display them individually.

More focus should also be put into finding other data sources that provide player position data. As mentioned in the data analysis section, the BBC is currently working on applications that obtain such information from common TV footage. Accessing these data providers could eventually lead to full player tracking possibilities and hence more visualisation options.

Social media integration Social media integration is one of key factors to the success of modern applications. Especially in applications like this, where users could exchange opinions about sports statistics, the direct integration of Facebook, Twitter or similar platforms could provide a large advantage and draw many consumers to the application.

Other content providers As suggested, the inclusion of content from third-party providers should also be assessed. Especially other sports data platforms (ESPN, etc.), audio commentary and other multimedia content should to be analysed and evaluated for integration.

Pilot phase The consumer evaluation gave valuable insight into the users' opinions of the application. However, it is necessary to extend these tests on a larger scale, if the work on this application continues. The pilot phase could involve a larger user group (hundreds to thousands of actual users) and be run on live data feeds, rather than data source simulations. This study could provide valuable insight that can be used for the further development of the prototype.

Bibliography

[ABC+03] Jürgen Assfalg, Marco Bertini, Carlo Colombo, Alberto Del Bimbo, and Walter Nunziati. Semantic annotation of soccer videos: automatic highlights identification. *Comput. Vis. Image Underst.*, 92(2-3):285–305, November 2003.

[ACc+03] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB J.*, 12(2):120–139, 2003.

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, Berlin, 2004.

[ACL+12] David J. Anderson, Giulio Concas, Maria Ilaria Lunesu, Michele Marchesi, and Hongyu Zhang. A comparative study of scrum and kanban approaches on a real case study using simulation. In Claes Wohlin, editor, *XP*, volume 111 of *Lecture Notes in Business Information Processing*, pages 123–137. Springer, 2012.

[Adm96] United States General Services Administration. Telecommunications: Glossary of Telecommunication Terms (Federal Standard 1037C). `http://www.its.bldrdoc.gov/fs-1037/fs-1037c.htm`, August 1996.

[Ado13] Adobe Systems Inc. Adobe flash. `http://www.adobe.com/products/flash.html`, 2013.

[Aga10] Sachin Agarwal. Toward a push-scalable global internet. *CoRR*, abs/1011.6129, 2010.

[Aga12] Sachin Agarwal. Real-time web application roadblock: Performance penalty of html sockets. In *ICC*, pages 1225–1229. IEEE, 2012.

[And03] D. J. Anderson. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. The Coad Series. Pearson Education, 2003.

[AS13] C. Anderson and D. Sally. *The Numbers Game: Why Everything You Know About Football is Wrong*. Penguin Books Limited, 2013.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis, editors, *PODS*, pages 1–16. ACM, 2002.

[BDK+02] Manish Bhide, Pavan Deolasee, Amol Katkar, Ankur Panchbudhe, Krithi Ramamritham, and Prashant Shenoy. Adaptive push-pull: Disseminating dynamic web data. *IEEE Trans. Comput.*, 51(6):652–668, June 2002.

[BL10] Fevzi Belli and Michael Linschulte. Event-driven modeling and testing of real-time web services. *Service Oriented Computing and Applications*, 4(1):3–15, 2010.

[BMTB10] I. Bechar, S. Moisan, M. Thonnat, and F. Bremond. On-line video recognition and counting of harmful insects. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 4068–4071, 2010.

[Bri13a] British Broadcasting Corporation. BBC Sports online platform. `http://www.bbc.co.uk/sport/`, 2013.

[Bri13b] British Broadcasting Corporation. http://www.bbc.co.uk/. `http://www.bbc.co.uk`, 2013.

[Bri13c] British Broadcasting Corporation. Who we are: At a Glance (Inside the BBC). `http://www.bbc.co.uk/aboutthebbc/insidethebbc/whoweare/ataglance/`, 2013.

[BvD08] Engin Bozdag and Arie van Deursen. An adaptive push/pull algorithm for ajax applications. In *Third International Workshop on Adaptation and Evolution in Web Systems Engineering (AEWSE'08)*, pages 95–100, July 2008.

[BW01] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, September 2001.

[CD12] Cédric Courtois and Evelien D'heer. Second screen applications and tablet users: constellation, awareness, experience, and interest. In *10th European Interactive TV Conference, Proceedings*. Ghent University, Department of Communication studies, 2012.

[CDZ06] Yi Chen, Susan B. Davidson, and Yifeng Zheng. An efficient xpath query processor for xml streams. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 79–, Washington, DC, USA, 2006. IEEE Computer Society.

[CH07] Larry Clarkin and Josh Holmes. Enterprise mashups. *The Architecture Journal*, 13, 2007.

[CLNW12] Xiaoni Chi, Bichuan Liu, Qi Niu, and Qiuxuan Wu. Web load balance and cache optimization design based nginx under high-concurrency environment. In *ICDMA*, pages 1029–1032. IEEE, 2012.

[CQRD03] B. Ciciani, F. Quaglia, P. Romano, and D. Dias. Analysis of design alternatives for reverse proxy cache providers. In *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*, pages 316–323, 2003.

[CRC+08] Meeyoung Cha, Pablo Rodriguez, Jon Crowcroft, Sue Moon, and Xavier Amatriain. Watching television over an ip network. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 71–84, New York, NY, USA, 2008. ACM.

[Cro06] D. Crockford. JSON RFC-4627. `http://www.ietf.org/rfc/rfc4627.txt?number=4627`, July 2006.

[Cro13] D. Crockford. JSON. `http://www.json.org/`, 2013.

[Dai11] Robert Daigneau. *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*. Addison-Wesley Professional, 1 edition, November 2011.

[DBLV12] Pete Deemer, Gabrielle Benefield, Craig Larman, and Bas Vodde. The Scrum Primer 2.0. Technical report, 2012.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[die13] die.net. fork(2) - Linux man page. `http://linux.die.net/man/2/fork`, July 2013.

[DLM+09] T. D'Orazio, M. Leo, N. Mosca, P. Spagnolo, and P. L. Mazzeo. A semi-automatic system for ground truth generation of soccer video sequences. In *Proceedings of the 2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, AVSS '09, pages 559–564, Washington, DC, USA, 2009. IEEE Computer Society.

[Eng13] Mark C. Engel. Node.js facebook-api. `https://npmjs.org/package/facebook-api`, 2013.

[Fac13] Facebook, Inc. Facebook. `http://www.facebook.com/`, 2013.

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. AAI9980887.

[FIF13] Fédération Internationale de Football Association (FIFA), 2013.

[Foo13] Football Association. Football Association Challenge Cup. `http://www.thefa.com/TheFACup/`, 2013.

[Fow10] Kim R. Fowler. Hard real-time systems: Construction. In *Encyclopedia of Software Engineering*, pages 361–382, 2010.

[FZ98a] Michael Franklin and Stan Zdonik. "Data in your face": push technology in perspective. *SIGMOD Rec.*, 27(2):516–519, June 1998.

[FZ98b] Michael Franklin and Stan Zdonik. "Data in your face": push technology in perspective. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 516–519, New York, NY, USA, 1998. ACM.

[Gar04] Tali Garsiel. How browsers work - Behind the scenes of modern web browsers. `http://taligarsiel.com/Projects/howbrowserswork1.htm`, February 2004.

[Gar05] Jesse James Garrett. Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications, February 2005.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*, chapter Behavioral Patterns. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[Goo13] Google, Inc. V8 JavaScript Engine. https://code.google.com/p/v8/, 2013.

[GY06] Dmitry O. Gorodnichy and Arjun Yogeswaran. Detection and tracking of pianist hands and fingers. *Computer and Robot Vision, Canadian Conference*, 0:63, 2006.

[Hen06] Cal Henderson. *Building Scalable Web Sites: Building, Scaling, and Optimizing the Next Generation of Web Applications*. O'Reilly Media, Inc., 2006.

[HJ98] Richard Howard and Bernard J. Jansen. A proxy server experiment : an indication of the changing nature of the web. In *ICCCN*, pages 646–653. IEEE, 1998.

[Hud13] Alex Hudson. Football's top teams tap into burgeoning data bonanza. http://www.bbc.co.uk/news/technology-22299503, April 2013.

[Int13] ECMA International. Ecmascript language specication. http://www.ecma-international.org/, 2013.

[Jal13] Amir Jalilifard. Real Time Commenting using SignalR and XSLT. http://www.codeproject.com/Articles/548549/Real-Time-Commenting-using-SignalR-and-XSLT, April 2013.

[Jav10] Java.net - The Source for Java Technology Collaboration. HTML5 Server-Push Technologies, Part 2. https://today.java.net/article/2010/04/26/html5-server-push-technologies-part-2, April 2010.

[Job10] Steven Paul Jobs. Thoughts on Flash. http://www.apple.com/hotnews/thoughts-on-flash/, 2010.

[Jon13] M. Tim Jones. Boost application performance using asynchronous I/O. http://www.ibm.com/developerworks/linux/library/l-async/?ca=dgr-lnxw02aUsingPOISIXAIOAPI, 2013.

[Joy13] Joyent, Inc. Node.js. http://nodejs.org/, 2013.

[KAH08] Krishna M. Kavi, Robert Akl, and Ali R. Hurson. Real-time systems: An introduction and the state-of-the-art. In *Wiley Encyclopedia of Computer Science and Engineering*, 2008.

[Keg11] Dan Kegel. The C10K problem. http://www.kegel.com/c10k.html, June 2011.

[Khr] Khronos Group. WebGL specification. http://www.khronos.org/webgl/.

[Khr13a] Khronos Group. Internet home of OpenGL ES. http://www.khronos.org/opengles/, 2013.

[Khr13b] Khronos Group. Internet home of WebGL. http://www.khronos.org/webgl/, 2013.

[Koz08] Frank Kozamernik. Evolution of the BBC iPlayer. http://tech.ebu.ch/docs/techreview/trev_2008-Q4_iPlayer.pdf, 2008.

[KVV08] M. Karlesky and M. Vander Voord. Agile Project Management. Or, Burning Your Gantt Charts. In *Proc. Embedded Systems Conference Boston, Boston, Massachusetts*, pages 247–267, October 2008.

[LA90] Shem-Tov Levi and Ashok K. Agrawala. *Real-time system design*. McGraw-Hill Computer Science series. McGraw-Hill, 1990.

[Lap97] Phillip A. Laplante. *Real-time systems design and analysis - an engineer's handbook (2. ed.)*. IEEE, 1997.

[LC11] Mark Lochrie and Paul Coulton. Mobile phones as second screen for tv, enabling inter-audience interaction. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ACE '11, pages 73:1–73:2, New York, NY, USA, 2011. ACM.

[LC12] Mark Lochrie and Paul Coulton. Sharing the viewing experience through second screens. In *Proceedings of the 10th European conference on Interactive tv and video*, EuroiTV '12, pages 199–202, New York, NY, USA, 2012. ACM.

[LD08] Dong Liu and Ralph Deters. The Reverse C10K Problem for Server-Side Mashups. In George Feuerlicht and Winfried Lamersdorf, editors, *Service-Oriented Computing - ICSOC 2008 Workshops, ICSOC 2008 International Workshops, Sydney, Australia, December 1st, 2008, Revised Selected Papers*, volume 5472 of *Lecture Notes in Computer Science*, pages 166–177. Springer, 2008.

[LG] Peter Lubbers and Frank Greco. HTML5 Web Sockets: *A Quantum Leap in Scalability for the Web*, howpublished = "`http://www.websocket.org/quantum.html`", lastchecked = 27.04.2013, author = Kaazing Corporation.

[Lub10] Peter Lubbers. How HTML5 Web Sockets Interact with Proxy servers. `http://www.infoq.com/articles/Web-Sockets-Proxy-Servers`, March 2010.

[Mat09] Markus Mathes. *Time-constrained web services for industrial automation*. PhD thesis, 2009.

[McM13] Simon McManus. Node.js twitter-oauth. `https://npmjs.org/package/twitter-oauth`, 2013.

[Mer06] Duane Merrill. Mashups: The new breed of web app, 2006. `http://www.ibm.com/developerworks/xml/library/x-mashups.html`.

[Mic06] B.M. Michelson. Event-driven architecture overview. *Patricia Seybold Group, Feb*, 2006.

[MSLO10] Pier Luigi Mazzeo, Paolo Spagnolo, Marco Leo, and Tiziana D Orazio. Football players classification in a multi-camera environment. In Jacques Blanc-Talon, Don Bone, Wilfried Philips, Dan C. Popescu, and Paul Scheunders, editors, *Advanced Concepts for Intelligent Vision Systems - 12th International Conference, ACIVS 2010, Sydney, Australia, December 13-16, 2010, Proceedings, Part II*, volume 6475 of *Lecture Notes in Computer Science*, pages 143–154. Springer, 2010.

[Mut05] S. Muthukrishnan. Data streams: algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, August 2005.

[Ngi13a] Nginx, Inc. nginx. `http://nginx.org/`, 2013.

[Ngi13b] Nginx, Inc. nginx - Company profile. `http://nginx.com/company.html`, 2013.

[Nod13a] Node.js. Cluster node.js. `http://nodejs.org/api/cluster.html`, 2013.

[Nod13b] Node.js; Author: "scothis@gmail.com". Node.js rest. `https://npmjs.org/package/rest`, 2013.

[OAu13] OAuth community. OAuth community page. `http://oauth.net/`, 2013.

[OBB] Ed Ort, Sean Brydon, and Mark Basler. Mashup Styles, Part 1: Server-Side Mashups.

[OFB04] Dan Olteanu, Tim Furche, and François Bry. An efficient single-pass query evaluator for xml data streams. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 627–631, New York, NY, USA, 2004. ACM.

[Opt13a] Opta Sportsdata Ltd. Internet platform of Opta Sports Data. `http://optasportsdata.com/`, 2013.

[Opt13b] Opta Sportsdata Ltd. Opta Pro Sports Data Modelling. `http://www.optasportspro.com/media/11966/final_optapro_sports_data_modelling.pdf`, 2013.

[Rau12] Guillermo Rauch. Socket.io. `http://socket.io/`, 2012.

[RR07] Leonard Richardson and Sam Ruby. *Restful web services*. O'Reilly, first edition, 2007.

[RS04] IBM Redbooks and C. Sadtler. *Patterns: Broker Interactions for Intra- and Inter-Enterprise*. IBM Redbooks. IBM Corporation, International Technical Support author, 2004.

[SAM11] Joonas Salo, Timo Aaltonen, and Tommi Mikkonen. MashReduce - Server-Side Mashups for Mobile Devices. In *GPC*, pages 168–177, 2011.

[Sha01a] Alan C. Shaw. *Real-time systems and software*, chapter Introduction: The World of Real-Time Systems, pages 1–14. In [Sha01b], 2001.

[Sha01b] Alan C. Shaw. *Real-time systems and software*. Wiley, 2001.

[Sim13] Kyle Simpson. JSON-P. `http://www.json-p.org/`, 2013.

[SJS12] D.I.K. Sjberg, A. Johnsen, and J. Solberg. Quantifying the Effect of Using Kanban versus Scrum: A Case Study. *Software, IEEE*, 29(5):47–53, 2012.

[SMNT10] Arto Salminen, Tommi Mikkonen, Feetu Nyrhinen, and Antero Taivalsaari. Developing client-side mashups: experiences, guidelines and the road ahead. In *Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '10, pages 161–168, New York, NY, USA, 2010. ACM.

[TC77] David Tebbs and Garfield Collins. *Real time systems : management and design*. McGraw-Hill, London, New York, 1977. Includes index.

[Twi13a] Twitter, Inc. Twitter. `https://twitter.com/`, 2013.

[Twi13b] Twitter, Inc. Twitter FAQ - What does the retirement of API v1 entail? `https://dev.twitter.com/docs/faq#17750`, 2013.

[U. 81] U. I. S. Institute. RFC 793 - Transmission Control Protocol. `http://www.faqs.org/rfcs/rfc793.html`, September 1981.

[W3C99] W3C. Hypertext Transfer Protocol – HTTP/1.1. `http://www.w3.org/Protocols/rfc2616/rfc2616.html`, 1999.

[W3C04a] W3C. Web Services Architecture. `http://www.w3.org/TR/ws-arch/`, February 2004.

[W3C04b] W3C. Web Services Glossary. `http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice`, February 2004.

[W3C08] W3C. Extensible Markup Language (XML) 1.0. `http://www.w3.org/TR/xml/`, 2008.

[W3C12] W3C. HTML 5 - A vocabulary and associated APIs for HTML and XHTML. `http://www.w3.org/TR/html5/`, December 2012.

[W3C13a] W3C. Same-Origin Policy. http://www.w3.org/Security/wiki/Same_Origin_Policy, 2013.

[W3C13b] W3C Schools. Html iframe tag. http://www.w3schools.com/tags/tag_iframe.asp, 2013.

[W3T13] Web Technology Surveys W3Techs. Usage of web servers broken down by ranking. http://w3techs.com/technologies/cross/web_server/ranking/, 2013.

[XDZ+07] Yang Xiao, Xiaojiang Du, Jingyuan Zhang, Fei Hu, and S. Guizani. Internet protocol television (iptv): The killer application for the next-generation internet. *Communications Magazine, IEEE*, 45(11):126–134, 2007.

[ZC04] Di Zhong and Shih-Fu Chang. Real-time view recognition and event detection for sports video. *J. Vis. Comun. Image Represent.*, 15(3):330–347, September 2004.

Appendix A

Opta data file extract

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Games timestamp="2011-08-18T10:13:01">
3    <Game id="360474" away_team_id="30" away_team_name="Bolton Wanderers"
         competition_id="8" competition_name="English Barclays Premier League"
         game_date="2011-08-13T15:00:00" home_team_id="52" home_team_name="Queens
         Park Rangers" matchday="1" period_1_start="2011-08-13T15:00:42"
         period_2_start="2011-08-13T16:08:56" season_id="2011" season_name="Season
         2011/2012">
4      <Event id="2127961007" event_id="1" type_id="34" period_id="16" min="0" sec="
           0" team_id="30" outcome="1" x="0.0" y="0.0" timestamp="2011-08-13T14
           :13:04.250" last_modified="2011-08-13T14:13:04">
5        <Q id="911629043" qualifier_id="30" value="1344, 28183, 2004, 27696, 19419,
             1587, 18428, 14668, 9765, 3630, 10089, 45175, 82263, 1307, 19930,
            1615, 15188, 19958" />
6        <Q id="1716712837" qualifier_id="227" value="0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0" />
7        <Q id="1791649219" qualifier_id="131" value="1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
             11, 0, 0, 0, 0, 0, 0, 0" />
8        <Q id="1299836508" qualifier_id="194" value="3630" />
9        <Q id="226775371" qualifier_id="59" value="22, 2, 4, 6, 5, 12, 7, 19, 17,
            14, 10, 1, 3, 11, 16, 20, 21, 31" />
10       <Q id="432006640" qualifier_id="130" value="2" />
11       <Q id="1461167565" qualifier_id="44" value="1, 2, 2, 3, 2, 2, 3, 3, 4, 4,
            3, 5, 5, 5, 5, 5, 5, 5" />
12     </Event>
13     <Event id="1713063904" event_id="1" type_id="34" period_id="16" min="0" sec="
           0" team_id="52" outcome="1" x="0.0" y="0.0" timestamp="2011-08-13T14
           :14:43.249" last_modified="2011-08-13T15:27:00">
14       <Q id="377505913" qualifier_id="227" value="0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0" />
15       <Q id="1814271195" qualifier_id="30" value="3119, 1840, 1216, 67731, 12674,
             1059, 4255, 1420, 19645, 39765, 2019, 10589, 12443, 18586, 27698,
            5730, 3731, 19946" />
16       <Q id="1650801177" qualifier_id="59" value="1, 8, 3, 11, 5, 6, 10, 4, 9, 7,
             21, 26, 2, 14, 16, 19, 22, 25" />
```

```
17      <Q id="1535569483" qualifier_id="44" value="1, 3, 2, 3, 2, 2, 4, 3, 4, 3,
            3, 5, 5, 5, 5, 5, 5, 5" />
18      <Q id="317904683" qualifier_id="131" value="1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
            11, 0, 0, 0, 0, 0, 0, 0" />
19      <Q id="776633407" qualifier_id="130" value="8" />
20      <Q id="1088775936" qualifier_id="194" value="39765" />
21    </Event>
22    <Event id="744757990" event_id="3" type_id="1" period_id="1" min="0" sec="1"
          player_id="3630" team_id="30" outcome="1" x="50.0" y="47.4" timestamp="
          2011-08-13T15:00:44.338" last_modified="2011-08-13T15:01:13">
23      <Q id="1950938333" qualifier_id="212" value="2.6" />
24      <Q id="1634459306" qualifier_id="56" value="Center" />
25      <Q id="1761914078" qualifier_id="213" value="1.2" />
26      <Q id="1333259305" qualifier_id="140" value="50.9" />
27      <Q id="68456476" qualifier_id="141" value="50.9" />
28    </Event>
29       ...
30       ...
31       ...
32  </Game>
33 </Games>
```

Appendix B

Data analysis - distribution tables

Frequency	Percentage	Event type
853	50.99 %	pass
145	8.67 %	out
108	6.45 %	ball recovery
84	5.02 %	clearance
64	3.83 %	aerial duel
50	2.99 %	foul
48	2.87 %	bad ball touch
39	2.33 %	tackle
36	2.15 %	deleted event
28	1.67 %	dispossession
26	1.55 %	interception
25	1.49 %	take on (dribble)
22	1.31 %	goalkeeper pickup
20	1.19 %	delay start or end
19	1.14 %	attempt saved
18	1.08 %	goalkeeper save
14	0.84 %	challenge
13	0.78 %	missed shot
10	0.6 %	corner
51	3.05 %	others (substitutions, cards, goals, good skills, . . .)
1673	100 %	total

Table B.1: Event Frequencies - `game1` (2011)

Frequency	Percentage	Event type
880	54.73 %	pass
138	8.58 %	out
86	5.35 %	aerial duel
85	5.29 %	ball recovery
71	4.41 %	clearance
42	2.61 %	bad ball touch
38	2.36 %	foul
27	1.68 %	tackle
23	1.43 %	take on (dribble)
20	1.24 %	interception
19	1.18 %	goalkeeper save
19	1.18 %	goalkeeper pickup
19	1.18 %	attempt saved
18	1.12 %	deleted event
18	1.12 %	dispossession
16	0.96 %	delay start or end
15	0.93 %	missed shot
12	0.75 %	corner
11	0.68 %	challenge
51	3.17 %	others (substitutions, cards, goals, good skills, . . .)
1608	100 %	total

Table B.2: Event Frequencies - `game2` (2012)

Team ID	Player ID	Number of events
30	***total***	***891***
	3630	92
	9765	63
	1581	61
	27696	53
	2004	70
	19419	100
	1344	61
	10089	84
	14668	63
	18428	84
	28183	92
	19930	24
	15188	13
	1615	14
	w.o. player id	17
average	*mean*	62.43
	median	*63*
	mode	*61, 63, 84, 92* (freq: *2*)
52	***total***	***782***
	1216	64
	2019	48
	39765	59
	67731	78
	1059	79
	4255	104
	1420	59
	3119	77
	1840	4
	19645	40
	12443	47
	12674	57
	18586	26
	3731	23
	w.o. player id	17
average	*mean*	54.64
	median	*58*
	mode	*59* (freq: *2*)
overall	***total***	***1673***
	w.o. player id	34
average	*mean*	58.54
	median	*61*
	mode	*59, 61, 63, 84, 92* (freq: *2*)

Table B.3: Events/player - `game1` (2011)

Team ID	Player ID	Number of events
21	***total***	***710***
	40142	76
	5716	34
	48717	70
	18073	73
	19575	67
	28147	57
	49413	88
	1344	52
	5306	36
	5609	54
	42954	21
	18818	23
	3289	31
	12799	12
	w.o. player id	16
average	*mean*	49.52
	median	*53*
	mode	*undefined* (all have freq: *1*)
6	***total***	***898***
	7958	41
	58621	102
	68815	84
	39194	87
	12679	78
	36903	89
	20226	80
	17349	47
	15109	93
	52876	73
	37915	70
	39104	18
	49944	19
	55422	2
	w.o. player id	15
average	*mean*	63.07
	median	*57.5*
	mode	*undefined* (all have freq: *1*)
overall	***total***	***1608***
	w.o. player id	31
average	*mean*	56.32
	median	*62*
	mode	*70, 73* (freq: *2*)

Table B.4: Events/player - `game2` (2012)

Team ID	Player ID	No. of events	Mean time	Max time	Median time
30	27696	53	1:46	10:17	0:40
	2004	70	1:19	7:48	0:34
	18428	84	0:55	6:23	0:15
	14668	63	1:17	7:12	0:33
	1587	61	1:31	9:42	0:39
	15188	13	0:50	2:19	0:25
	10089	84	1:05	11:38	0:26
	1615	14	0:33	1:20	0:21
	9765	63	1:22	8:34	0:50
	19930	24	0:33	2:13	0:20
	3630	92	1:00	5:03	0:45
	19419	100	0:56	4:44	0:27
	1344	61	1:32	9:11	0:35
	28183	92	0:59	5:04	0:24
52	1420	59	1:33	9:31	0:47
	2019	48	1:31	8:30	0:27
	3731	23	0:53	2:12	0:46
	12674	57	1:31	7:27	1:09
	3119	77	1:10	6:17	0:56
	67731	78	1:11	4:57	0:42
	4255	104	0:52	11:56	0:20
	1216	64	1:28	9:08	0:44
	19645	40	2:16	9:45	1:23
	39765	59	1:14	6:22	0:41
	1059	79	1:10	11:57	0:22
	1840	4	1:23	3:36	0:27
	12443	47	1:49	7:06	0:49
	18586	26	0:49	6:05	0:20

Table B.5: Time spans between player events - `game1` (2011)

Team ID	Player ID	No. of events	Mean time	Max time	Median time
21	19575	67	1:24	11:29	0:30
	5609	54	1:39	10:07	0:24
	12799	12	2:16	10:16	0:56
	48717	70	1:22	8:33	0:35
	40142	76	1:16	7:25	0:41
	5716	34	1:38	6:30	1:12
	3289	31	1:20	8:53	0:36
	18073	73	1:13	10:00	0:37
	42954	21	1:25	7:33	0:36
	49413	88	1:03	7:24	0:26
	5306	36	1:52	6:34	1:08
	18818	23	2:45	21:14	1:18
	1344	52	1:49	10:13	1:11
	28147	57	1:34	7:16	0:36
6	12679	78	1:13	10:5	0:39
	17349	47	1:41	9:39	0:51
	15109	93	1:01	5:06	0:32
	49944	19	0:30	2:16	0:07
	58621	102	0:55	6:31	0:24
	36903	89	1:04	6:21	0:40
	39194	87	1:05	4:43	0:37
	7958	41	2:20	17:9	1:04
	68815	84	1:08	6:49	0:41
	37915	70	1:18	5:43	0:39
	55422	2	0:45	0:45	0:45
	20226	80	1:11	5:23	0:39
	39104	18	0:55	4:01	0:19
	52876	73	1:10	6:41	0:43

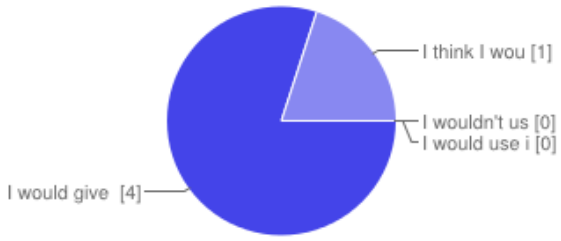
Table B.6: Time spans between player events - `game2` (2012)

Appendix C

User evaluation
Summary of questionnaires

Summary

I would use this app...

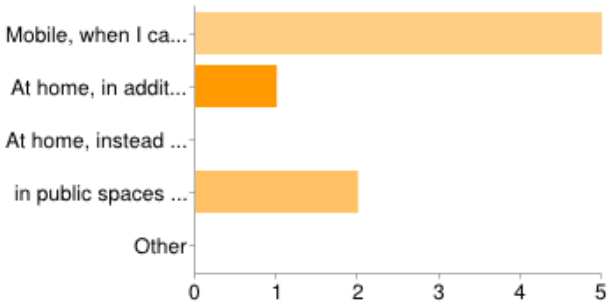


I wouldn't use it	**0**	0%
I would give it a try	**4**	80%
I think I would use it on a regular basis	**1**	20%
I would use it as often as I can	**0**	0%

Why would / wouldn't you use it?

Allows to decrypt the behaviour of players and their global performance during a game. Can also analyse the action that lead to a goal. 1) Seems interesting visualization of a football game. 2) Access to statistics that usually displayed only during half time break or after the game ends. It is quite interesting to be access those statistics any time. 3) I wouldn't use it regularly because the live broadcast shows only a small part of the action (only around the ball) I think it is an interesting way to see a play by play visualization of the game, and to keep up with all the latest action. If the internet connection is too slow to broadcast real-time streaming and/or on mobile phones in order not to miss the match details. I think the application is informative in providing the flow of the game. The interactive node bring the game more lively

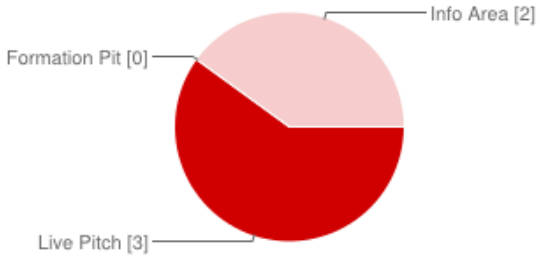
When/How would you use this app?



Mobile, when I cannot watch the TV broadcast of a game	**5**	63%
At home, in addition to watching the TV broadcast	**1**	13%
At home, instead of watching the TV broadcast	**0**	0%
in public spaces (Stadium, Restaurant, Pub, ...) (Please use 'other' to explain why)	**2**	25%

Other	**0**	0%

Which of the three parts was most appealing to you?



Live Pitch	**3**	60%
Formation Pitch	**0**	0%
Info Area	**2**	40%

Please provide a reason for your choice.

The effect of live match is maintained. It is nice to see players moving on the pitch without using a lot of resources as in the case of real-time streaming. The live pitch has more important flow and focuses on whereabout of the ball location. Other information required additional clicks or steps to reach them. Access to statistics that usually displayed only during half time break or after the game ends. It is quite interesting to be access those statistics any time. Displaying the game action, is what I would primarily be using the application for. Yields live data about the performance of the players during a game. Very useful to understand the behaviour of an individual player.

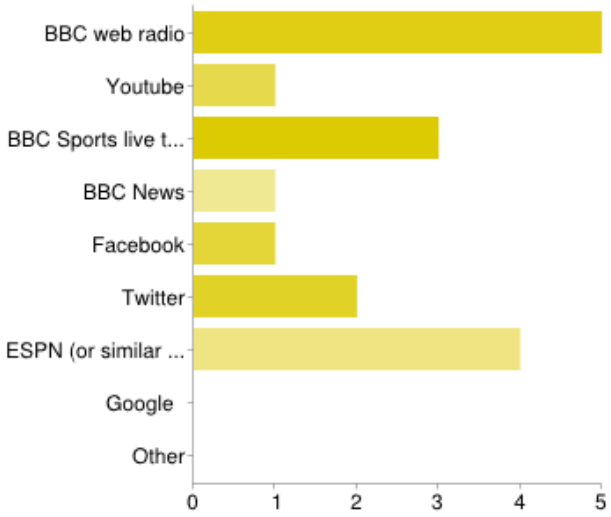
Understanding of Visualisations

Yes, although the visualizations should possibly be split into two tabs for each half of the game. Partially, the arrow have good indication of direction and type of event, yet it is easily lose track as the arrow fade in fade out too fast, it would great to have list of event stack to assist these information. Yes I partially understood the visualisations. Even for the whole game, where there are more data to be presented, the statistics were quite clear. Some times for some players it was a bit complicated (too many players / lines) Yes. Very clear and visual. Maybe black isn't the appropriate colour for an an accurate pass. Yes, everything is clear. I could clearly see the parts of the pitch with most actions happened.

Do you think integration of Social Media (e.g. Twitter) is a good idea? Please explain why you think so.

Yes, gives access to reactions of viewers to the overall performance of the team, but also of individual players. It can also help the follower (on the app) understand the game. Yes, that is definitely good idea as it share the thoughts of other about the games and possibility their analysis and view how about the game eventually lead. It encourage more discussion and bring the game lively. Indeed is a good idea. It would be interesting to see comments and live discussions about the game and specific plays / players etc. That is what we do when watch a real game! I have no idea about this :) I think it is a good idea, allowing you to see live fan reactions to the game you are currently watching.

Do you think data of these data providers should be integrated?



BBC web radio	**5**	29%
Youtube	**1**	6%
BBC Sports live ticker	**3**	18%
BBC News	**1**	6%
Facebook	**1**	6%
Twitter	**2**	12%
ESPN (or similar sports platforms)	**4**	24%
Google +	**0**	0%
Other	**0**	0%

Please list parts of the application that you DID NOT like. Please give a short explanation why you think so.

The way that sometimes the movement of the ball seemed to jump from the its last location without an arrow indicating how it got there. Overall, this application is good. But it needs more support and optimized use of resources. The switching symbol between pitches - makes the user think it is a refresh button. No time-out for reconnection - not implemented yet? The live broadcast need more data to be realistic. Live Pitch, need more literal information

Please list parts of the application that you DID like, including a short explanation

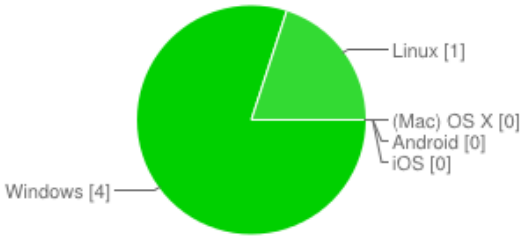
The live play and the breakdown of each players actions up to that point during the game. As I said, the application prototype is quite close to ideal. The heat map, very clear. The potential of having live twitter newsfeed. The data visualisation page where the the statistics are presented/ Live Pitch, an interactive and visual representation of the game Formation Pitch, an overview of the players from both team Twitter feed, sharing thoughts with other game watchers.

Please list features that you would like to see in a final version of the application?

All real actions that are happening on the pitch. Would be better to see exact emulation of real

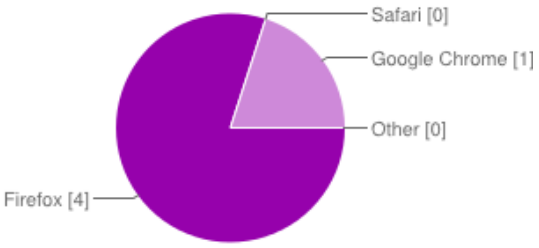
game. Introduce substitutes and their statistics. Also have written statistics for passes accuracy. I would like see a list of text represent the event history, to assist my understanding of the flow It would be nice to have the audio feed of the game integrated into the application synced up to the action. I would have appreciated a 'legend' indicating what the colours meant for the passes etc. This was explained to us verbally, but would be nice to see. Other players positions to have a realistic game simulation. The ball position. More statistics presented with the traditional way (numbers) A chat about the game with other users watching the application

I used



Windows	**4**	80%
Linux	**1**	20%
(Mac) OS X	**0**	0%
Android	**0**	0%
iOS	**0**	0%

I used ... (browser)



Firefox	**4**	80%
Safari	**0**	0%
Google Chrome	**1**	20%
Other	**0**	0%

Appendix D

Software Eng. Process & Project timeline

The Kanban philosophy fit well with this project and simplified the time management. Since certain time constraints such as lectures, exams and write up phases occasionally slowed down or even paused the project, the sprints needed to be adapted in length to counteract these interruptions.

D.1 Project timeline

The project spanned a period of slightly more than seven months, from February until the beginning of September. The work capacity varied during the project period, as in the first half (from February until end of May) there were lectures, coursework and exams that required attention, work and time resources. The second half (June until August) was fully dedicated to work on the project.

The project was divided into five major phases, each having a milestone its end:

- Kick off & Evaluation
- Implementation 1 (core functionality)
- Implementation 2 (features)
- Optimisation, Enhancement & Clean up
- End phase

Each of these phases had a general theme that it was following. However, due to the nature of Kanban and the plan of having an agile project, the exact implementations during each phase varied. This was approved upfront by all project stakeholders.

The deadline of the project was September 6^{th}. This was a hard deadline, which

could not be extended. The project was ended prior to this deadline on August 15^{th}. This lead to a *buffer* period of three weeks, that was used for final bug fixes, the write up of the dissertation, organisational tasks such as printing of the dissertation, etc. or any other task that was not finished before.

D.2 Adaption of Kanban

Kanban relies on quick and constant reviews of the process. Although the process itself does not require sprints or retrospectives, there is the need for analysis of the employed development system. For this reason, the project was carried out in monthly *review cycles*. After each of these review cycles, a retrospective summarised the progress of the project and evaluated the effectiveness of current Kanban methodology.

To facilitate the retrospectives a sprint system was introduced. The idea of these sprints was based on the concept of short implementation cycles and the availability of working software at the end of the sprint. Although the exact sprint length was flexible, a two weeks guideline was used.

In contrast to Scrum sprints, the sprints in this project were flexible and required less documentation. At the beginning of a sprint the work packages with the highest priority were selected and their complexity estimated. Following the lean management philosophy the tasks in the sprint backlog could be exchanged, removed, extended and modified to fit the changed project requirements.

Depending on the sprint's time frame, the sprint length could be adapted to be longer or shorter. An example for this adaptation is the university exam period where exam revision required attention. An exam was at the end of the month, and it was better to have one three-week long sprint and a week of exam preparation without any project work, rather than two two-week long sprints, where the second one would have only had minor progress.

D.2.1 Gantt chart

It is very unusual to follow Gantt charts in agile projects, as it is in general not possible to combine the idea of detailed planning and lean project management [KVV08]. Nevertheless most projects require the creation of Gantt charts to visualise milestones, review cycles and the project's timeline to the management. In this project, the 'management' consists of the BBC supervisors, the academic supervisor and the readers of

the initial report, the progress report and the dissertation.

Figure D.1 shows the Gantt chart that has been agreed upon at the project start.

The project involved five milestones. The plan required to have the following tasks accomplished at the given dates respectively:

Deadline Initial Report (March 7)

- finish initial analysis of the data
- perform initial literature research
- agree on a project scope with the stakeholders

Data Representation (May 10)

- perform most of the background research
- finish first data parser implementation
- first implementation of the user interface
- write progress report

Core Application (July 8)

- finish main functionality of the application
- optimise the performance
- carry out evaluation and performance tests

Planned Project End (August 15)

- implementation and testing stop
- dissertation version 1 finished
- start of 3 week *buffer period*

Deadline Dissertation (September 6)

- hand-in of the dissertation (two copies; printed & bound)
- final version of application ready and all bugs fixed

D.3 Conclusions

The flexibility provided by the Kanban project management was a key factor for the success of the project. The ability to prioritise tasks at any point facilitates development, especially for a prototype project like this one. The reason for the need to rearrange work packages was due to the fact that new information and findings changed the envisaged project work-flow. Regular meetings at the BBC (in average every two weeks) resulted in a close relationship with the *customer* (the BBC). The informal process resulted in only little documentation overhead and well informed project stakeholders.

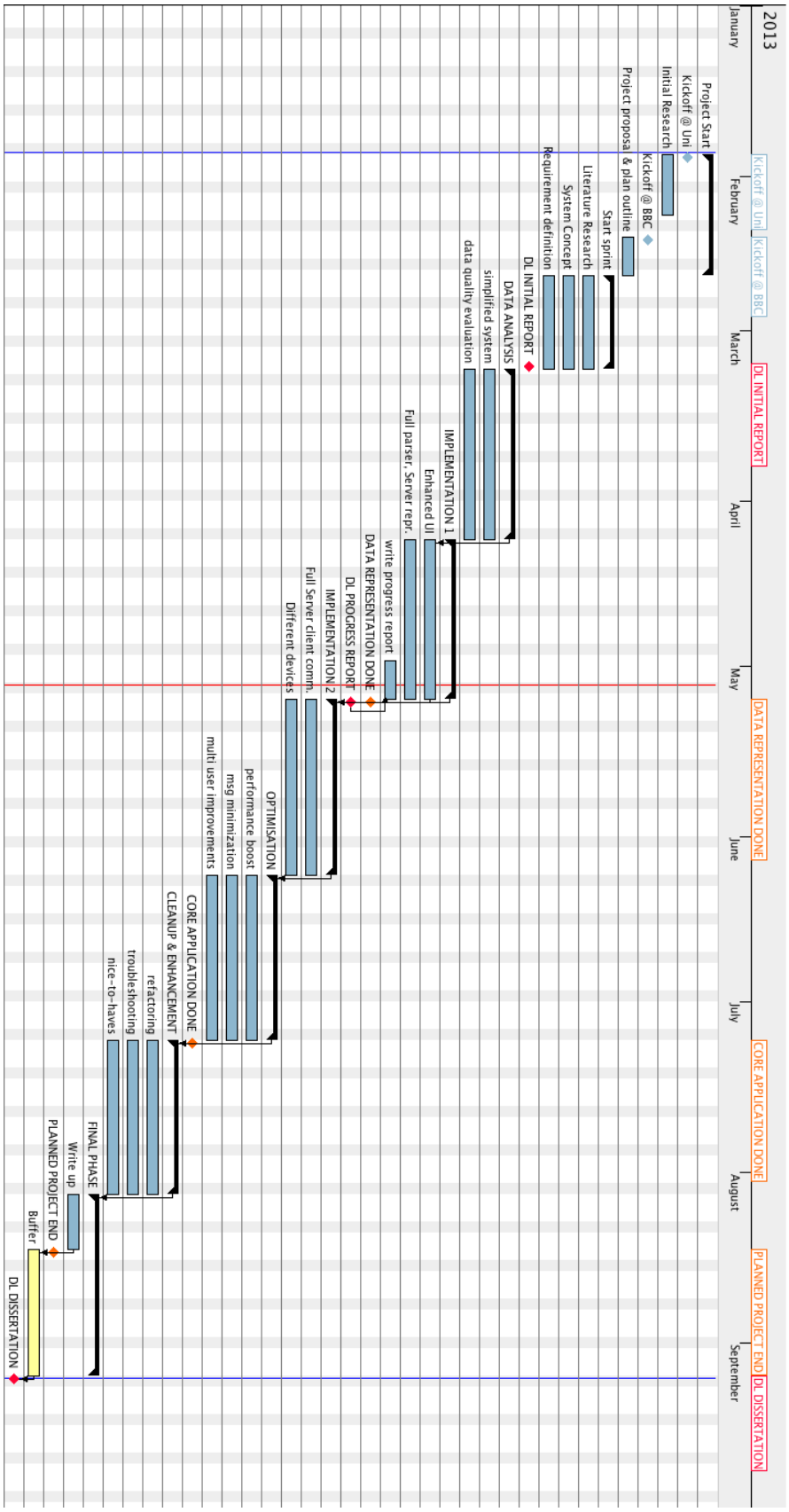


Figure D.1: Gantt chart showing an estimate of the project timeline