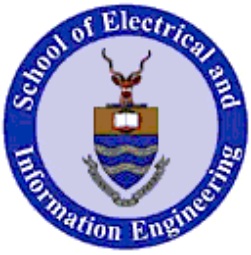

School of Electrical and Information Engineering

University of the Witwatersrand, Johannesburg

ELEN 4010 Software Development 3 Project 2010

Dribble: The Design, Implementation and Analysis of a Geosocial Networking System: Server Aspect

May 2010

Andrew Paverd

0702663D

Group Members:

Ashleigh Campbell	0605245J
Chad Epstein	0706965X
Greg Favish	0713568E
Daniel Mankowitz	0616159H

http://github.com/ajpaverd/Dribble

DRIBBLE: THE DESIGN, IMPLEMENTATION AND ANALYSIS OF A GEOSOCIAL NETWORKING SYSTEM: SERVER ASPECT

Andrew Paverd

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: Geosocial networking is a new type of application in which the user's geographical position is used to provide relevant information. Dribble is an open-source geosocial networking system using a component-based client-server architecture. Scalability is the primary requirement of the server-side components. Dribble uses the JavaEE framework and runs on a Glassfish application server. The web communication tier uses webservices to interface with clients and JMS to interact with other components. The business processing tier consists of Message Driven EJBs which perform various processing functions. The system was tested and analysed and found to meet the specified requirements.

Key words: Dribble, ELEN4010, geosocial networking, server, software development

1 INTRODUCTION

Dribble is a geosocial networking system which enables users to post and receive messages relevant to their current geographical locations. The primary user interface to this system is a mobile phone application developed for the Android platform. This application communicates with a central server in order to upload and retrieve content. Since this system was developed as a team effort, different members focussed on different sections of the work. The core focus areas of this report are the functionality and features of the communication and processing components of the server-side application. The role of these components is to facilitate communication between the client and the server and to perform processing on the data to convert it into relevant information.

Section 2 provides background information about the system and the context in which it is used. Section 3 defines the scope of the project through the identification of requirements, constraints, assumptions and success criteria. Section 4 is a high-level overview of the complete system and Section 5 provides further details of the philosophy and implementation of the server-side application. Section 6 and Section 7 explain the functionality and features of the application's Web Communications Tier and the Business Processing Tier respectively. Section 8 shows how the Data Storage Tier is used by the server application. Section 9 explains how the system was tested and Section 10 describes the project management aspects associated with the development. Section 11 is a critical analysis of the system and recommendations for future work are presented in Section 12.

2 BACKGROUND

The first social networking site was launched in 1997 following the advent of the internet [1]. Since then, services of this type have greatly increased in popularity. However, the driving concept behind many of the current social networks is the idea of f̈riendsör contacts with which a user is linked. These links form the channels through which information is shared.

As mobile devices become more sophisticated, a new type of application has emerged called geosocial networking. In applications of this type, location based information and services are integrated with the traditional model of social interaction. Pioneering applications in this field include Brightkite and Google Latitude [2]. At present, the most popular application of this type is Foursquare [3]. This application uses the Facebook and Twitter structures for social networking and adds an element of location-based game-play.

Dribble would be classified as a geosocial network because its core focus is to gather relevant information from users and present this to other users based on geographical location. Unlike similar applications, Dribble is an open-source project.

3 PROJECT SCOPE

The requirements, constraints, assumptions and success criteria define the scope of this aspect of the project. The components described in this report are critical parts in the overall system and so they share and implement parts of the scope of the overall system as described in the group report [4].

3.1 Requirements

Apart from the requirements of the overall system, the server application and its components in particular are required to be highly scalable. They must be able to accommodate a variable load depending on the number of users of the system. These components must also have a high level of fault tolerance so that the server can continue operation even in the event of a failure of a component.

As with the overall system, these components are required to conform to modern standards and best practices so as to allow for continuation and extension of the project.

3.2 Constraints

Although the server components can utilise the full computational power of the server, they are still constrained by the limitations of this computational resource especially when the application increases in scale. Also, since the client-server communication takes place over a mobile network, the quantity of data transferred must be minimised.

Since these components form part of an open-source project, one of the constraints is that open-source technologies should be used where possible.

3.3 Assumptions

The first assumption made is that the infrastructure to communicate between the remote client and the central server is available. In this case, the GSM and 3G mobile networks are assumed to be present wherever the client application is used.

The second assumption is that the application will reach a certain critical mass of users. Since this system provides value using crowd-sourcing principles, the number of users directly affects the ability of the application to deliver its intended functionality.

3.4 Success Criteria

The success criteria for the project and for this section are to have implemented at least 90% of the specified functionality at a prototype level and to have tested at least 90% of the implemented functionality. An auxiliary success criterion is the continuation of development on the project in the future.

4 SYSTEM OVERVIEW

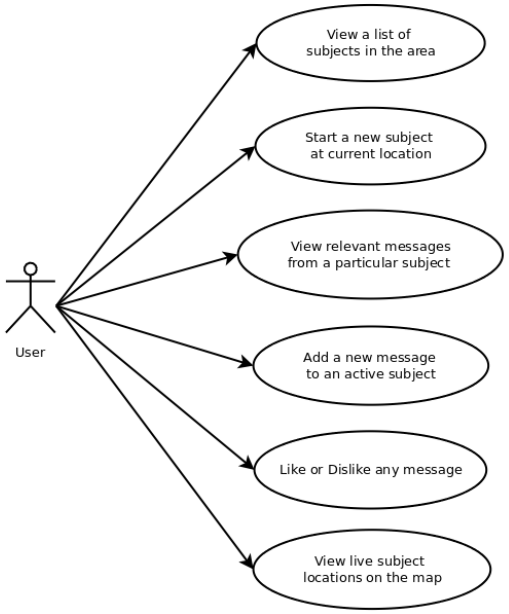

Figure 1 : Use Case Diagram of the system

The main use cases of the system are shown in the UML Use Case Diagram in Figure 1. This diagram shows how users can upload new content or view existing content relevant to their current locations. Although not depicted in the diagram, the application actually relies on a number of users forming a linked geosocial network.

In order to model the application domain in software, two fundamental classes were defined throughout the system. A Drib is a representation of a single message and a DribSubject represents a subject. Each Drib contains a reference to

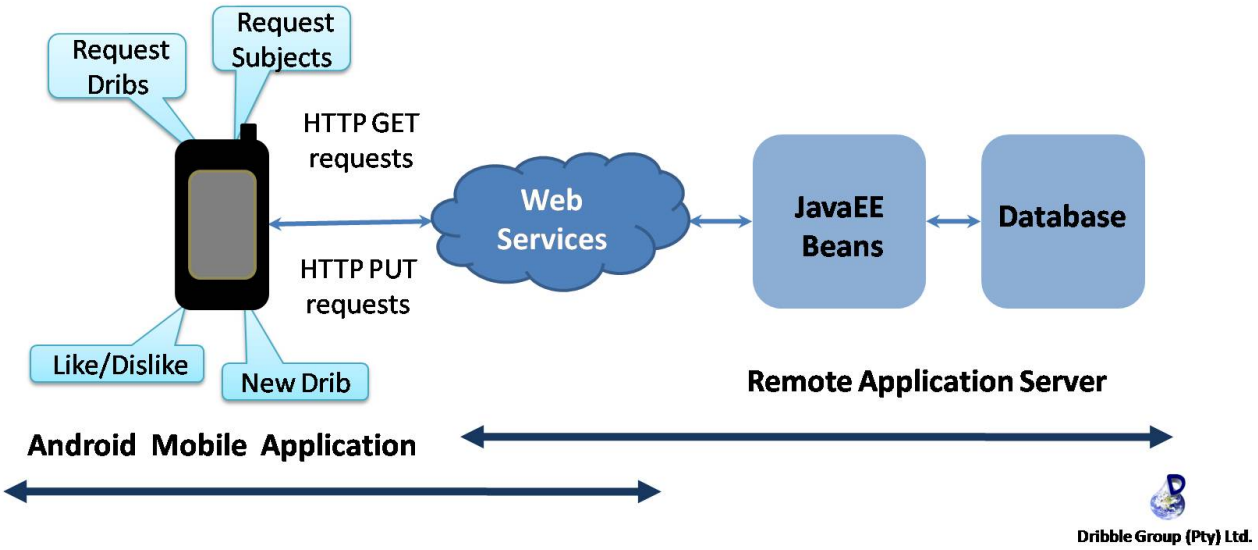

Figure 2 : Overview of the system

a DribSubject and hence multiple messages can have the same subject. These two classes also contain attributes pertinent to their content. By standardising these classes, communication between the different components of the application is greatly simplified.

As shown in Figure 2, system uses a client-server architecture and a request-response communications paradigm. Multiple clients communicate simultaneously with a central server or cluster of servers. Both the client and server components of the system are written in Java although different editions of the language are used for the different platforms.

The client-side of the system consists of a mobile device application running on the Android platform. At present, the Android platform is implemented by smart phones which have advanced communication and location-awareness capabilities. Although this is not the most prevalent platform by market share, it has been widely targeted for advanced applications including some of the similar solutions discussed in *Section 2*. The mobile application provides the user interface for the application. It allows users to view the list of geographically relevant subjects and the messages posted for each subject. It gives users the option to post a new message with a new subject or to reply to an existing subject. It also displays the geographic location of subjects on a map representation.

The server-side of the system consists of a Java Enterprise Edition (JavaEE) application running on an application server. This application provides a central point to which messages and subjects are uploaded and from which they can be requested. Since the bandwidth limitations of the communication network between the client and the server are a constraint of the system, only the minimum amount of data can be transferred. Therefore, most of the processing must take place on the server-side of the application. The server application is responsible for processing and storing the uploaded messages and servicing requests for lists of subjects or messages. Further details of the server-side application are given in the next section.

5 SERVER-SIDE APPLICATION

JavaEE (formally known as J2EE) is widely used for professional enterprise grade applications. The platform is stable and mature and a large support base. It adheres to open standards as required for this project. JavaEE is simply a different edition to that of the Android Java client application. Therefore, the matching of the languages used by the server and client applications improved the inter-operability between them.

One of the main advantages of using JavaEE for this application is that the framework lends itself to the use of a component based architecture. From a software engineering point of view, this practice of separating or decoupling different

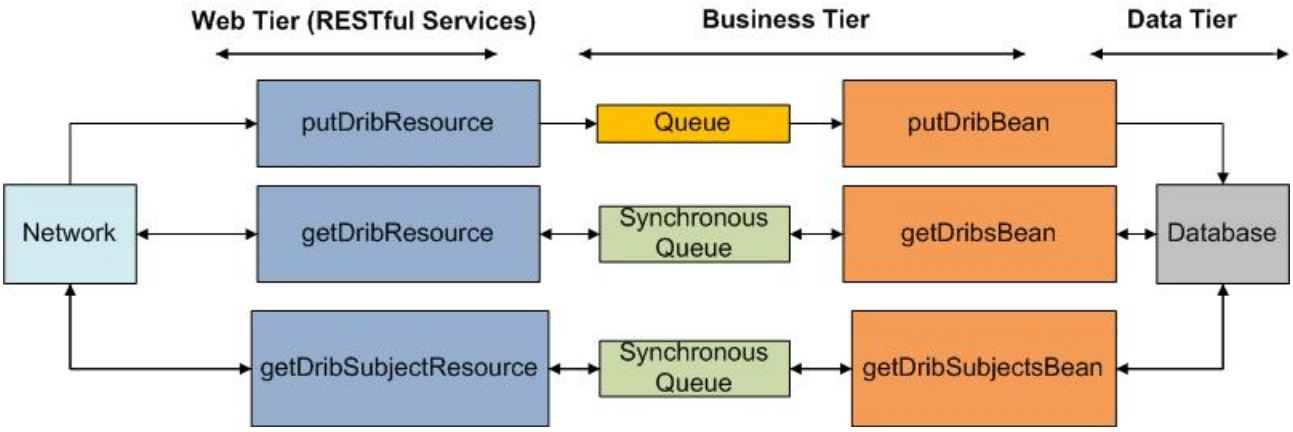

Figure 3 : Overview of the server-side application

functional areas improves the flexibility, reliability and maintainability of the application. Using a component based approach, functional elements can be tested in isolation, errors can be localised and changes can be made without affecting the rest of the system. As shown in Figure 3, the server application was split into three tiers of functionality, namely, the web communications tier, the business processing tier and the data storage tier. Within each tier, different components were created to fulfil various roles.

The Glassfish JavaEE application server was chosen to manage the server-side application. Glassfish is an open-source project which was formally sponsored by Sun Microsystems. This application server is standards based and supports all the functionality required for this application. Glassfish also provides a built-in Derby database implementation and interface which was used as the data storage system for this application.

The use of an application server requires a slight shift in development paradigm. Since the application server manages the lifecycle of the components, the developer does not have complete control of when a particular component will be instantiated or destroyed. The role of each component must be clearly defined so that the application server, depending on the requirements of the system, can create multiple instances of components on demand.

6 WEB COMMUNICATIONS TIER

The primary purpose of the components in the communications tier is to provide a link between the client and server applications. This is achieved through the use of webservices.

6.1 Webservices

The term, "webservices" is used to describe a number of technologies used for communication between applications over the internet. The advantage of using this form of communication is that it conforms to established standards. Simple Object Access Protocol (SOAP) is a common type of webservice in which objects are encapsulated within an XML structure and sent over the network. Although SOAP would fulfil the communication requirements for this application, its use of multi-layer XML encapsulation causes a large data overhead when sending a message. Since these messages will be sent over the constrained mobile networks, this is not ideal.

An alternative to SOAP is Representative State Transfer (RESTful) webservices. These are a much more lightweight class of webservices in which information is transferred using the standard HTTP protocol. RESTful webservices still use XML to represent the object but many of the additional layers found in SOAP are removed by using the standard HTTP message types, "PUT", "POST" and "GET".

These message types are ideal for use in this application. To provide communications functionality, three RESTful webservices form the web tier. Each webservice component has a unique path which is added to the end of the server's URL in order to access the service. The *PutDribResource* webservice accepts a new or modified message object from the client via a "PUT" request. The *GetDribSubjectsResource* and *Get-*

DribsResource webservices respond to "GET" requests by returning lists of subjects and messages to the client. In the "GET" requests, the client specifies its current latitude and longitude as parameters in the http message. When an HTTP message is received, the application server locates or creates an instance of the appropriate webservice resource. In this way, the system can handle multiple simultaneous HTTP messages and can therefore scale as the demand increases.

At a low level, webservices operate by transforming the object into an XML string by a process of serialisation. In order to send the messages and subjects, the *Drib* and *DribSubject* classes had to implement the *serializable* interface. In theory, this XML string can be sent over the network and de-serialised on the receiving end to recreate the object. However, in this project, it was found that the implementation of the webservices differed between the server and client. In order to overcome this, the XML representation of objects on the server-side was modified slightly through the use of JavaEE annotations. The transferring of lists of subjects and messages was also found to be problematic and so wrapper classes were created to encapsulate these lists. Using this method, only a single object is sent via the webservice and the de-serialisation procedure successfully reconstructs the object.

Once a message has been received by a webservice, it must be communicated to the business processing tier of the application. In this application, this communication was achieved by using the Java Messaging Service (JMS).

6.2 Java Messaging Service

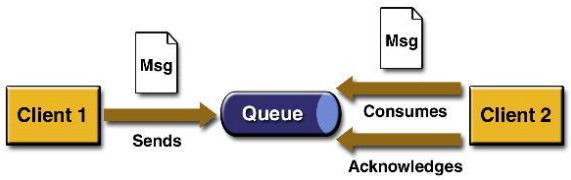

Figure 4 : Example of JMS Queue

Java Messaging Service (JMS) is a JavaEE framework for sending messages between different components within an application. A JMS Destination is a virtual address to which messages can be sent and from which they can be received. The application server manages all JMS destinations. A JMS Queue is a specific type of JMS Destination which enforces a one-to-one relationship as shown in Figure 4. When a message is placed in the queue by a sending object, the application server locates or creates an instance of a receiving object for that queue and passes the message to that object.

A JMS Message object can take on different forms depending on the type of information it contains. When a Drib object is received from the client, the *PutDribResource* creates a JMS ObjectMessage in which it stores the Drib. Ordinary JMS Messages are sent by the other two webservices since they only contain location information.

The disadvantage of this approach is that a JMS Message is required whenever communication takes place. The sending and receiving of these messages adds slightly more overhead to both the sending and receiving components. However, this structured approach becomes a significant advantage as the scale of the application increases. Since JMS is an asynchronous communication technology, the webservice does not need to wait for for the processing of the new Drib to be completed. Once the JMS message has been sent to the queue, the webservice can continue to respond to new incoming messages. The application server monitors the length of each queue and will create additional instances of receiving objects as required. In this way, load-balancing takes place within the application since there can be different numbers of sending and receiving objects depending on how much time each process requires. Using JMS technology, the server application can automatically scale to meet the demands of the clients.

7 BUSINESS PROCESSING TIER

The business processing tier receives instructions from the communications tier and performs the necessary processing operations. This tier also interacts with the data storage tier in order to store and retrieve information. The processing tier consists of three main components, namely, the *PutDribBean*, the *GetDribSubjectsBean* and the *GetdribsBean*. These three components are Enterprise JavaBeans (EJBs) and based on the similarities in names, each corresponds to one of the components in the web tier.

7.1 Enterprise JavaBeans

EJB technology is the component-based architecture which constitutes the main processing layer in a JavaEE application. Each component is called a Bean and its functionality varies depending on its type. In this application, each of the three components of the processing tier are Message Driven EJBs. A Message Driven EJB is always associated with a JMS Destination. The EJB acts as a consumer of the messages from the queue. When a message is waiting in the queue, the application server will locate or create the required EJB and pass the message into the required *onMessage()* method of the EJB. If this method executes without throwing an error, the message is removed from the queue since it is considered to be processed. In this application, the three components perform different processing tasks.

The *PutDribBean* first determines if it has been sent a new Drib or an update to an existing Drib. If it is a new Drib, it is assigned a unique identifier and then added to the dataset. If it is an updated Drib, the Drib with the corresponding identifier is retrieved from the dataset. The information is merged between these two objects and the result replaces the existing entry in the dataset. Whenever a new or updated Drib is received by this Bean, the time value stored in the Drib is updated to reflect the current system time. In this way, the time stored in each Drib represents the most recent modification of the Drib.

The *GetDribSubjectsBean* receives messages which request lists of DribSubject objects for a particular location. This Bean then uses the dataset to retrieve all subjects which are within an approximately 3 km radius of the specified position. Since this list can become very large, only the most relevant subjects must be sent to the client. To achieve this, the Bean uses the algorithm described in Section 7.4 to calculate a popularity score for the subject. The size of the list is reduced to the number of results required by the client. In order for this Bean to return the list to the webservice which requested it, the JMS *replyTo* destination is used. In this way, the Bean uses the synchronous JMS approach which is described in Section 7.2. The *GetDribsBean* only differs from the previous Bean in that it processes requests for messages for a particular subject.

The use of EJBs in the processing tier again leads to a modular system design. This configuration is very beneficial from a development point of view because the functionality is divided into segments. Each component can be individually tested and the logged messages indicate exactly which one of the EJBs caused a particular fault.

7.2 Synchronous JMS

As described in Section 6.2, JMS is usually used as an asynchronous communication technology. However, in this application, a request-response model is needed so that the webservice can request messages or subjects for a particular location from the processing layer and the processing layer can return the required list. This is achieved by using a synchronous JMS configuration.

If the web tier component which is sending the message requires a reply, a temporary JMS Destination must first be set up by the sender. This temporary destination is the same as a normal destination except that only its creator may receive messages from it. A reference to this temporary destination is included as the *replyTo* parameter in the original message. Once the message has been sent, the sender waits for a response in the temporary queue. When the EJB receives the message, it procedes with the processing and repackages the result as a new JMS message which is sent to the temporary queue. In this way, the webservice which requested the response can then send it to the client.

There is a built-in method for using JMS in a synchronous manner and this method automatically constructs the temporary reply queue. However, this method does not allow for a timeout to be set so if a reply is not received due to a possible error in the processing tier, the webservice which used this method will wait indefinitely. This affects the robustness of the system and so in order to alleviate this restriction, synchronous JMS was configured manually from the web tier and a timeout was added that allows the webservice to continue even if the reply was not received. This increases the reliability of the system.

By using synchronous JMS, the system can still take advantage of the load-balancing capability of JMS while using a request-response communication model. As explained in the previous section,

the use of JMS improves the scalability of the application.

7.3 Unique Identifiers

One of the critical requirements for this system is the identification and referencing of the different Drib and DribSubject objects. Each object requires an identifier which is unique throughout the system. This identifier must also be sent with the object over the network to the client.

It is possible to use Java's built-in functionality to generate a standard Universally Unique Identifier (UUID) but this results in a very long identifier string. Due to the bandwidth cost and limitations of the mobile network, it is not efficient to send such a long identifier. The application therefore generates its own identifiers in integer format.

The application maintains a system-wide list of the identifiers which are currently in use. When a new identifier is requested, the application generates a random integer and checks if it exists in the list. If it is already in use, the system re-generates it and checks again. Once an unused identifier is found it is added to the list and assigned to the object. When an old object is deleted, its identifier can be removed from the list.

7.4 Popularity Score

The popularity score is a measure of how relevant a particular Drib or DribSubject is to a particular position at a certain instant in time. This score is calculated for each object to facilitate comparison between the objects.

In the case of a subject, the popularity is linearly influenced by the number of times the subject has been viewed (v) and the number of messages linked to that subject (m). As the distance from the subject's location increases or the subject becomes older, the popularity decreases. This decrease can be roughly approximated by stating that the popularity halves for every kilometre of distance from the user's location and/or for every five minutes older the subject becomes.

For a message, the number of times users 'liked' the message (l) is used to scale the popularity. Since this value would initially be zero for a new message, a constant factor is also included. The message popularity halves in the same way as the subject popularity. The formulae for the calculation of the popularity score are shown below where x and t represent normalised distance and time:

$$DRank_{subj} = (1000(v) + 1000(m))(\frac{1}{2})^{\Delta x + \Delta t} \quad (1)$$

$$DRank_{msg} = (1000 + 1000(l))(\frac{1}{2})^{\Delta x + \Delta t} \quad (2)$$

Once the popularity scores have been calculated, the subjects or messages are sorted using a built-in implementation of the merge-sort algorithm. This algorithm is used for its speed in performing the sort since this is an important factor in reducing server load. A quick-sort algorithm could have been used but one of the worst-case scenarios with this algorithm is when the list is in exact reverse order. This condition is very probable since the data storage system follows a first-in-first-out principle and the popularity score decreases with time so the first-out objects are the most likely to have the lowest score and vice-versa.

8 DATA STORAGE TIER

The Dataset interface provides a means of accessing the underlying data storage system from the business tier. This interface hides the unnecessary complexity of the storage system from the EJBs while still allowing sufficient functionality to perform the business logic operations.

In this application, the data storage system is a Derby relational database. This database is included with the Glassfish server since it is the recommended choice of database for use with applications of this type. However, since the business tier is designed to use only the Dataset interface, it does not matter what type of system is used for data storage. By designing a different class which also implements this interface, it is theoretically possible to change the underlying storage layer from a database to something like a cloud storage system. Since the business tier is only loosely coupled with the data storage tier, it relies on the assumption that all configuration and maintenance of the underlying infrastructure is performed by the class which implements the Dataset interface.

9 SYSTEM TESTING

In order to test the functionality of the system, various different approaches were used. The first was to generate comprehensive logging output from all components of the application. On both the client and server side, this was done using the standard logging tools of the platform. The Glassfish output log gives precise details of any error including the nature of the error and the component which caused it.

In order to test the webservices in the web communications tier, the RESTful webservices testing framework provided with the NetBeans IDE was used. This allowed for testing of each individual webservice under controlled conditions. The tests were initially run from the same machine so as to eliminate any possible errors caused by the network. Later, these tests were run from separate machines over a LAN and finally over the internet.

To ensure the complete functionality of the system, end-to-end integration testing was performed using both an emulated version of the client application and an application running on a physical device. This arrangement also allowed for testing of concurrent access to the system by multiple clients.

Load testing for the server-side components was investigated but was not implemented due to time constraints. This type of testing would generate multiple simultaneous messages and requests so as to place a high load on the server. By evaluating the server's performance during such a test, any computational bottlenecks could be identified and corrected. This type of test would give a good measure of the scalability of the system.

10 PROJECT MANAGEMENT

Since this system was designed and implemented by a team of developers, aspects of project management were critical to the outcome of the project. Although the main points on this topic are covered in the group report [4], additional relevant points are presented in this section.

10.1 Development Paradigm

In developing the components of the web communication tier and the business processing tier, a pair-programming approach was used as recommended by the eXtreme Programming (XP) development paradigm [5]. In this approach, A. Paverd worked collaboratively on most of the components with D. Mankowitz. This strategy proved to be beneficial both in terms of designing and implementing the system.

In order to facilitate this collaborative programming, a Git version control system was used [6]. This distributed source code management system was chosen for its advanced capabilities in merging two modified versions of the same file. The source code is also stored in a central location as a publicly accessible Git repository at http://github.com/ajpaverd/Dribble.

To avoid compatibility issues, continuous integration and testing tactics were employed throughout the development of these components. By testing for functionality as soon as a component was operational, a number of critical issues were identified and corrected before they had a significant effect on the overall project.

10.2 Time Allocation

The author's original estimate of the time required to complete this section of the project was 38 hours. This was purposefully set above the recommended 31.5 hours since a significant part of the system involved the use of new technologies and systems.

The actual time taken for the project was approximately 48 hours. Even though the estimate was increased to compensate for the use of new technology, this was still a factor exceeding the estimate. The use of new technologies on the client side also required extra time during the integration phase. Another factor was that this part of the project was subject to a small amount of scope creep. Features which were not originally part of the specification were added to the project and although they were successfully implemented and add significant value, they increased the time beyond the original estimate. The detailed breakdown of the time spent on the project is shown in Appendix A.

11 SYSTEM ANALYSIS

This section provides a critical analysis of the design and implementation of the web communication and business processing tiers in terms of how well they meet the requirements and what trade-offs were made in the final design.

11.1 Requirements Verification

The requirement of scalability is met in various ways within these two functional tiers. The use of a component based architecture consisting of webservices and EJBs allows the system to scale by dynamically creating more instances as the need requires. The use of JMS allows the system to perform load levelling by using different quantities of webservice and EJB instances. However, the scalability of the system may be limited by the current implementation of the unique identifier generation algorithm.

In terms of reliability, all components have been designed with a level of fault-tolerance such as that demonstrated by the timeout in the synchronous JMS implementation. The enterprise grade Glassfish server also adds a significant amount of reliability to the system.

By using modern technologies such as JavaEE version 6 and current best practices, this application can be classified as a modern software development project.

11.2 Design Trade-offs

In the final design of these two functional tiers of the server application, the primary trade-off was between scalability and performance. The use of technologies such as JMS and even the application server itself reduce the performance of the application by adding overhead and complexity to many operations. However, these technologies are the key to the system's scalability. At low usage levels, the performance costs of these technologies are greater than the advantages but at very high levels, the advantages far outweigh the performance costs. Therefore there is some nominal level of usage of the system above which the advantages of the scalability design decisions begin to appear.

12 FUTURE WORK

In order to improve the system, a number of enhancements can be made. On the server hardware side, the application could be configured to run on a cluster of machines. This would increase the overall performance of the system and allow greater scalability. The cluster could also be configured to provide a high availability service by using multiple redundancy. This is important because the central server is a single point of failure for the complete system. However, since users do not store personal information on the system, a server failure would not cause any loss of important data.

Within the software components, an enhancement can be made to make the locations of the subjects change dynamically based on the posted messages. At present, the location of each subject is fixed at the point where it was created. By using a weighted averaging algorithm, the location of the subject could be dynamically shifted so as to always represent the geographical centre of its messages.

The current implementation of the data storage tier requires manual maintenance on the underlying database. This could be improved by implementing a self-healing data storage system.

To increase the relevance of messages and eliminate the possibility of spam, the popularity score algorithm could be replaced by an intelligent control system possibly using a neural network to determine the popularity rankings of subjects and messages.

Finally, the application has a potential for commercial viability. The entity which controls the central server and data storage system has access to all the messages and subjects present in the system. Using data mining and analysis techniques, it would be possible for the controlling entity to identify popular events and newsworthy stories using this crowd-sourced data. Since this information is completely anonymous, it could be legitimately sold to interested parties such as news or advertising agencies. The system would still remain completely free for users.

13 CONCLUSION

Dribble is a geosocial networking system which allows users to upload and view messages relevant to their current locations. The primary requirements of the system are that it is scalable and reliable. The system uses a component-based architecture consisting of various functional tiers. This design uses loose coupling between components which improves the flexibility, maintainability and reliability of the system. The client-side consists of an application for the Android platform. The server-side application uses JavaEE technology and the Glassfish application server to provide a modern and reliable implementation platform. The web communications tier uses RESTful webservices and XML to communicate with the client applications. JMS queues are used for passing messages asynchronously from this layer to the business processing tier. The processing tier consists of Message Driven EJBs which perform the required processing such as the assignment of unique identifiers and the ranking by popularity score. The Dataset interface provides a standardised interface to the underlying data storage system. The system was tested using multiple methods including integration testing. A pair-programming paradigm, Git version control system and continuous integration were used in the development of these components. The project took slightly longer than the original time estimate due to new technologies and partial scope creep. Although the system meets all the specified requirements, the primary tradeoff is between scalability and performance. There is potential to improve this geosocial networking system including a possibility of commercial viability.

REFERENCES

[1] R. Gross and A. Acquisti. "Information Revelation and Privacy in Online Social Networks." In ACM, editor, *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. New York, 2005.

[2] Mashable. "Foursquare: Why It May Be the Next Twitter." URL `http://mashable.com/2009/07/25/foursquare-app/`. Last Accessed: 29 April 2010.

[3] Foursquare. "foursquare." URL `http://foursquare.com`. Last Accessed: 29 April 2010.

[4] A. Campbell, C. Epstein, G. Favish, D. Mankowitz, and A. Paverd. *Dribble: The Design, Implementation and Analysis of a Geosocial Networking System*. University of the Witwatersrand, Johannesburg, May 2010.

[5] D. Wells. "Pair Programming.", 1999. URL `http://www.extremeprogramming.org/rules/pair.html`. Last Accessed: 29 April 2010.

[6] Git. "Git Fast Version Control System." URL `www.git-scm.com`. Last Accessed: 29 April 2010.

Appendix A - Time Management

Table 1 : Estimated and Actual Times for each Sub-task

Sub-task	**Estimated hours**	**Actual hours**
Server-Side		
Setting up Web-server	4	6
Web Services	6	7
Queues	6	8
Enterprise Beans	4	7
Processing Algorithms	6	6
Dataset Interface	2	2
Database Implementation	2	2
Integration	4	5
Testing	4	5
Total	**38**	**48**