ELEN4010
Software Development III

Author:
Greg Favish
0713568E

Group members
A. Campbell
C. Epstein
D. Mankowitz
A. Paverd

Abstract

Dribble mobile client-server application that has been developed. It is a social networking application that allows users to send anonymous message into the public space. It is also location-based so one can only see messages that have been posted within a certain radius of the current location. The mobile application was developed for Android phones and the server was developed Java Enterprise Edition (JEE) framework and consisted of a multi-tiered architecture that was deployed onto a Glassfish Application server. This report details the individual contribution of Greg Favish to the Application. It focuses on data storage on the server side of the application.

1. Introduction

The client-server application Dribble was designed, implemented and tested. The application is large and has many components. The purpose of this report is to explain the development of the database and how it was interfaced with the application web-server as well as explain how the application functions as a whole. To see the exact breakdown of time spent implementing the project ,refer to Appendix A.

2. Application overview

In this application the messages are called by users, and in the code itself, “Dribs”. The topic of a drib is called a “Drib-Subject”.

The application will now be explained from the perspective on an end user. The user will start the application from an android cell phone. They will then have the option to either start a new Drib-Subject or enter a drib or they can elect to view all dribs within 3KM their current location at the time and reply to any drib that they can view. They can also view all the topics that are currently “active” within the 3KM radius. These simple rules of what a user can and cannot view and reply to can result in interesting behaviours if many users use the application. For instance a topic that has been created in one location can spread as people reply to dribs. This dynamic behaviour is shown in figure 1.

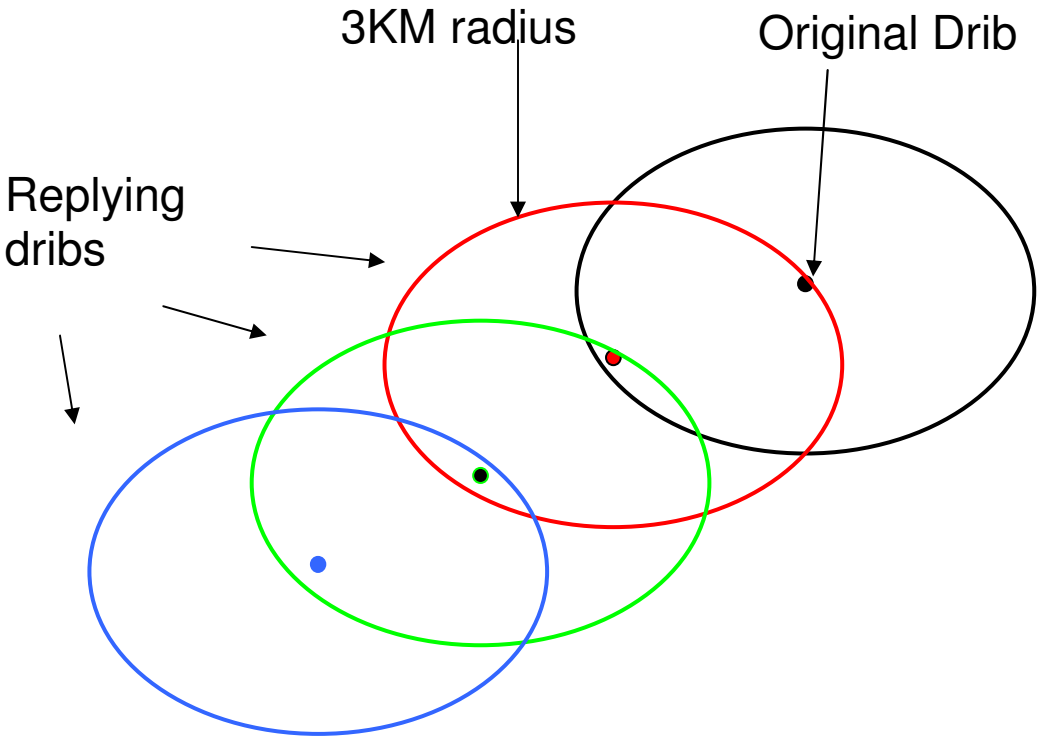


Figure 1: Dynamic behaviour of Dribble

3. Total System Technical Overview

The following is verbatim from the group report [x]. It explains the design of the entire application from a technical standpoint.

A generalised system model of the overall Dribble system is depicted in Figure 2 The system is primarily divided into two components – the Mobile Application (Android) and the Remote Application Server (Glassfish). The Mobile Application is executed on the HTC Dream mobile phone and provides the GUI with which the users will interact. The mobile application is composed of logical JavaME source code and presentation layer XML scripting. The mobile phone initiates a connection to the server via web services, set up on the remote application server. The mutual packages shared between these two sides, define the two main object classes `Drib' and `Drib-Subject' which are used to convey messages and message subjects, respectively, to and from the server. The objects are serialised in order to be sent over a HTTP connection, using one of the following network technologies: 3G, EDGE, GPRS or HSDPA. The sent object is de-serialised on the server side, and processed using Enterprise JavaBeans. The database is queried with the relevant request and returns the desired result. This result is returned using Enterprise JavaBeans and serialised to be sent back to mobile application using RESTful web services. The mobile application uses Java and XML to graphically display the Dribs and DribSubject.

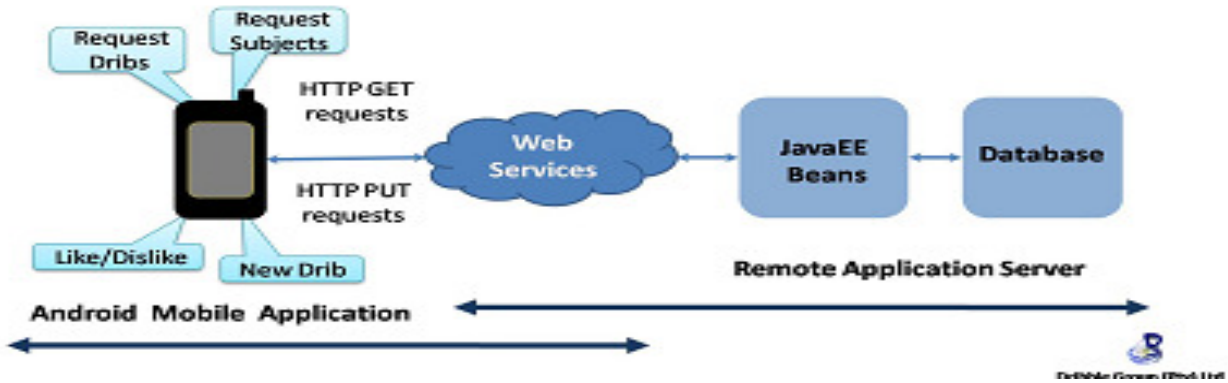


Figure 2: Software model of Dribble

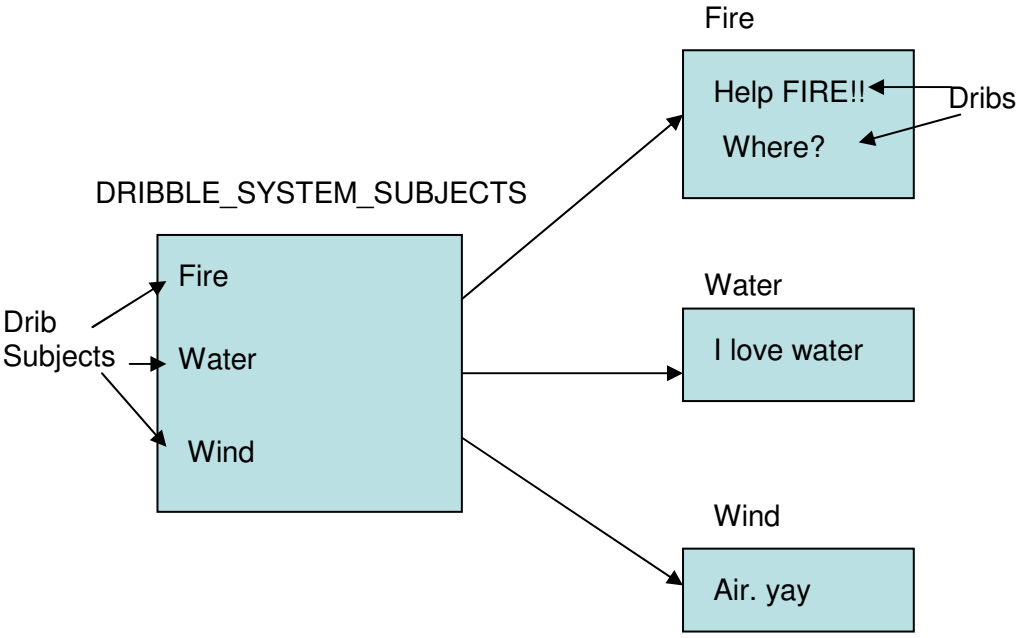
4. Database

4.1 Database

The actual database used was a Derby Relational Database. Derby is an embedded database management system that is included with Glassfish. [1] It uses the Java, JDBC, and SQL standards. It was chosen for use due to its easy integration into the web server as well as Java. The structure of the database in relation to dribs and drib subjects is shown in the figure below.

As can be seen the database was structured more in a logical fashion that an efficient fashion. Each Subject has its own table where dribs are stored. The

entire lists of subjects are stored in their own table. As new subjects are created by users more tables are dynamically created by the code.



The actual storage of dribs, and drib subjects was eventually implemented using a database however the design of the way dribs were stored was implemented through an interface.

4.2 Interface

The interface created was called Dataset. This implies that dribs were stored in some form (not necessarily a database) .The interface was designed with the basic functionality to:

-store dribs
-retrieve dribs based on location
-store and retrieve drib subjects
-update dribs

The interface gave the allowed the main server to call the functions of the interface but the actual internal code can be very flexible as long as the correct parameters were input and returned. Thus it makes it very easy to create scalable storage of all dribs. As long as the interface is adhered to the actual storage can be done on a small embedded database or if needed a large cloud storage server. The actual implementation of the dataset that was used is a class called SQL communicator. The interface was designed and agreed upon by the whole group initially so that it could perform the functions that were needed in the webserver.

4.3 SQLCommunicator

SQLCommunicator implements Dataset. As implied by the name the way of interacting with the Database is through SQL. The actual database is connected to in the constructor of SQLCommunicator. The core of the class converts objects (e.g. dribs) into records in a database. This class establishes the connection to the database using a Java Database Connector (JDBC) driver on initialisation. In order to improve the system efficiency, this connection is obtained from a connection pool maintained by the server. Apart from the methods required by the interface, this class also provides auxiliary methods which are used to facilitate the primary methods. Each method and how it relates to the application as a whole will now be explained.

adddrib(Drib d)
This method adds a drib to the database. If the topic exists then it will add it to the relevant table. If the Drib Subject does not exist it will create a new table for the Drib Subject then add the drib to this newly created table. Also updates the post count for the Drib Subject being added to.

deleteDrib(Drib d)
This method deletes the record of the drib sent into the method. This method also updates the post count of the topic and deletes a topic if it does not hold any dribs.

updateDrib(Drib d)
This method looks for the drib that is given in the database and updates it with new values for its data memebers.

getDrib(DribSubject s, int DribID)
This method returns a drib object made from the record in the database using only its subject and ID to identify it.

DribSubject getDribSubject(int SubjectID)
This method returns a drib Subject object made from the record in the database using only its ID to identify it.

DribSubject getDribSubject(String SubjectName)
This method returns a drib Subject object made from the record in the database using only its name to identify it.

getDribs(DribSubject s, double lat, double longitude, double radius)
This method returns all dribs in a certain subject within a certain range of a geographic coordinate.

getDribSubjects(double lat, double longitude, double radius)
This method returns all drib subjects that contain dribs within range of a geographic coordinate

5. Improvements

Although the database functioned correctly it would greatly simplify the code if the database was an Object orientated database. Instead of relatively error prone SQL strings the database would be accessed by just using methods to store and retrieve entire objects from the database. [2] The upside to this is flexibility. If the drib object for instance has an extra parameter added the database interfacing code should not break. However object orientated databases are less standard and more niche than normal relational databases and some of the benefits of SQL are lost (e.g. complicated queries that include calculations and processing).[3] However for our applications an Object orientated database still may be more appropriate to use.

6. Conclusion

The client-server application Dribble was designed, implemented and tested. The application is large and has many components. The report explained the development of the database and how it was interfaced with the application web-server as well as explain how the application functions as a whole. The database design and interface was explained.

8. References

[1] Apache Derby http://db.apache.org/derby/ Last Accessed: 30 April 2010.

[2] B. Obasanjo. An Exploration Of Object-oriented Database Management System.", 2001. URL: http://www.25hoursaday.com/WhyArentYouUsingAnOODBMS.html. Last Accessed: 30 April 2010.

[3] M. D. Zdonik, S. B. Object-oriennted Database Systems. 1990.

Appendix A: Time Breakdown

The following table shows both the estimated and actual times spent by Greg Favish on components of the project.

Task	Estimated time (hours)	Actual time (hours)
General Group discussions and design	5	8
Interface Implementation	5	3
Database Implementation	10	15
Integration	5	7
Testing	5	5
Total	30	38