Mobile Voting System

Final Year Project Final Report

Liam Whelan

A thesis submitted in part fulfilment of the degree of BSc. (Hons.) in Computer Science with the supervision of Dr. John Murphy and moderated by Dr. Neil Hurley.

School of Computer Science and Informatics

University College Dublin

05 April 2006

Abstract

The goal of this project is to create a GPRS based mobile voting system that is more natural, graphics based and user friendly than current systems. The project will define and implement a GPRS socket based communication protocol between a server and a mobile client. The client will perform a vote in a rich, graphical environment and the server will relay information to the client by interrogating an SQL database.

Acknowledgements

I would like to thank O2 Ireland, for their generosity and sponsorship of the project. O2 provided mobile devices & SIM (Subscriber Identity Module) cards free of charge and without limitation for the duration of the project.

I also acknowledge Eamonn Lawler, a previous fourth year Computer Science student, who developed the basic mobile voting system as part of his final project last year. It is based on his existing work, that I added the extra features and functionality.

Table of Contents

1 Introduction

In this section, the full project specification is outlined along with the problems concerning current voting systems. A brief description of the alternative approach follows and a summary of what the rest of the document entails.

1.1 Specification

Description
The aim of this project is to create a working GPRS-based voting system for use on a mobile device. Voting systems currently rely on a text and a number based system, which is cumbersome and difficult to use. This new system should be more natural and graphics based. The voting system will work as follows: the client will select a vote, by using a word or phrase. The server will respond by sending a list of options that can be voted upon. This part of the process will make use of graphics and lists in order to make the process more user-friendly. The server will then register the vote selected by the user. It would also be desirable to implement a vote management system through which a user can enter information pertaining to a particular vote that they want to conduct (including graphics), monitor the current status of a vote, initiate a vote and terminate a vote, specify how the vote is performed - via sms or a gprs socket based interface. This work will build on previous work that was performed.
Mandatory
Take the existing software and get it working on a mobile device. Add functionality to enable graphics to be stored in the database and be rendered on the mobile device. Add functionality to enable the current statistics in the database to be viewed via a web interface.
Discretionary
Add functionality to enable voting to be done via sms or a gprs socket interface. This should be configurable information, which should be in the database. Add functionality to enable votes to be added/deleted/terminated from the system.
Exceptional
Add security functions - add authentication functions to management system and add functions to track multiple votes and remove them if necessary.

1.2 Limitations of current voting systems

In this modern era, with advanced wireless mobile technologies, it would be expected that a simple and modern mobile voting system to be in place and yet many current mobile voting systems still rely on an inefficient and cumbersome SMS-based system which requires the user to send a series of specific words or phrases to a number in order to register a vote.

These SMS based voting systems are abundant nowadays. The voting procedure for television programs like 'You're a Star' is an example of such a system. The user is typically required to send a text message containing a specific phrase on their phone to a specified number. A while (sometimes a significant time) later a message is received back, listing a number of choices for the vote. Once again the user is required to send a phrase by SMS to the designated number and must wait for confirmation of their vote or be greeted by an ugly list of statistics.

This system is clearly outdated and needs to be replaced by a more intuitive and user friendly GPRS based system.

1.3 A solution

Indeed the primary purpose of using SMS as the voting technique is so that the user can be charged for the vote. However, this old SMS technology lacks functionality to provide an attractive user interface, and is awkward, ugly and slow to use.

This project proposes to build on an existing project and develop a GPRS based system, so that a user can vote on a topic in a natural and user-friendly environment, far more efficiently. Also an option for the administrator to charge for the vote or not will be easily configurable through a web based administration system. This is achieved by registering the vote partially by SMS, invisibly to the user, and is discussed in more detail later.

1.4 Rest of this document

In the chapter titled 'Strategy & Design', a detailed insight into the methodology and strategy used to tackle the problem is presented along with the order of high-level components and tasks required to implement the proposed system. Aspects of the system where problems occurred are discussed along with issues, which were of concern as the project progressed.

The 'Detailed Design & Implementation' chapter focuses on system design at a class level, and describes the system using popular UML diagrams and notation.

'Evaluation & Analysis' includes screenshots of the completed system, and tests that are performed relating to robustness and performance. The final section provides a brief conclusion of the project along with limitations of the approach and future work that may be carried out.

Firstly though, the document concentrates on relevant background research & information relating to the development of the project. The general order in which topics were researched is conserved in the document. It is assumed that the reader has little or no knowledge concerning Java mobile technologies.

2 Background Research

This section will focus on the technologies that were used to implement the system, and why certain technologies were selected ahead of others.

2.1 GPRS

General Packet Radio Services (GPRS) is a packet-based wireless communication service. It is based on Global System for Mobile (GSM) communication and complements existing services such as circuit-switched mobile phone connections and the Short Message Service (SMS). GPRS splits information into distinct but related ‘packets’, transmits these packets of data and reassembles them on the receiving end. Radio resources are only used when users are sending or receiving data, this mean that GPRS allows more efficient use of bandwidth.

GPRS is a mobile bearer of IP traffic and it is this functionality that allows the client to talk to the server using the TCP protocol.

Data rates are reasonably high and theoretical maximum speeds of up to 171.2 kilobits per second (kbps) can be obtained with GPRS [1]. Realistically speaking, average speeds range from 56 – 114Kbps. These higher data rates, along with continuous connection to the Internet for users allows for a more efficient and enjoyable voting experience.

2.1.1 Advantages over SMS

GPRS facilitates instant connections whereby information can be sent or received immediately. This is not the case with SMS. GPRS also supports new applications that were not previously supported, due to limitations in speed of circuit switched data (9.6 kbps) and message length of the Short Message Service (160 characters) [2]. Intuitively a GPRS implemented mobile voting system would be far more user-friendly and easy to use than current SMS-based ones. Applications may also be downloaded to a device via GPRS from a public web server.

2.1.2 Limitations

Since GPRS uses a packet switching strategy, packets intended for a single destination may be sent through different access points to reach a target. Even though GPRS incorporates retransmission and data integrity techniques, *transit delays* may occur if a packet is lost or corrupted during radio transmission. Also, the GPRS standard does not incorporate storage mechanisms such as the Store and Forward mechanism used by SMS.

2.2 Server / Client system

It is evident that the voting system requires a server / client based implementation. The server implemented in J2SE, acts as a host to multiple J2ME client connections. To provide functionality to handle many simultaneous users voting at once, the server would need multithreading capability. This means that many threads, or client connections, can run concurrently.

Firstly the server waits (or listens) for a client to connect. Once a connection is established, the client and server communicate via *sockets*.

2.2.1 Sockets

A socket is a single end-point of a two-way communication link between two programs running on a network. A socket needs to be bound to a port number so that the TCP layer can identify the destination application for which the data intended for [3].

A connection between a client program and a server program is represented using Socket classes in Java. Two classes are provided by the java.net package: Socket, which implements the client side of the connection, and ServerSocket which implements the server side of the connection.

Usually, a server has a socket bound to a specific port number and waits indefinitely to the socket for a connection request from a client. The client makes a connection request by specifying the server's hostname and port. Once a connection is negotiated the server and client communicate by writing and reading to their sockets.

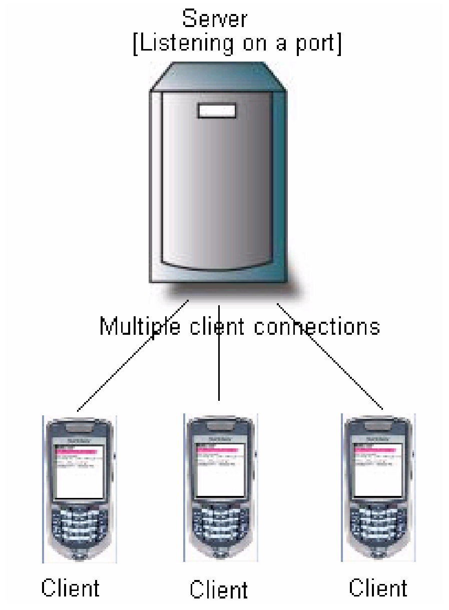


Figure 1. Server with multiple simultaneous client connections.

2.2.2 TCP vs. UDP

Transfer Control Protocol (TCP) imposes reliability of data transfer by using *flow control*. Flow control ensures packets are re-sent where necessary, and all packets are in sequence. User Datagram Protocol (UDP) however offers no form of flow control or error detection [4]. It is often the preferred protocol for streaming audio and video due to its extra speed, whereas TCP is used for WebPages, database information etc. where loss of data is unacceptable.

UDP works well in peer-to-peer designs where there is not always a central server; instead each host communicates with whatever host it needs to. TCP is obviously the better choice for the server/client architecture used in the project. It is stream orientated and ensures a reliable, lossless and more secure form of communication.

2.3 Java 2 Platform

The Java 2 Platform is more commonly referred to as simply 'Java'. It is split into three editions: Java 2 Enterprise Edition (J2EE), Java 2 Standard Edition (J2SE), and Java 2 Micro Edition (J2ME) [5].

A complete environment for running Java-based applications is provided by all three editions, including runtime classes and the Java virtual machine (JVM). Each edition also targets particular applications running on certain device architectures. J2EE is used to develop server-based applications, emphasising deployment and component based programming. J2SE provides all necessary user interface classes for desktop-based application development. J2ME, which is of particular interest in the context of this project, targets embedded and handheld mobile devices.

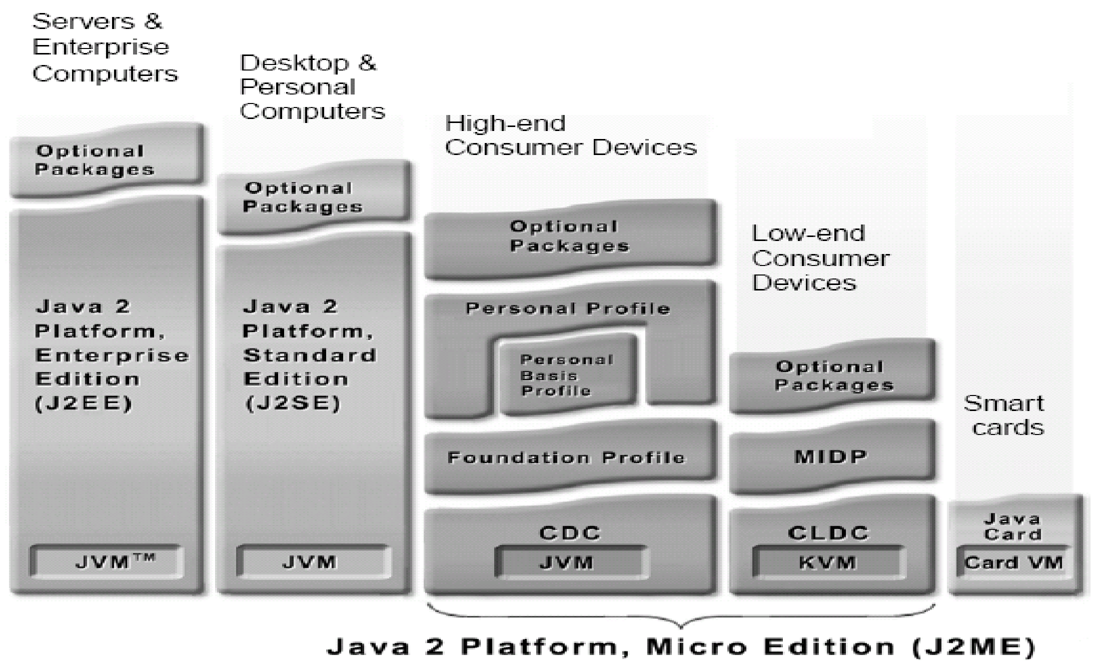

Servers & Enterprise Computers
Desktop & Personal Computers
High-end Consumer Devices
Low-end Consumer Devices
Smart cards
Optional Packages
Optional Packages
Optional Packages
Optional Packages
Java 2 Platform, Enterprise Edition (J2EE)
Java 2 Platform, Standard Edition (J2SE)
Personal Profile
Personal Basis Profile
Foundation Profile
MIDP
CDC
CLDC
Java Card
JVM™
JVM
JVM
KVM
Card VM
Java 2 Platform, Micro Edition (J2ME)

Figure 2. Java 2 fragmentation

Simply put, J2SE could be interpreted as a subset of J2EE and J2ME as a subset of J2SE. Each edition defines a set of class libraries. For instance, a piece of Java byte code could potentially run on all three editions *provided* all editions have the set of classes required to run the byte code, available. Since J2SE runtime classes can take up megabytes of space, which of course too big to for small mobile devices, J2ME defines a much smaller set of class libraries for use in developing mobile applications.

The Java Community Process (JCP) defines the various specifications for all three editions. Java Specification Requests (JSRs) are the actual descriptions of proposed and final specifications for the Java platform [6]. At the time of writing of this document J2ME has seventy-six JSRs alone.

Since the main interest of the project lies in development of mobile device technology, the next section will focus on one of these three editions, J2ME, in greater detail introducing the concepts of configurations, profiles and optional packages associated with this technology.

2.4 Java 2 Micro Edition - J2ME

The Java Runtime Environment (JRE) in J2ME is adapted so applications can be developed for constrained devices i.e. a device, which may have certain limitations such as memory, processor speed, and screen size when compared to server or desktop computers. There are two broad categories of products at which J2ME is targeted [7]:

- *Fixed connected information devices ('high-end' devices).*
  In Figure 2, this category is represented by the section labelled CDC (Connected Device Configuration). Examples of devices here include TV set-top boxes, auto navigation systems, and internet enabled screen phones.

- *Personal and mobile devices ('low-end' devices).*
  In Figure 2, this category is represented by the section labelled CLDC (Connected Limited Device Configuration). Examples of devices included in this section are mobile phones, pagers, and personal organisers.

J2ME supports flexible deployment demanded by consumers and embedded markets, by having a modular and scalable architecture. It also offers a range of virtual machine technologies, each optimised for different processor types and memory footprints. For the low-end and resource limited devices as described above, J2ME supports minimal configurations of Java APIs and the Java Virtual Machine (JVM). These special configurations capture only the essential capabilities of the device. As new features and applications are developed for these devices, additional APIs can be specified for the configuration.

J2ME, unlike J2SE, is not a single specification or a piece of software. It is a platform, a collection of specifications and technologies all of which are designed for different parts of the small device market. There is three core concepts at the heart of Java 2 Micro Edition: *configurations*, *profiles* and *optional packages*. To develop a J2ME application, understanding of these concepts is vital since they determine which Java features you can use, which Application Programming Interfaces (APIs) are available, and how your applications are packaged.

2.4.1 Configurations – CDC / CLDC

A configuration consists of three main things: a set of core Java runtime classes, a JVM to execute byte code, and native code to interface the underlying system. Altogether they make up a Java Runtime Environment [8]. A device must meet certain requirements in order to use a configuration. The configuration's formal specification defines these minimum requirements. The set of core classes is generally very small and needs to be complemented with additional classed provided by J2ME *profiles*.

Two configurations are defined by J2ME, namely the Connected Device Configuration (CDC), and the Connect Limited Device Configuration (CLDC). The CDC is a superset of the CLDC and provides a full Java VM and a reasonably large set of core classes. It thus requires higher-end devices with larger memory capacity and faster processors.

CLDC is of particular interest in the context of this project. It is designed for very resource-constrained and small limited devices i.e. devices with slow processors and/or small memory capacity. The JVM supplied by the CLDC omits many important features and the set of core runtime classes is bare minimum; a tiny fraction of J2SE core classes.

As with all J2ME technology, the CLDC is a specification that had passed through the JCP. CLDC v1.1 is defined in the JSR 139. The set of APIs that it defines for input/output is known as the Generic Connection Framework (GCF). However the CLDC does *not* define APIs related to user interfaces, or how applications are loaded, activated or deactivated on a device. Profiles that use the CLDC as their base, define these and other things. They are discussed in the next section.

2.4.2 Profiles – MIDP

A profile fills in missing functionality of a configuration by adding domain specific classes. Most profiles, for example, allow interactive applications to be built by defining interactive classes. A device, in order to use a particular profile, must meet the minimum requirements of the underlying configuration and also all of the additional requirements as mandated by the profile's formal specification.

We will concentrate on the *Mobile Information Device Profile* (MIDP). It is a CLDC based profile for running applications on resource-constrained devices. JSR 118 defines MIDP 2.0. It adds many features to the basic APIs defined by the CLDC including support for application lifecycle management, HTTP based network connectivity based on CLDC's GCF, user interface support, and persistent storage of data.

MIDP applications need to follow specific packaging rules. The application is referred to as a MIDlet. The MIDlet's class must extend the `javax.microedition.midlet.MIDlet` class. It must define abstract methods which the main class implements. The MIDlet is notified that its state is changing through system calls to these methods. A MIDlet suite is composed of one or more MIDlets packaged together [9]. It contains a JAD (Java Application Descriptor) file and a JAR (Java Archive) file.

2.4.3 MIDP Life Cycle

The Application Management Software (AMS) is part of the device's operating environment and provides the runtime environment for the MIDlet. It guides the MIDlet through its various execution states while also enforcing permissions and security, managing downloads, installations and removal of applications.

These execution states are defined by the life cycle of a MIDlet; they are ***Paused***, ***Active*** and ***Destroyed***. Unlike desktop or server applications, MIDlets should not have a `public static void main()` method. If one is found the AMS ignores it. MIDlets are initialised when the AMS provides the initial class needed by CLDC to start the MIDlet. The AMS then guides the MIDlet through its various changes of state.

An instantiated MIDlet resides in one of the three possible states. There are three guidelines for creating MIDlet states: Initialisation of the application should be short, it should be possible to put an application in a, Non-active state and it should also be possible to destroy an application at any time. When a MIDlet begins execution, the AMS initially creates a new instance of the MIDlet. Once the constructor returns, the MIDlet is placed into the *Paused* state. The AMS then calls the `startApp()` method to shift into the *Active* state. In order to transition back to the *Paused* state again, the AMS calls `pauseApp()`. It is at this point that the MIDlet releases any resources which it obtained in `startApp()`. Thus the MIDlet may shift to and from these states at any time during execution.

The `destroyApp()` method is called with an argument. Its type is a `boolean` and, if `true`, the MIDlet will enter the *Destroyed* state. However if this argument is false, the MIDlet may send a request to the AMS, telling it not to enter the *Destroyed* state by throwing a `MIDletStateChangeException`.

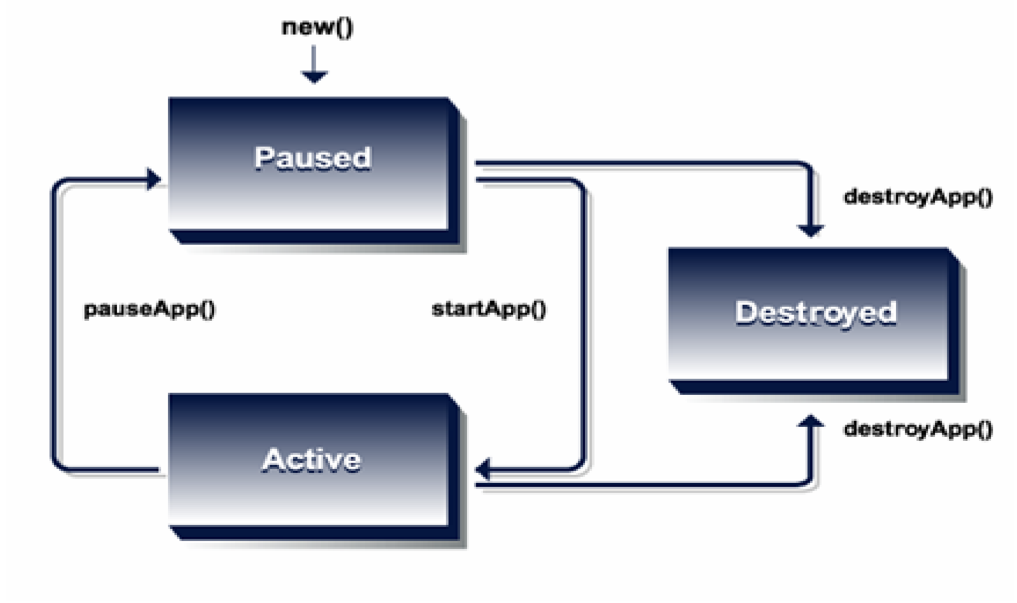


Figure 3. The MIDlet life cycle model

Understanding the MIDlet life cycle is fundamental to creating any MIDlet. Figure 3, above, shows valid MIDlet states and transitions.

2.4.4 OTA Provisioning

Over-the-air provisioning (OTA) allows a mobile application to be deployed on a large scale, and describes the ability to download & install a MIDlet suite over a wireless network [10]. Any public web server can be implemented as a download site provided it's configured to recognize MIDlet suites. The MIME-types for JAR and JAD files need to be specified in the server configuration.

A device can then proceed to download the MIDlet suite from the remote server by simply typing the URL of the suite's JAD file into the browser. It is at this point that the AMS takes over and manages the installation of the suite. Generally the sequence is as follows:

1. The client device sends a HTTP GET for the specified URL to the server.
2. Server sends a HTTP response with its message body being the suites JAD file.
3. Client verifies the response, extracts relevant attributes (`MIDlet-Jar-File` and `MIDlet-Jar-Size`) from the suite.
4. The client device sends a HTTP GET for the JAR file to the server.
5. The server responds by sending the JAR file as the message body.
6. Device verifies the JAR file and installation may begin.

2.5 MySQL Database

An SQL (Structured Query Language) database is a type of database technology that is currently the most widely used in computing environments. Data stored in such a database provides high levels of functionality by using a strictly structured format. Advantages of SQL database systems over older technologies include performance, robustness, security and access simplicity [11].

MySQL is a major SQL based database system, and is purely relational (RDMS Relational Database Management System). It reportedly has over six million installations worldwide to date. It has many uses, some of which include web applications like PHP-Nuke, and is closely linked to the popular PHP language, as PHP provides functions to manipulate MySQL databases.

MySQL is available on a wide variety of platforms and operating systems, and is classified as free software under the GNU General Public License (GPL). Administration of MySQL databases is typically achieved through the included command line tool, or alternatively several GUI administration tools are available for download.

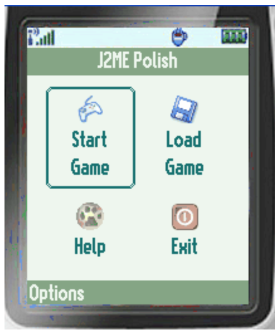
The SQL query language is used to retrieve, update and delete and manipulate data stored in MySQL databases. With regards connecting to the database and executing queries from a Java interface special drivers are required, such as a JDBC drivers.

2.5.1 JDBC Driver

MySQL connectivity for client applications written in the Java programming language is provided via a JDBC driver called MySQL Connector/J. This is specifically a JDBC 3.0 'Type 4' driver, which indicates that it uses the MySQL protocol to communicate directly to the MySQL server, and implements version 3.0 of the JDBC specification [12].

2.6 J2ME Polish

J2ME Polish (http://www.j2mepolish.org/) is a suite of open source tools for creating 'polished' J2ME applications [13]. Each of these tools meets a need of J2ME developers, and the collection of components includes:



- A powerful GUI (primary concern for the project), which allows you to design an interface with simple CSS text, files outside the source code.

- Build tools (specifically using ANT) with an integrated database of devices that allows you to build an application for multiple target devices.

- Game engine that allows use of the MIDP 2.0 API on MIDP 1.0 devices.

- Logging framework and collection of utilities.

More information and J2ME Polish documentation can be found at http://www.j2mepolish.org/documentation.html

2.7 PHP/HTML

PHP Hypertext Pre-processor (PHP) is an extremely popular and widely used open source tool and scripting language that can be embedded into HTML and is principally suited for web development. What distinguishes PHP from other languages like client-side JavaScript is that the code is executed on the server, rather than the client machine. Using PHP it is possible to create dynamic web pages quickly, and to retrieve, display and manipulate entries in a MySQL databases via a web interface.

2.7.1 Database communication with PHP

In the world of dynamic web scripting MySQL & PHP are sometimes referred to as the ‘dynamic duo’. They work particularly well together, and PHP provides a number of built-in functions to access and manipulate MySQL databases [14]. For example this simple PHP script connects to a database and performs a query:

```
<?php
    mysql_connect('csserver.ucd.ie','username','password');
    mysql_select_db(voting) or die('Cannot connect');
    mysql_query("DELETE * FROM topic");
?>
```

This simple script may be embedded directly into a HTML document, and will attempt to connect to the MySQL server using the username and password specified. Once the correct database is selected a query is executed which deletes everything from the *topic* table.

3 Strategy & Design

This chapter focuses in depth into the approach and procedure used at each step, right from analysing user requirements through to project completion. We take a look at some of the high level components required to build the system, and critique reasons for choosing certain approaches over others.

3.1 Analysing the requirements

A bad idea is to begin implementing the code right away. A significant amount of planning and research needed to be untaken even before looking at the existing code.

Initially a number of use case scenarios were developed. These ‘scenarios’ depicted possible features and functionality that users may want from the system and how they would interact with it. Obviously there would be two primary ‘actors’ or users of the system: The person voting on a mobile device & the administrator:

The **client** would require:

- Ability to easily download and install the application onto his/her mobile device.
- Quick and efficient response times from the server over GPRS.
- An attractive, user-friendly interface from which to vote.
- Confirmation of his/her vote and indication of vote statistics.
- Ability to vote again on another topic if desired.
- Ability to view current web statistics via a web interface

The **administrator** would require:

- Ability to easily & securely add/modify/delete votes from the system via a web interface.
- Start & expiration dates on topics so that users of the system can only vote on topics that are currently active.
- Ability to charge the user for specific votes if desirable.
- Stop users from voting on the same topic more than once.

3.1.1 Use Case Diagrams

The following Unified Modelling Language (UML) use-case diagrams [15], overview the usage requirements of the system in terms of the client user and, secondly, the project administrator.

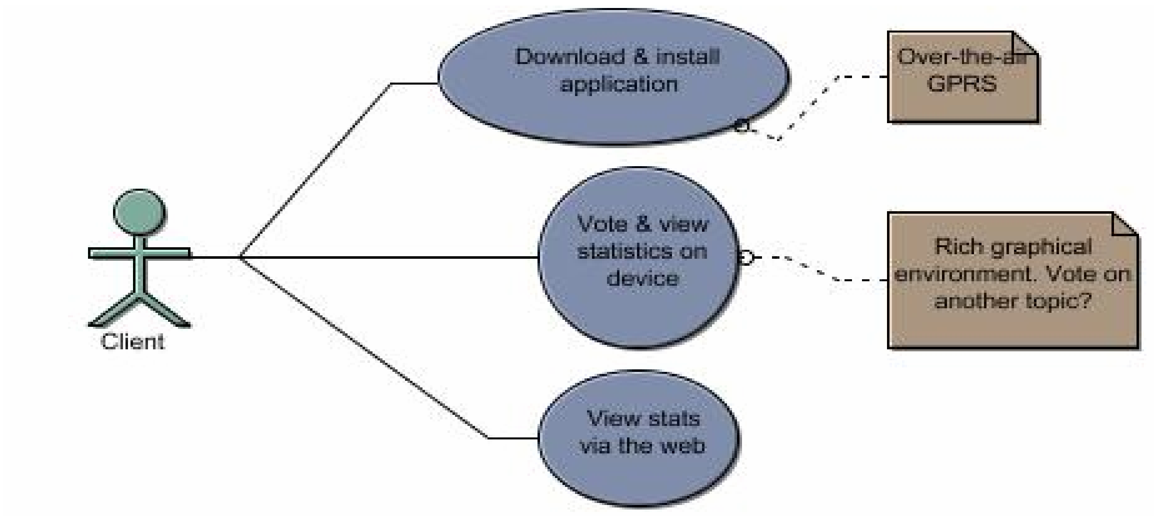

Download & install application
Over-the-air GPRS
Vote & view statistics on device
Rich graphical environment. Vote on another topic?
Client
View stats via the web

Figure 4. Use Case model for the client user

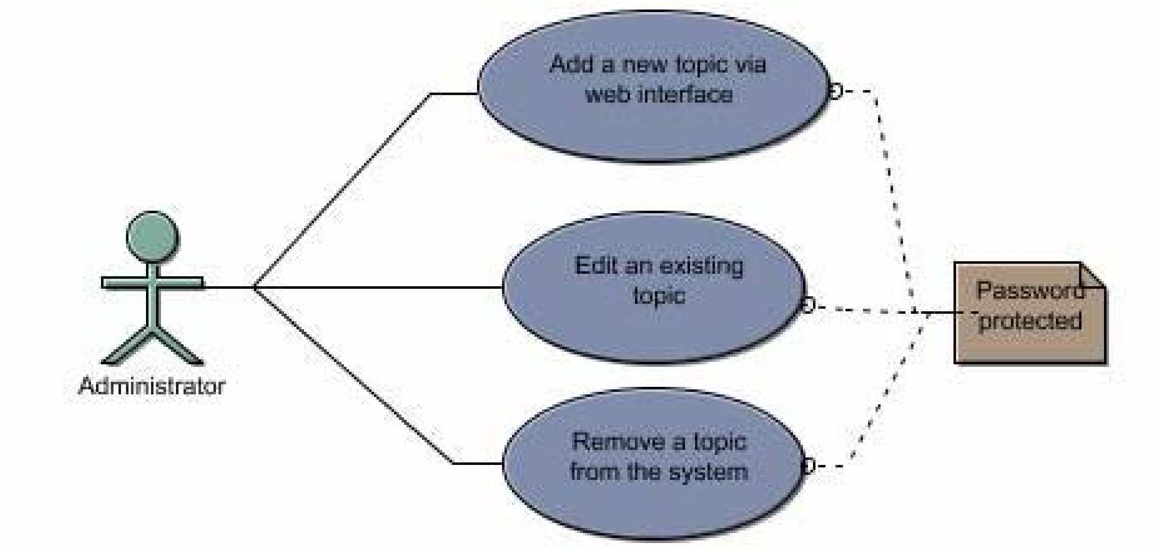

Add a new topic via web interface
Edit an existing topic
Administrator
Password protected
Remove a topic from the system

Figure 5. Use Case model for the system administrator

3.1.2 Next steps

Once the initial analysis of user requirements was completed, the core components of the system were chosen.

Eclipse IDE (Integrated Development Environment) was the IDE of choice to develop both the server (written in standard Java, J2SE) and the device application (using J2ME libraries). It is a very easy-to-use, reliable and popular IDE that provides a comfortable environment on which to code. Eclipse is free, open source software and plug-ins for various different languages and Java editions are widely available. The particular plug-in of interest was EclipseME (http://eclipseme.org/), used to develop J2ME MIDlets which allows the developer to easily connect wireless toolkits to the Eclipse development environment. This allows focus to stay on development of the application as opposed to specific concerns relating to J2ME.

Before the project could be ported onto an actual device, an emulator would be required to test the application during development. The official SUN Java Wireless Toolkit provides a generic emulator for many devices that are in the CLDC category and are based on J2ME's MIDP. The toolkit includes the emulation environment and can easily be linked to EclipseME.

MySQL would be the database system used. Not only was the existing project using a MySQL server, but MySQL also provides many key features necessary for project completion including communication via Java and PHP, along with easy administration.

Once planning and choosing the primary components was complete, setup of the existing project could begin, followed by adding the required functionality in order to meet the specification.

3.2 Setting up the initial system

Once both the MySQL server and Eclipse IDE were in place, there were a number of steps involved in getting the initial system up and running:

- The EclipseME plug-in was installed to allow development of the J2ME application in the Eclipse environment.
- Installed Sun's J2ME Wireless Toolkit (JWT) 2.2 and linked it to the EclipseME interface.
- The JDBC driver 'MySQL Connector/J' was downloaded from the www.mysql.com website, and this JAR was added to the build path of the *Server* project in Eclipse.
- A new MIDlet suite project was created in Eclipse to develop the device application and the existing source files were imported.
- The server was setup to connect to the MySQL server on *csserver.ucd.ie* with the appropriate username and password, and was then executed on the local machine. The client was configured to connect to *localhost* on the specified port. Upon execution of the client application the JWT emulator launched and the display indicated a connection to the server and a list of topics from the database appeared on the device.

It is important to note that the *Server* is implemented using Java 2 Standard Edition (J2SE) and it's purpose is simply to listen continuously for a client connection and spawn a new thread for handling each new session. This multithreaded functionality ultimately allows several clients to vote simultaneously where the server would process each one on a separate thread. The server would also perform all communications with the MySQL server in order to feed the client any necessary information and to also manipulate the database, thus registering the vote.

The *Client* on the other hand, is implemented using Java 2 Micro Edition (J2ME) technologies. Once the application has been installed, upon execution a GPRS connection is made to the server and the session begins. Initially the client receives a list of topics from the server, the user selects a topic, a response containing a list of possible choices is displayed on the client device, and finally, once a choice has been made, the current vote statistics are displayed on the device. The user is then free to exit the application or restart and vote on another topic.

The MySQL server initially contained two tables in the project database, namely *topic* and *topicAnswers*. At this stage in the project the <u>topic</u> table consisted of fields: *idNum* (integer) and *topic* (varchar). The <u>topicAnswers</u> table had four fields: *topicAnswers* (varchar), *idNum* (integer), *topicId* (integer) and *registerAns* (integer).

The server would retrieve the *idNum* and *topic* fields and send this information to the client; once the client chose a topic, the server would query the topicAnswers table to retrieve the *topicAnswers*, and *IdNum* fields that corresponded to the chosen topic ID. As soon as the client voted, the server would execute an update, which incremented the *registerAns* field for the specified answer ID.

This whole operation is performed very quickly due to the efficiency and performance of MySQL. A quick look at the current system reveals a simple diagram of components.

3.2.1 The overall system design

Taking a high level look at the system, we can depict very basically how the overall architecture will now look.

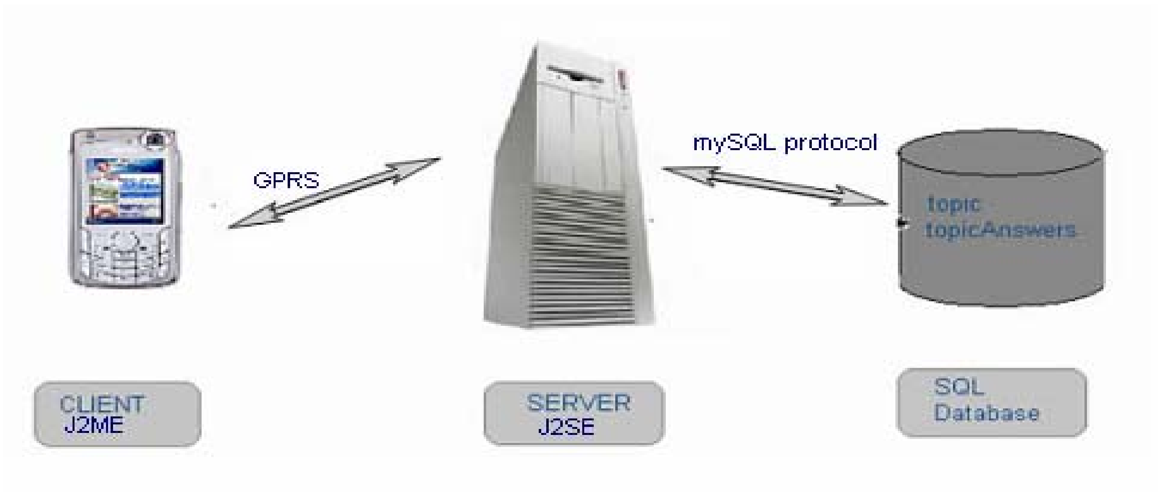


Figure 6. The three-tier system design

Figure 6, shows very simply the overall system design. The client communicates with the server and the server communicates with both the client and SQL server. The SQL server contains a *voting* database within which two tables are defined, namely *topic* and *topicAnswers*. When the client initiates a connection with the server, the server interrogates the database and returns the appropriate data to the client.

Once the initial system was up and running, other tasks could be tackled according to the specification. Firstly though, a good, flexible and well-coded base needed to be ensured, on which the extra features would be built.

3.3 Refactoring the existing code

Some major improvements and refinements to the original code and general system included:

- Changing how the client and server communicated at a low level. Originally the system required the developer to write strings or integers, character by character between the two endpoints. This was achieved using *InputStream* and *OutputStream* classes. To read data from the client *StringBuffer*s were used and data was appended piece by piece until finished. To write an integer to the client it needed to be first converted to a char array, sent to the client and then reassembled at that end. This was very monotonous, repetitive code, and resulted in unnecessary parsing at both endpoints and a high lack of readability. The new method of communication would involve *DataInputSream* and *DataOutputStream* classes which allowed string and integers to be sent and received using *readInt()*, *writeInt()*, *readUTF()* and *writeUTF()* functions respectively. Readability of code was undoubtedly improved and the system seemed far more efficient than before. This would also allow integration of graphics into the system by sending the image in bytes to the client.

- Some unnecessary classes were removed (such as a *Sender* class, implemented to specifically send messages to the client was not required), and the class to handle the client session (the thread) was separated from the actual listening class. This again made the code that bit more readable.

- At each endpoint of the communication, sessions were set up to loop infinitely until the user terminated the connection. i.e. so that the user could vote again on another topic.

- On the MySQL database, some minor changes and removal of fields was undertaken but in general the system was left intact.

3.4 Start / Expiration dates

From both the administrator’s and the user’s point of view, it made sense to add functionality so that only topics that were currently active could be voted on, and consequently displayed on the client device.

The first step would involve adding six new integer fields to the *topic* table: one for start day, start month, start year, and three others similarly for the expiration date.

When the server queries the database for the list of topics, these six fields will also be retrieved and two new *Calendar* objects will be created, one for the start date and one for the expiration date:

```
Calendar retStDate = Calendar.getInstance();
retStDate.set(stYear,stMonth,stDay);
```

These dates are then be compared to the current date, and only if the current date is between the start date and expiration date, will the topic be transmitted to the client device:

```
Calendar c = Calendar.getInstance(); // Returns the current date
if((c.compareTo(retStDate) > 0) && (c.compareTo(retEndDate) < 0)) { ... }
```

3.5 Adding Graphics

Adding graphics to the system was one of the prime concerns. One of the serious limitations of current SMS based voting systems is the lack of user-friendliness and attractive interface. Providing this rich graphical environment to the client would prove to be quite a difficult task, primarily due to limitations in the MIDP API itself.

The primary steps in adding this graphic functionality were:

- Editing the topic table, via the MySQL command-line interface, by adding a new field, *image*, of type BLOB[1] (which could store an image in the database), and adding field *ansimage* to the topicAnswers table to allow images to be associated with choices also.

- Retrieving these images, in bytes, from the database and consequently sending via *DataOutputStream* to the client.

- Constructing a new Image object on the client from the raw byte data:

  ```
  Image[] images = new Image[num];
  ......
  images[i] = Image.createImage(imagedata,0,length);
  ```

  where *imagedata* is the image represented in bytes.

3.5.1 J2ME and Graphics

The existing system used the *ChoiceGroup* UI item to display the list of topics on the client device. The ChoiceGroup constructor could accept an array of images as argument and render these on the device, but these images proved to be far smaller than required, and after searching java forums and sifting through the MIDP 2.0 specification it was clear that the image dimensions could not be adjusted.

The other options included use of *ImageItem*s, and using a *Canvas* to try drawing the graphics manually. Using *ImageItems*, the images could be displayed on the screen but it was very awkward to resize and reposition the images and text in the right manner. Drawing graphics via a *Canvas* would involve re-thinking the whole structure of the display, would require far too much time. Both approached proved unfeasible.

[1] BLOB or Binary Large Object, a field type in MySQL for storing data of variable size.

3.5.2 Using J2ME Polish

I discovered the J2ME Polish tool (www.j2mepolish.org) while trying to find a solution to the graphics problem. Its main functions are detailed in the background section.

It is already mentioned that J2ME Polish offers "a powerful GUI which allows you to design the interface with simple CSS text files outside the source code". It uses Apache Ant to build files (http://ant.apache.org/). Again, there were a number of steps involved in the integration of J2ME Polish into my existing MIDlet suite:

- After installation, copying of the '*build.xml*' file and the '*resources*' folder from the *sample* directory, into the project root.
- Any resources needed to be then moved in the '*resources*' folder of the project. (Subfolders could not be used since these were used for the automatic *resource assembling* of J2ME Polish).
- Adjusting of the '*build.xml*' file: specifying the MIDlet class (*ClientMIDlet*) in the <midlet> element, and temporarily adjusting the <deviceRequirements> element to "Generic/midp2" (One of the four device groups, specified so that the application built would be suitable for the MIDP 2.0 emulator).
- Ensure the "*usePolishGui*" attribute of the <build> element is set to "true".
- Added '*tools.jar*' from the lib directory of the JDK installation, to the Ant classpath ('Window > Preferences > Ant > Runtime' in Eclipse) to provide the necessary libraries for Ant buildfiles.

Once the system was fully integrated, 'Ant' could be called within Eclipse on the 'build.xml' file to build the application and effectively deploy the Jad & Jar files. These files could then be executed on the external JWT emulator for testing.

In order to apply the desired look and style to the user interface, J2ME Polish provides a 'polish.css' file in the resources folder that is completely separate from the source code. This cascading style-sheet contains predefined styles, which influence the look of several MIDP items. For example, the style used for highlighting the currently specified item in *ChoiceGroup*s, is defined as:

```
  focused {
    padding: 3;
    background {
            type: round-rect;
            arc: 8;
            color: bgColor;
            border-color: black;
            border-width: 2;
    }
    font {
            style: bold;
            color: white;
            size: medium;
    }
    layout: expand | left;
}
```

Other types of styles that may be defined are static and dynamic. If a static style is defined in the *polish.css* file then it may be defined in the source code of the application with the #style preprocessing directive by placing it before the constructor of the target item. For example:

```
//#style menuItem
si = new StringItem("Status:", " ");
```

sets the style of the *StringItem si* to that of menuItem which is defined in 'polish.css'.

The result of applying J2ME polish can be more clearly seen in the Screenshots section.

3.6 SMS functionality

Already mentioned is the option to charge users for certain votes, if deemed appropriate. This does *not* simulate current voting systems like the 'You're a Star' system. Rather, when the user selects a choice on the device, instead of the client initiating a direct GPRS connection to the server, the chosen answer ID is sent via SMS to another phone, and this phone in turn will connect to the server via GPRS in order to register the vote. Since access to a proper SMSC centre was not possible for the project, this approach simulated it by using another phone to 'relay' the vote, thus proving that it is possible for the client to vote via SMS.

- The initial step involved adding two new fields to the topic table – *sms* and *smsnum*. The *sms* field acted as a flag, either set to 1 or 0. It indicated whether the topic was an 'SMS vote' or not. If it was set, then the number in the *smsnum* field was the mobile number of the phone running the relay application.

- Once the client device, which initiates the voting application, receives the value of the *sms* field and discovers it is set, it communicates as normal with the server *except* instead of sending the answer ID to the server it is sent via SMS to the phone number specified. The client then continues to receive vote statistics.

- A new MIDlet suite project was created, *SMSRelay*. The sole purpose of this MIDlet application was, once installed on the client device, to listen continually for incoming SMS messages. It needed to use libraries provided by the Wireless Messaging API (WMA) specification (JSR120):

  ```
  import javax.wireless.messaging.*;
  ```

  Once an answer ID is received, a new GPRS connection is made to the server and the message payload (the ID) is sent via TCP to the server.

- The server would now need the ability to handle more than one type of session, one of type client, and another which will be the SMS relay. On connection, either one will send an integer, or session ID, to the server so that the server can perform the required action. On receiving a connection request from the SMS Relay, the server will simply register the chosen answer, and inform the relay of vote success.

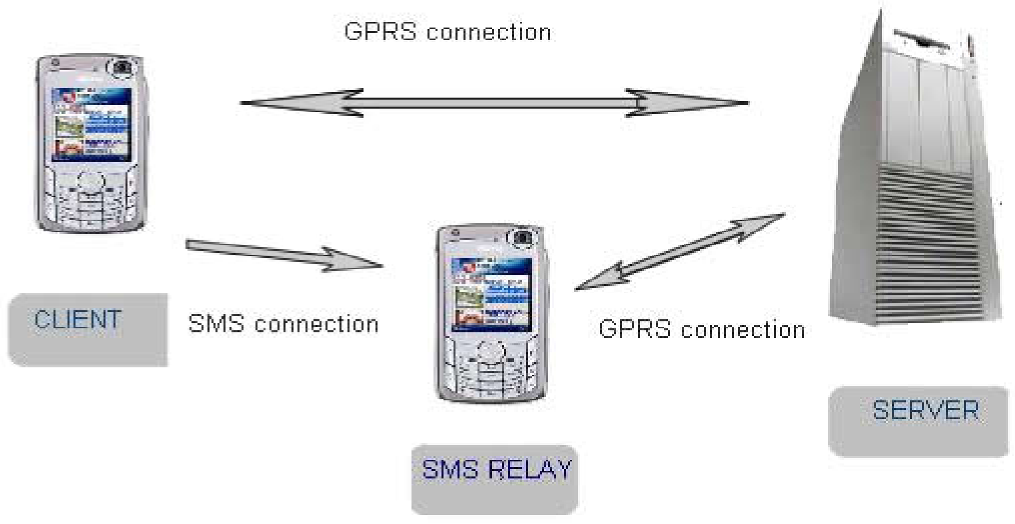


Figure 7. The system including the SMS relay

3.7 Deploying and testing on a mobile device

In order to deploy the Client MIDlet, an Ant build is performed on the J2ME Polish *build.xml* file. The two files of type JAD (Java Application Descriptor) and JAR (Java Archive) are automatically generated into the 'dist' folder of the project. In order to port this application to a mobile device, a Siemens CX65 in this case (specified in the <deviceRequirements> element of the *build.xml* file), it could either be copied via data cables or more efficiently and effectively using OTA provisioning.

3.7.1 OTA Provisioning

Over-the-air provisioning, as described in the background chapter, is a method of deploying your MIDlet application over GPRS/WAP. In order to set up OTA for the MIDlets:

- Copy the JAD and JAR files that were generated for each suite, to a public web server (csserver.ucd.ie). Create a new html file with links pointing to the location of each jad file.

- Specify MIME types for the server. Creation of a .htaccess file in the same directory as the jad & jar files with contents:

  ```
  AddType text/vnd.sun.j2me.app-descriptor jad
  AddType application/java-archive jar
  ```

  This provided the information necessary to the web server about the file types.

- On the client device, set the browser URL to the path of the html file. Once the correct link is selected the device AMS will download and install the MIDlet, which is now ready for execution.

This process allowed both the client and relay MIDlets to be downloaded and installed onto a mobile device quickly and efficiently.

3.8 Web based administration

An administration interface via the web would undoubtedly benefit both the user and the administrator. The user would have the ability to view current vote statistics at any time and keep track of preferred vote topics, while the administrator could easily add/remove/edit votes in the system via any web browser.

Each interface would be coded in PHP/HTML, in particular taking advantage of MySQL functions provided by PHP. The computer science web server (http://csserver.ucd.ie/~s02bf067/web) would host these pages, and required an upgrade of PHP to support MySQL functions.

3.8.1 Statistics interface

The idea of this page is to display current vote statistics along with images, retrieved from the MySQL database. Each topic is displayed in a HTML table, along with associated choices, statistics and images.

The primary PHP functions used are the result sets, and the process of looping through this sets while building the table rows on the fly:

```
$result = mysql_query("SELECT idNum, topic, image FROM topic");
while($row = mysql_fetch_array($result)) {………}
```

Within the `while` loop a new table row was created with the topic string and image. An inner `while` loop traversed thorough all choices for the current topic and printed these statistics and images as new rows.

3.8.2 Add / Edit / Remove a topic

These three interfaces would be implemented as separate pages. In order to add a new topic, form fields are filled with the required information and once the update button is clicked, these values are sent to the PHP processing script via HTTP POST method and an SQL ‘insert’ statement inputs the data into the appropriate tables.

The ‘edit’ interface, provided a simple dropdown menu of all topics in the database and a form whose fields were loaded with values for the chosen topic. These fields could then be edited at will, and once the update button was pressed, SQL update and insert statements would edit the data in the database to match the changes.

Removing a topic is a simple operation. The user is shown a table of all topics currently in the database, enters the ID of the topic he/she wishes to delete and presses the ‘remove’ button. The SQL commands:

```
mysql_query("DELETE FROM topic WHERE idNum=$id");
mysql_query("DELETE FROM topicAnswers WHERE topicId=$id");
```

achieve this.

3.9 Security functions

3.9.1 Web management authentication

In order to secure the administration pages for adding, editing and removing topics from normal users, a web based authentication system was put in place. To password protect the target directory .htaccess and .htpasswd files were created. The .htaccess file is placed at the root of the folder you wish to protect. It looks like:

```
AuthName liam
AuthType Basic
require valid-user
AuthUserFile /home/2002/s02bf067/public_html/web/admin/.htpasswd
```

The first couple of lines are self explanatory, and the last line specifies the full path to the .htpasswd file that contains:

```
liam:xs7c7IlWE5QXw
```

The second bit is an encrypted form of the password. This file is generated, along with the hash by using the command:

```
htpasswd [-c] .htpasswd liam
```

where the '-c' option creates a new file. When a user tries to access the protected directory from their web browser, a prompt to enter authentication details is displayed. Unless the username and password matches that of the .htpasswd file, access is forbidden.

3.9.2 Preventing multiple votes

Intuitively, users should only be allowed to vote <u>once</u> on a topic by default. Multiple voting should only be permitted in circumstances where, for example, the user is charged to perform the vote.

- A new table *hostlist* was created in the database, which contained simply two fields: a topicID and host field, which would effectively store (ID, host) pairs. The host represents the host address of the connected client, and the ID is the topic which he/she has already voted.

- When a client attempts to vote on a particular topic, the server checks the database to discover if the client's host address already exists along with the topic ID. If it does exist, the client device alerts the user that he/she has already voted and should choose another topic. On the other hand, if no match is found then the user is allowed to vote, but once the vote has been registered a new entry is inserted into the *hostlist* table corresponding to that of the client's host address and the topic ID on which was voted on.

The administrator also has full control over this system via the web management interface. Hosts may be removed from the list via topic ID (thus clearing all hosts for that particular topic), or alternatively all entries may be removed at once.

4 Detailed Design & Implementation

4.1 UML Class Diagrams

In UML (Unified Modelling Language), a class diagram is one that shows the structure of a software system by modelling its classes and relationships between them [15].

4.1.1 Server - J2SE Classes

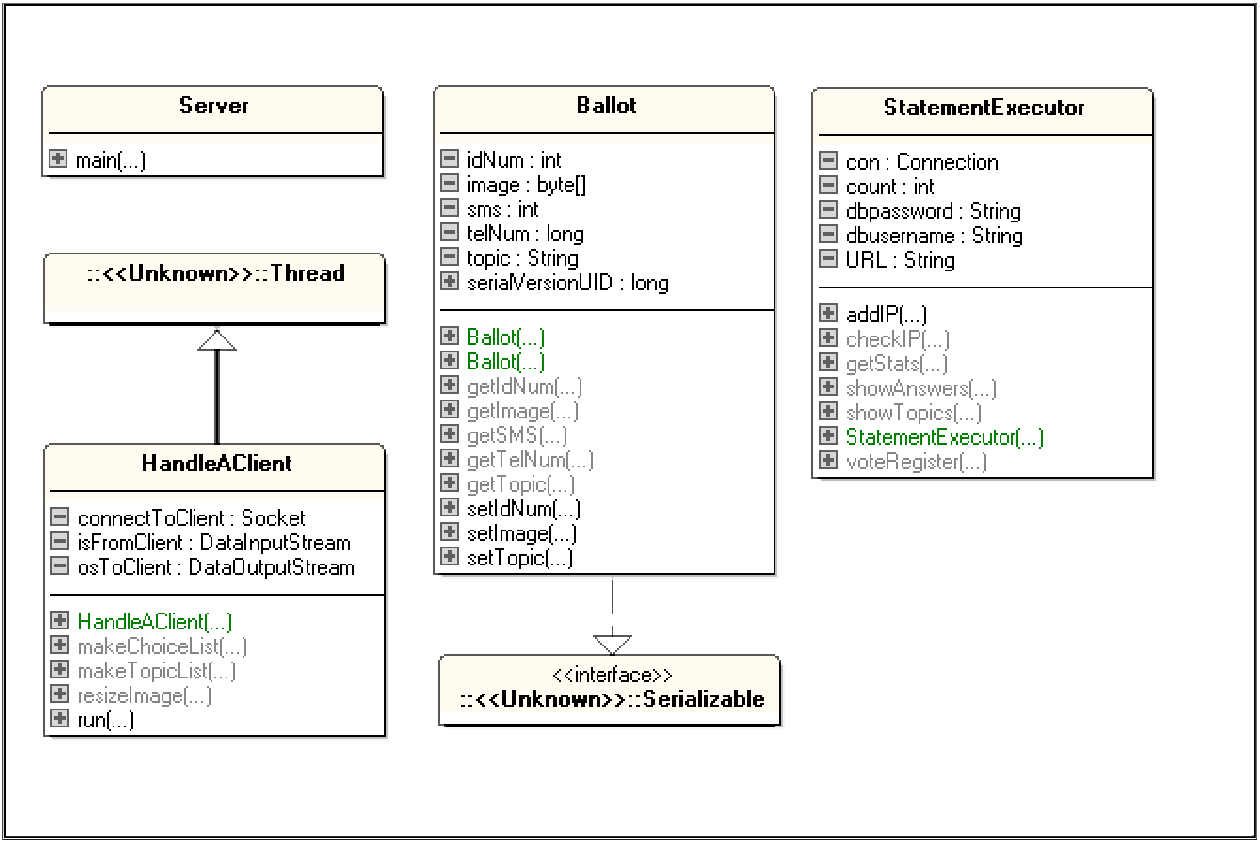


Figure 8. UML Class model of the server classes

The server side classes, once developed, were uploaded to *csserver.ucd.ie* and executed on the remote server via SSH. There are four primary classes as evident from the diagram: The *Server*'s purpose is to listen on the specified port, and spawn a new thread, *HandleAClient*, when a new connection is received. The *StatementExecutor* utility class handles all connections and queries to the MySQL database and performs the actual updating of tables, for example incrementing the *registerAns* field in order to register a vote.

Instances of the *Ballot* class encapsulate data for each topic. Each *Ballot* object contains all information pertaining to the vote. Information on a particular vote may be then retrieved or modified by the server using the private methods in the *Ballot* class. This kind of encapsulation takes advantage of the object-oriented paradigm and improves readability and code structure.

4.1.2 Client – J2ME MIDP classes

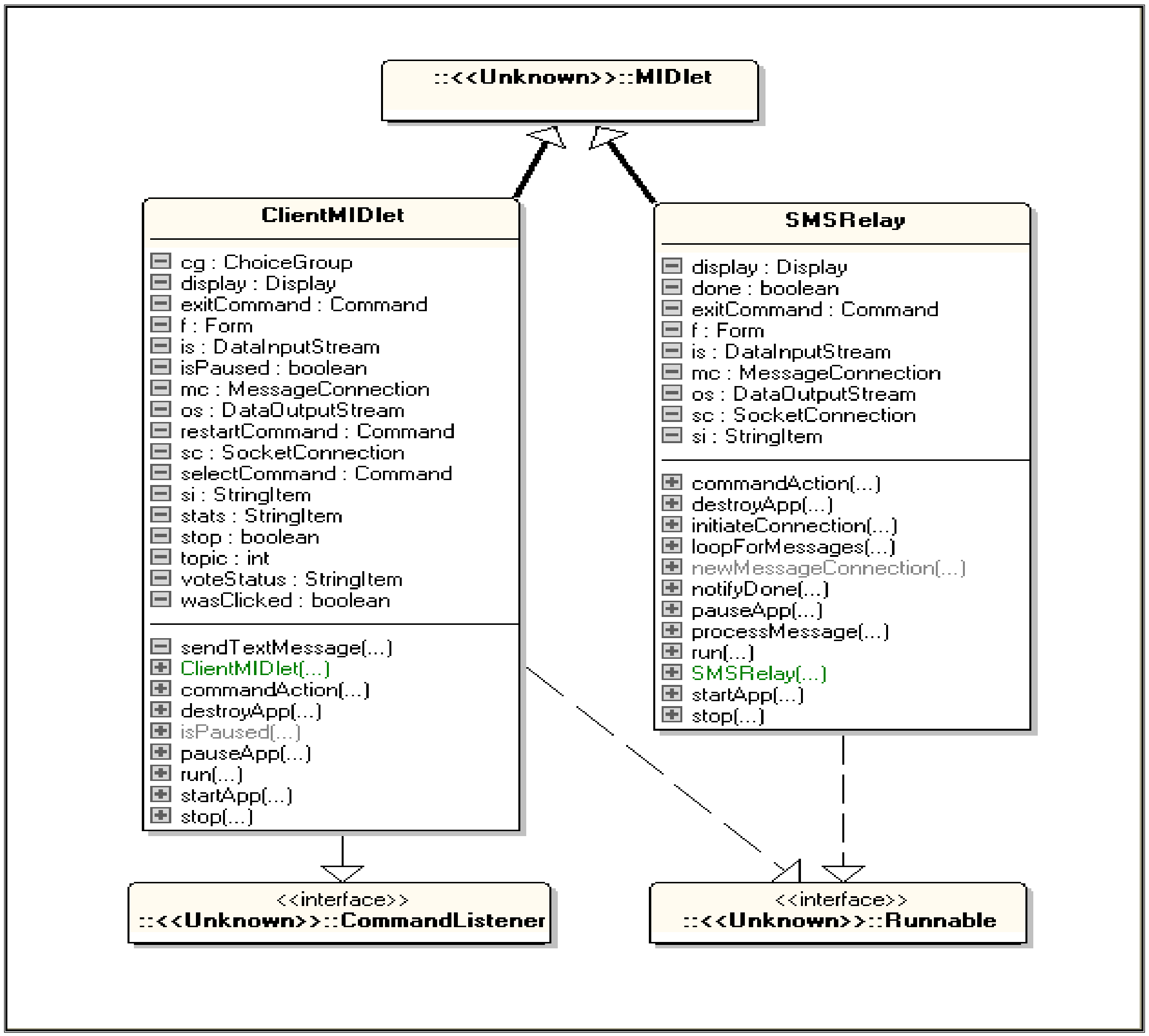


Figure 9. UML Class model of the client classes

Two independent MIDlets are defined here. However they are both similar in structure: each extends the default MIDlet class, and implements the *Runnable* interface. They are both client applications and have defined life cycles.

The *ClientMIDlet* class is the primary one. Once deployed and executed on the client device, it will negotiate a TCP connection to the server, and allow the user to perform a vote in the rich graphical environment provided by J2ME Polish. Although it is not visible in the diagram, J2ME Polish, as already described, was vital to the graphical success of the project.

The *SMSRelay* MIDlet, once executed on the device, will listen infinitely on the *MessageConnection.* Once an SMS is received from the primary client device, it extracts the appropriate ID from the message, and connects via GPRS to the server in order to register the vote.

4.2 UML Activity Diagram

The process and protocol of performing a standard vote is described in the diagram below. The voting procedure involves three entities: The user, the client application running on the user device, and the server.

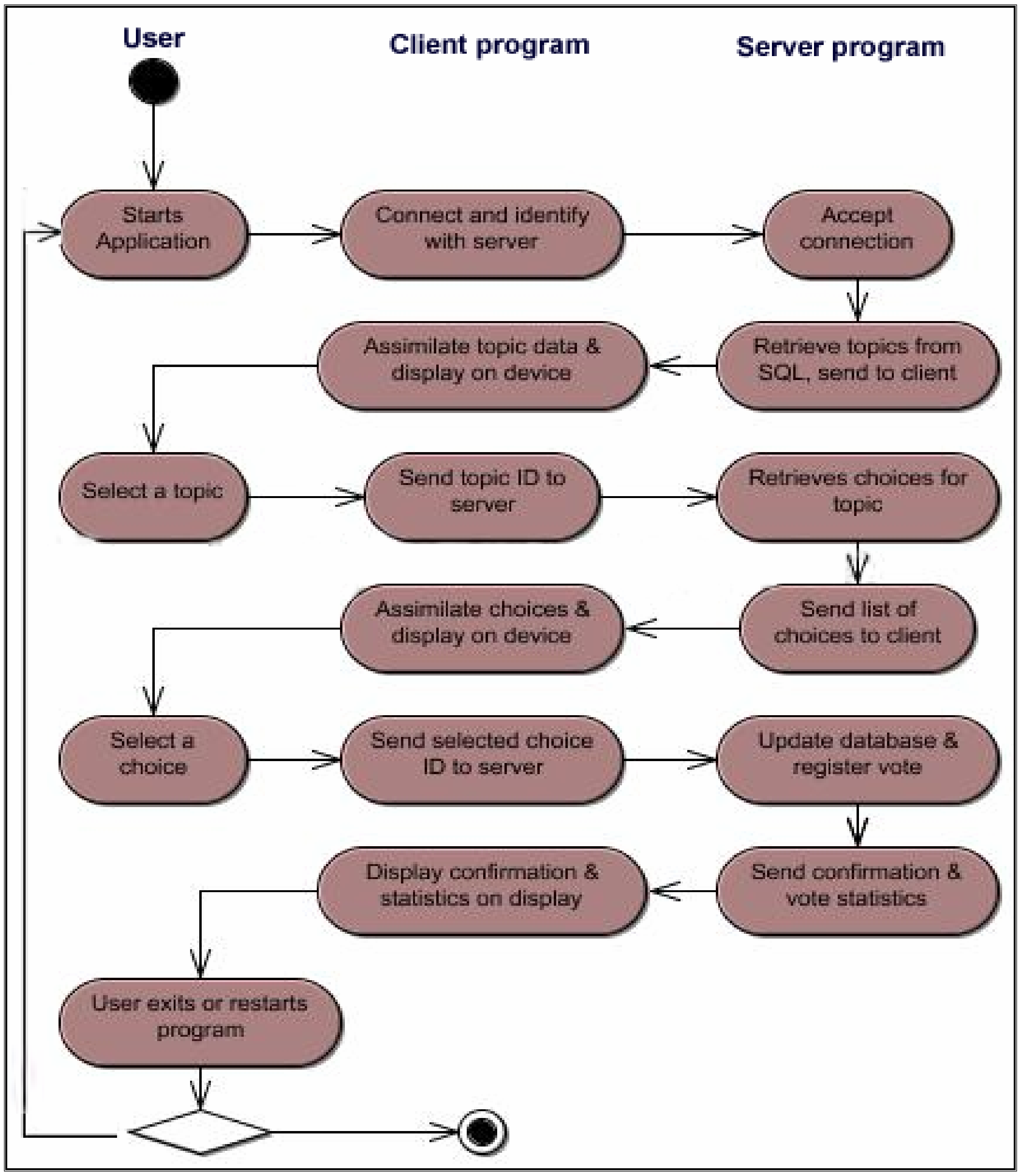
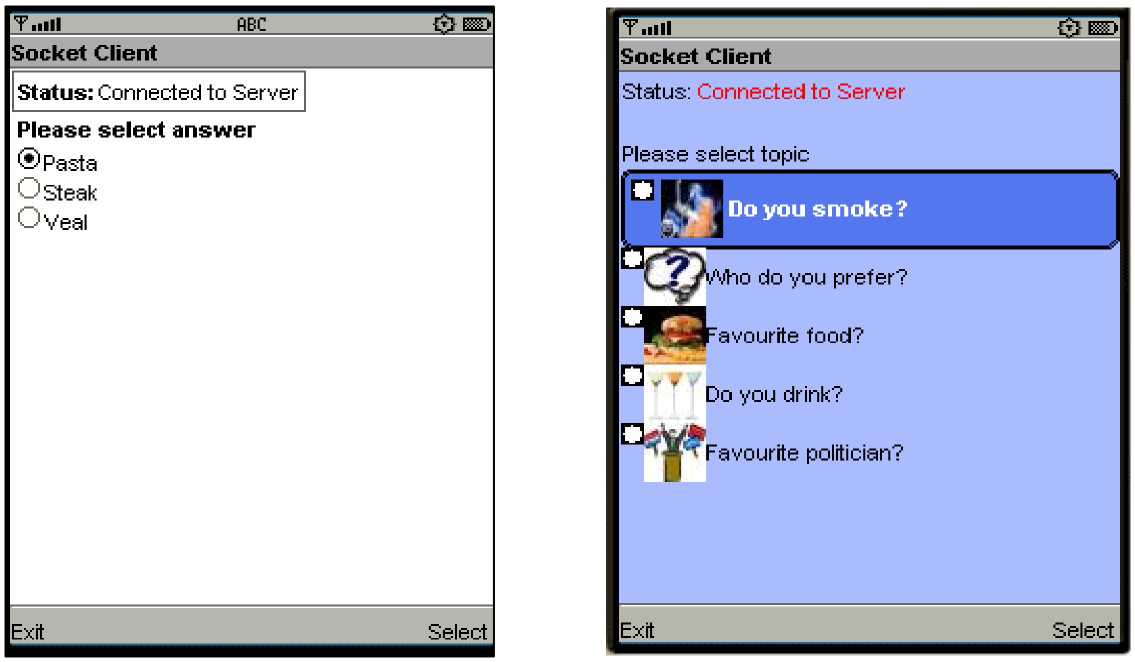

User
Client program
Server program
Starts Application
Connect and identify with server
Accept connection
Retrieve topics from SQL, send to client
Assimilate topic data & display on device
Select a topic
Send topic ID to server
Retrieves choices for topic
Send list of choices to client
Assimilate choices & display on device
Select a choice
Send selected choice ID to server
Update database & register vote
Send confirmation & vote statistics
Display confirmation & statistics on display
User exits or restarts program

Figure 10. The UML activity diagram [15]

5 Evaluation & Analysis

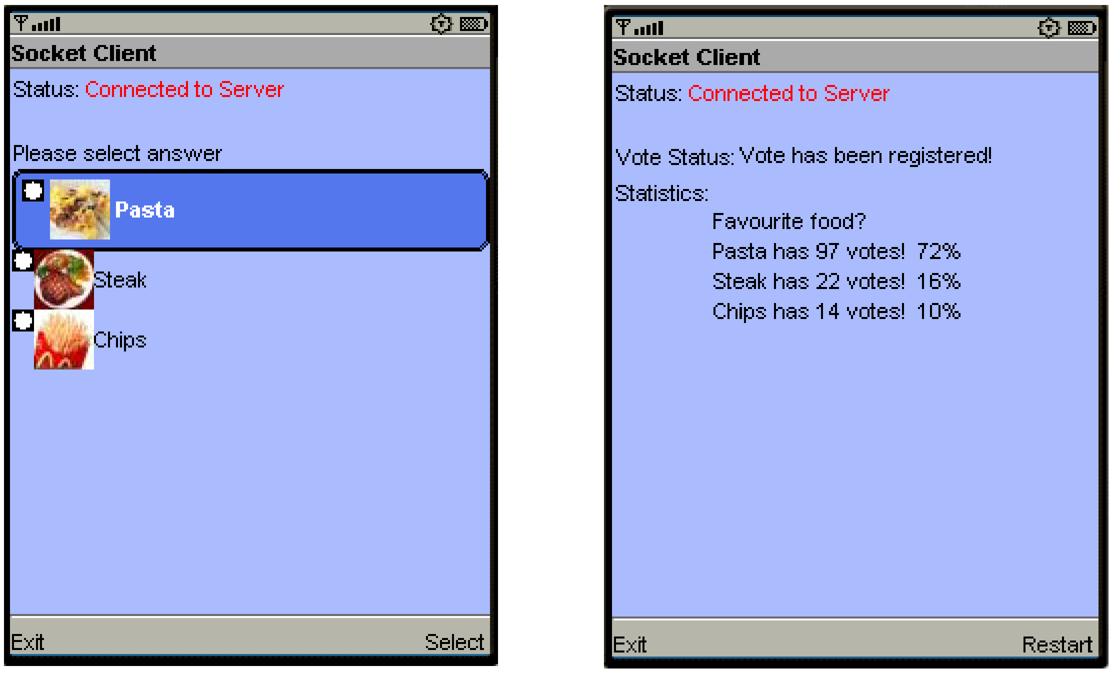
5.1 Screenshots of completed system

In order to understand the motivation behind, and requirement to develop a better and more visually appealing user interface, screenshots of the device display from different stages of development are shown, along with one from the web interface.


ABC
Socket Client
Status: Connected to Server
Please select answer
● Pasta
○ Steak
○ Veal
Exit Select

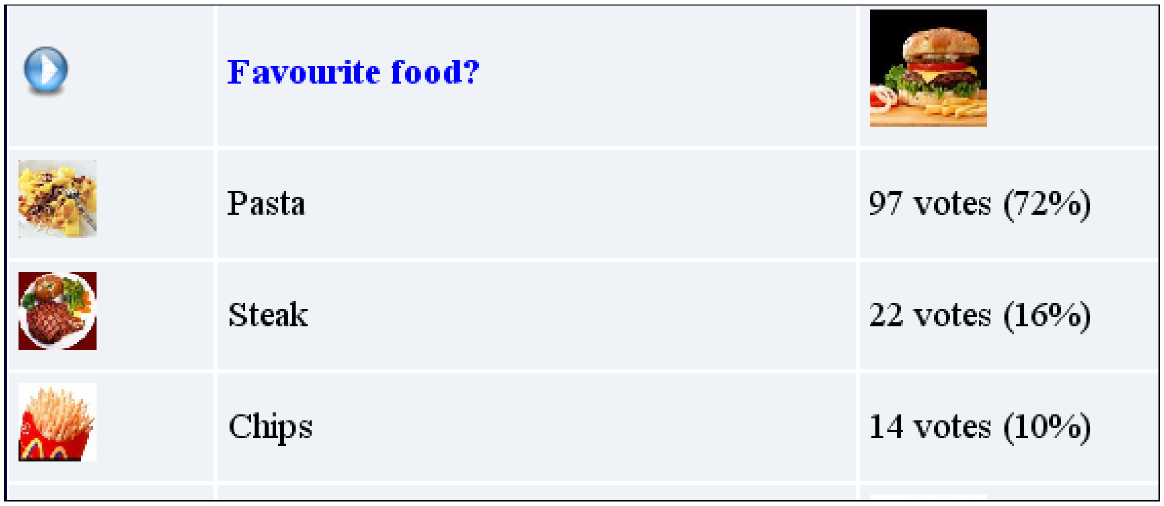
Socket Client
Status: Connected to Server
Please select topic
Do you smoke?
Who do you prefer?
Favourite food?
Do you drink?
Favourite politician?
Exit Select

Figure 11. User Interface: Before and After


Socket Client
Status: Connected to Server
Please select answer
Pasta
Steak
Chips
Exit Select

Socket Client
Status: Connected to Server
Vote Status: Vote has been registered!
Statistics:
Favourite food?
Pasta has 97 votes! 72%
Steak has 22 votes! 16%
Chips has 14 votes! 10%
Exit Restart

Figure 12. Voting screens.



	Favourite food?	
	Pasta	97 votes (72%)
	Steak	22 votes (16%)
	Chips	14 votes (10%)

Figure 13. The web interface displaying vote statistics.

5.2 Progress Analysis

Figure 13 below shows a simple timeline of project progress. Important dates are included, as well as various tasks that have been completed in order to meet the specification.

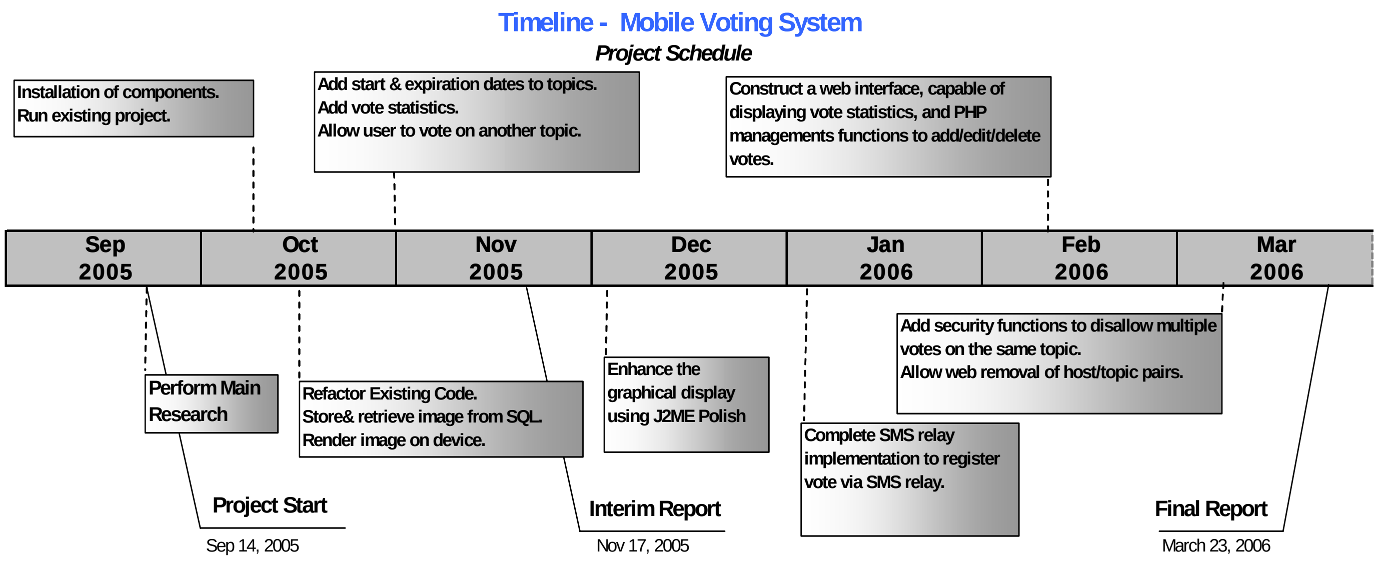

Timeline - Mobile Voting System
Project Schedule
Installation of components. Run existing project.
Add start & expiration dates to topics. Add vote statistics. Allow user to vote on another topic.
Construct a web interface, capable of displaying vote statistics, and PHP managements functions to add/edit/delete votes.
Sep 2005
Oct 2005
Nov 2005
Dec 2005
Jan 2006
Feb 2006
Mar 2006
Perform Main Research
Refactor Existing Code. Store& retrieve image from SQL. Render image on device.
Enhance the graphical display using J2ME Polish
Add security functions to disallow multiple votes on the same topic. Allow web removal of host/topic pairs.
Complete SMS relay implementation to register vote via SMS relay.
Project Start
Sep 14, 2005
Interim Report
Nov 17, 2005
Final Report
March 23, 2006

Figure 14. Project timeline

5.3 Testing with multiple clients

In order to fully test the robustness and reliability of the system a number of automated and manual test cases were performed. The most beneficial method of testing such a configuration is to rigorously examine the true multithreaded capabilities & stability of the server, by running multiple clients simultaneously. Of course, in reality such a system may have to handle thousands of such connections at any one time.

Due to the obvious difficulty in acquiring and installing the application onto numerous actual devices, emulators were preferred. The server and MySQL would still execute and listen on csserver.ucd.ie. A test class was written to initially construct twelve such instances of the client, and the client MIDlet class would be automated to randomly choose from the list of topics and choices to provide an instant response to the server.

In conclusion, the server responded well to the high load and performed the multithreaded capabilities as expected. No node failed in the process and votes were successfully registered. Since the server was now deemed stable and robust to multiple client connections, the next vital component to test was the server response speeds or latency.

5.4 Server response latency

The efficiency of the program goes hand in hand with the network latency and response delay. There are undoubtedly a number of factors which influence this delay; server load, network congestion, type and speed of connection, and of course the efficiency of the code itself.

The delay measured was the time taken for the client to send the answer ID to the server, the server to connect to the database and register the vote, and finally report to the client whether it was successful or not.

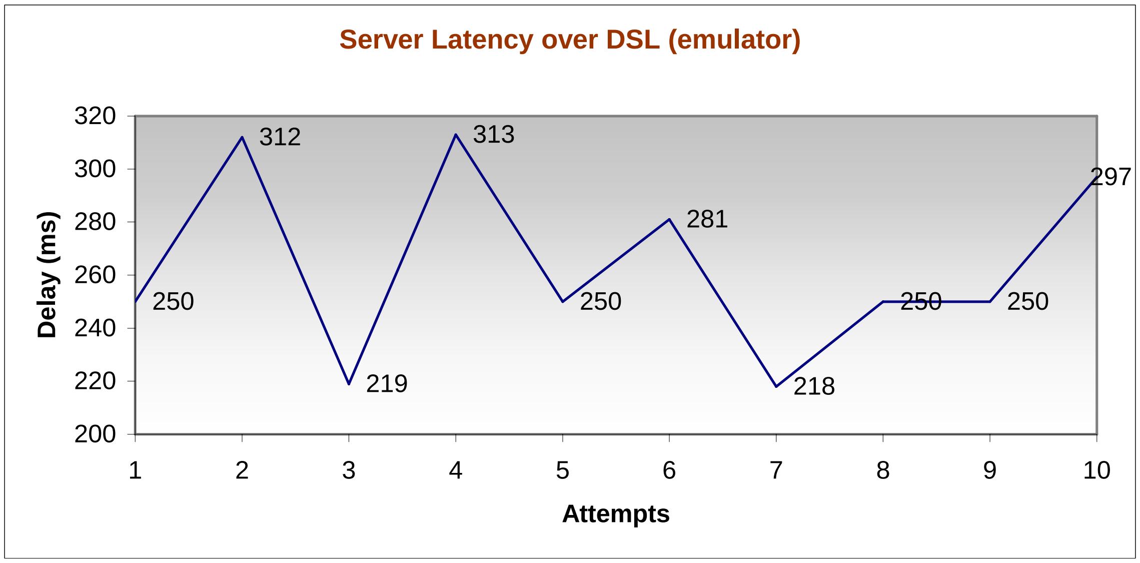


Figure 15. Latency of server using emulator (broadband connection)

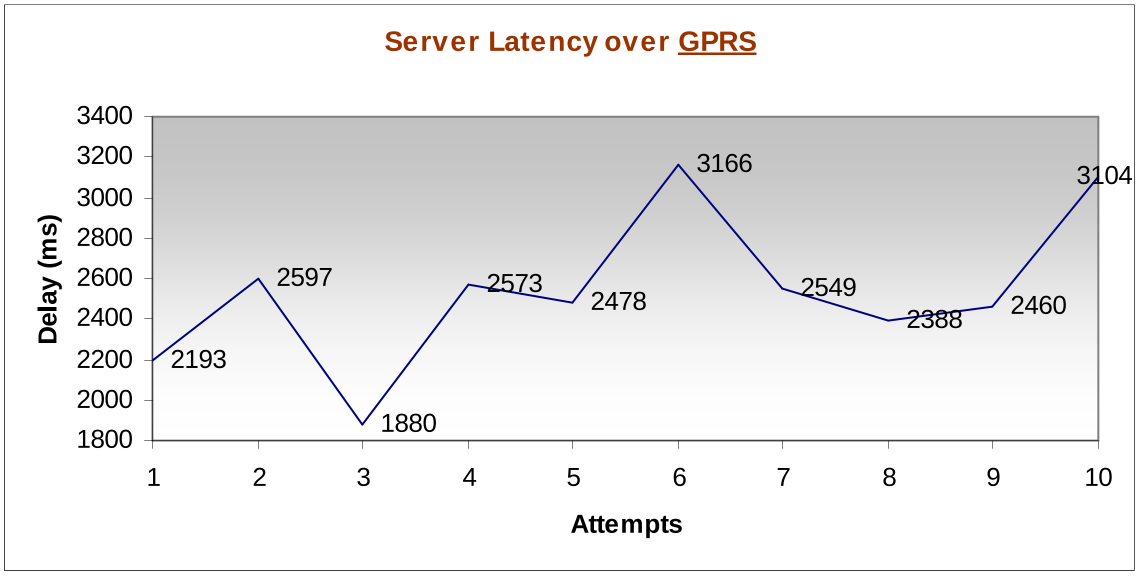


Figure 16. Latency of server using actual device (GPRS)

From Figure 14, it is apparent that the server does in fact register the vote very efficiently provided the network connection itself is of reasonable speed. Using an emulator on the PC, the vote is registered and confirmation is displayed on the screen with an average speed of 264ms (approximately one quarter of a second) over ten tries.

However, testing the response times on an actual device operating over O2 GPRS, we get an entirely different set of results. Figure 15 shows that the average speed of performing the action is quite significantly slower over GPRS, at approximately two and a half seconds, or 2539ms. This kind of latency is quite noticeable from a user's point of view and shows how current speeds and reliability of GPRS can provide some serious limitations for networked applications, as is discussed in the next chapter.

6 Conclusions and Future work

Overall the project was a success. The features developed throughout the project met the project specification and more. Users can vote in a rich graphical environment, securely and reliably, and track topics via the web interface. The primary requirements of both the administrator and the client user have been implemented, but of course the system has its limitations.

On project completion there were a number of conclusions to be drawn, both on the achievements and limitations of the system itself, but also regarding future work and extensions.

6.1 Scalability

Although the project was initially considered a small scale one, J2ME Polish in particular provides features that make it scalable in terms of devices on the market. It easily allows the MIDlet application to be built for any number of devices in its database. Each device build, will produce a more specialised application for the screen-size, capabilities etc. of that device. During the project, for example, a separate application was built for the Siemens C65 and CX65 devices which were provided by O2 Ireland. This could easily be extended to hundreds of unique vendor devices as discussed in the future work section.

6.2 Limitations

6.2.1 GPRS reliability

The major limitation behind the GPRS approach is in fact the speed of GPRS itself, at least in Ireland. It is clear from section 5.4 that there is a significant difference in speed and performance between testing the system over DSL and GPRS. This is especially true when transferring images (even after resizing) over the network. There are also some minor issues concerning reliability of GPRS, where it was noticed on few occasions that connection dropped for no apparent reason. This rare dropping of connection was never witnessed using emulators over DSL.

6.2.2 No ‘back’ button

Perhaps the user, after selecting a topic, may change his / her mind and wish to go back one step and choose again. Unfortunately this functionality does not exist on the current system. Once a client initiates a connection to the server, a session begins and a strict protocol is followed throughout that session. A user may of course exit the program at any time. In order to implement this ‘back’ functionality, a different approach must be taken. One possible solution is to split that one large session into a number of smaller ones. For example, a client could connect and request a list of topics as one session, then begin another session to send an ID to the server to be registered, and finally request the list of statistics. In this case, a user could use the back button and repeat a session over and over.

6.3 Future projects

With the evolution of new 3G and wireless technologies by service providers around the world, the opportunities for progression are endless. This section will briefly discuss some possible extensions and consequent work that could be carried out to improve the existing project.

6.3.1 Online voting

A nice addition to the web interface may be to implement a voting system online. The user may then have to option to either vote online or via their mobile device. For votes, whereby the user should be charged, the voting could perhaps be limited to just mobile over GPRS/SMS. This online system could be further extended to implement logins and personalization, so that if there are hundreds or thousands of topics, only topics relevant or to the user are displayed. This leads on to the next section or that of genres.

6.3.2 Genres

If the system were to grow in scale, it is possible that an enormous number of topics may exist in the database. Such a large number of topics would not only be monotonous and tedious to browse through, but extremely slow to download onto the device. The concept of arranging topics in the database according to genres, and allowing the user to select a genre at the beginning of the session would result in only relevant votes being displayed on the client device, and a more efficient response from the server.

6.3.3 Deploying for more devices

As already mentioned, the current client application can easily be built for any number of existing mobile devices. It the project became popular, it would be necessary to deploy a specific build for every major vendor and model. These specialised applications could then be uploaded to the web server, and an interface created to allow users to browse through the available builds and install one that is suitable for their device.

7 References

- [1] http://www.gsmworld.com
  Information on evolving wireless communications technology.

- [2] http://aeont.com/Technology-GPRS-en.aspx
  Detailed content covering GPRS, SMS and circuit-switched data.

- [3] http://java.sun.com/docs/books/tutorial/networking/overview/networking.html
  Client server networking basics.

- [4] http://java.sun.com/docs/books/tutorial/networking/sockets/index.html
  Introduction to TCP and UDP sockets.

- [5] http://java.sun.com/index.jsp
  Provides detailed information on all three Java 2 technologies.

- [6] http://www.jcp.org/
  The official Sun website or the Java Community Process also listing all JSRs.

- [7] http://java.sun.com/j2me/overview.html
  More specific learning guides for J2ME.

- [8] http://java.sun.com/products/cldc/overview.html
  Specific information concerning the CLDC configuration.

- [9] http://www.onjava.com/pub/a/onjava/2002/12/18/midp.html
  Background information on the MIDP profile.

- [10] http://java.sun.com/products/midp/OTAProvisioning-1.0.pdf
  Paper describing Over the Air Provisioning for MIDlets.

- [11] http://dev.mysql.com/doc/refman/5.1/en/index.html
  MySQL 5.1 official reference manual.

- [12] http://java.sun.com/products/jdbc/
  Information regarding JDBC drivers.

- [13] http://www.mobilegd.com/article77.html
  On installing and setting up J2ME Polish.

- [14] http://www.php-mysql-tutorial.com
  Tutorial on how to use PHP to connect to MySQL databases.

- [15] http://www.agilemodeling.com/essays/umlDiagrams.htm
  Describes various UML 2.0 diagrams.