UNIVERSITY OF NOTTINGHAM

G52GRP

Democratic Conferencing Tool

Interim Report

gp09-sdb
December 2009

Group:	gp09-sdb
Supervisor:	Dr. Steven Benford
Group Members:	Robert Golding, William Redrup, Tammie Seo, Christopher Lensvelt, Henry James, Zhongda Zhu

Table of Contents

1 Executive Summary

This document describes in detail the group's progress thus far in both designing and developing a text-based Democratic Conferencing Tool. The initial section of the report focused on introducing the problem, highlighting the key aspects. After expanding upon the initial project description, the section gave us a clear insight into what was required of us as a group, and helped us decide where our key focus should lie. We did some research into existing systems and the technologies available to us, and after some discussion, we arrived at a number of conclusions.

Firstly, we would develop a web-based application using the Django web-framework after deliberating about whether Desktop-based or web-based would prove most advantageous for our system. After a section of research regarding how we would store and share our work, we arrived at the conclusion that the project could be hosted on GoogleCode.

We then decided to develop a prototype system which would include most of the core features we had decided upon, keeping the prototype basic though functional. This not only helped design, but also acted as a feasibility study into the Django web-framework we had decided upon. The prototype was deemed successful after a test-plan was drawn up, allowing us to check functionality of the system.

After creating a working prototype and testing its features, we then focused on the functional specification. This was introduced with a brief overview of the functional aspects of the system, and was followed by two user scenarios we created to grasp an idea of the people that would potentially be users of our system, and how they could perhaps be introduced to it initially. A flow-chart to accompany these user stories was created, to display how a new user would interact with the system from first visiting the homepage to providing feedback on a particular topic. A list of key requirements, both functional and non-functional, is included in this section to help keep track of what is required when building the system.

We then looked into the problems we had encountered over the course of the project, and discussed solutions to potentially solve each of them. Some of the problems are very general, and likely encountered by each group collaborating with each other, however some are more specific to our group, regarding meetings and planning.

Finally, our project time-plan was included in the report in both table form and displayed via a Gantt chart, allowing us to keep track of our progress at all times during the project time-frame, and also highlighting particular periods where we are ahead or behind with a specific piece of documentation.

See Appendix 6 for Meeting Minutes & Attendance

2 Introduction

The purpose of this report is to provide an in-depth look at the status of our group, gp09-sdb, and contains all documentation for the work completed thus far. The structure of the report will begin at the initial problem and expand based on what we have discovered that needs addressing. Then, the research will be listed in segments based on the area the research is focusing on, i.e. market research, technical research, etc. After deciding on particular aspects of the software via our research, the functional specification shall be completed based on these decisions. In the later stages, the focus will shift to planning our system implementation, and the platforms we will be considering.

3 The Problem

Project Description: *"This project is to design, implement and evaluate a new text conferencing tool that aims to support more democratic participation. Each participant is given a "talk budget" which may be refreshed from time to time. When this runs out they can no longer talk, unless another participant donates some of their remaining budget to them."*

To clarify, our group project is to design and build a Democratic Conferencing Tool. Our primary aim when building this application is to introduce a new product to a market brimming with competition. To combat this however, we should be introducing completely new ideas and implementing them into our product while also focusing on the features that make the existing competition so popular. For this to work, it is important to carry out adequate research into every area of our project, ranging from platform suitability to utilising the best method to communicate within the group.

The key feature that was identified immediately was that this product must differ from a conventional chat tool. Simply because the primary focus is that rather than just offering a place for people to chat about random topics without structure, our application should offer users a system allowing them to join existing debates where they can share their own structured opinion on any of the subjects being discussed, or allow them to begin their own subject and open it for discussion. Either way, the primary goal is to provide a place to go for those who want to take part in more serious discussion than you find in a regular web chat room.

Secondly, another key aspect of the system is that it must be democratic, as indicated by its name. To ensure this aim remains true is vital for us as it will help set us apart from the competition mentioned earlier in the report. To ensure this is the case a feedback system should be implemented involving votes by the users within a conference room, allowing good or bad feedback to be provided about the topic discussed. It is also important to allow the listeners to abstain from voting if they don't have a positive or negative opinion on the subject, or provide a 'neutral' option.

In essence, what we hope to build is a democratic, user-friendly, and self-moderating community where opinion-sharing is not only welcome, but encouraged. Of course, the balance required to achieve this objective is extremely delicate, and many things could go wrong if, for example, poor decisions are made, it could affect the entire project. It is vital that, as a group, we get a handle on the key aspects of the system in the early stages and maintain our stance throughout, to avoid indecision at a later stage.

4 Background Information & Research

4.1 Survey of Existing Systems

We have carried out research into similar systems, in order to gather a better understanding of features that are considered ‘a must’ for this type of project, and those that could perhaps be improved upon. Two particular projects that caught the attention of our researcher were Google Wave, which unfortunately is currently limited to invitation testing only (could be considered a closed beta), and Effusia Business Messenger, a very popular tool for conferencing in the business sector. More on this can be found in the research notes.

4.2 Market Research

4.2.1 Google Wave

Wave is without doubt one of the most innovative communication software packages on the internet. It is effectively a collaboration tool; its aim is to allow users to communication and work together on what is more or less a whiteboard on which everyone can work. Here are the key features:

- Real-Time – Almost a prerequisite for communication software, Wave runs in real time. This allows for fast-paced collaboration and communication between co-workers.
- “Converdocuments” – the Wave interface offers users a unique position; they can both communicate and collaborate on the same document, thus allowing free reign to work together without restriction.
- Email-like structure – Waves are live, but once they are no longer being used they are stored in an email like fashion. Simply go to your “inbox” and you have access to all your collaborated efforts from the past.
- Meeting notes – Wave facilitates the use of “Meeting notes”, effectively real time minutes updates which allow total meeting organisation.
- Open Source – Google Wave is entirely open source. This allows users to nurture Wave into something far more developed that what it started out as, and also encourages innovation; something very important in a world full of budding programmers!
- Ability to Embed – Any Wave can be embedded in another web page very simply. This allows users to incorporate the usage of Wave into other projects that are already running, or need to use some other tool.
- Many more innovative features! – This list represents features that seem relevant to our project; Wave has many other clever features which could be looked at if our system is to be expanded from the original specification in the future.

4.2.1.1 Robots

Robots are a hugely innovative extension within Wave. As the name suggests, they are a component within Wave that is intelligent; they can interact with and help the users, which is something that hasn’t really been seen in a mainstream market until now. Although the programming and development involved in such a component is perhaps on a scale too large for this project, it is this kind of innovation that sets products apart from each other, and so researching such components is important; they can still inspire other innovative ideas on a lesser scale.

Current robots can perform functions such as debugging and auto-incorporating external features (i.e. Tweets from Twitter can be imported by the robot Tweety!).

4.2.2 Effusia Business Messenger – Business IM

Although not the most similar product I found, it was the most user friendly and shares similar design aims to our system (effectively, conferencing – Democracy isn't touched!). It has a hugely aesthetic interface, and is very easy to use. As it is aimed entirely at a business market, it is full of features that could inspire some great ideas of our own. Here is a list of the few that I really thought were worth noting in the summary:

- Simplicity of communication – Effusia is incredibly easy to use. Whereas Wave is a complete mash up of different features, Effusia is designed to be very simple. From the large text display area, to the smaller text input area underneath and the contact list on the right (think IRC), any user will be able to manage this program from the moment it is installed.
- Organised contact list – The simple contact list on the right can be divided into various groups with a few clicks and drags of the mouse. For example, if I wanted to divide my contacts between my G52GRP group members and other people from my course, it could be done very easily. More so, I could then see what you guys in the group are doing (which would involve you updating your status.
- Notes – A user can easily leave a note attached to his profile, e.g. "Back in 15 minutes". As it aimed at a business environment (which usually involves tight schedules), this informative feature is a must.
- Meeting invites – This feature is something that could be considered for our system. It allows a user to invite others into a meeting, and shows a list of attendees with all their details. It also shows at what time people entered the meeting, and different names are in different colours.

Effusia is a very well thought out program. It really allows business workers to communicate freely. Plenty of inspiration for our project could be taken from this product.

See Appendix 1 'Market Research' for other products found

4.3 Technical Research, Platforms & Tools

4.3.1 Frameworks

We carried out some initial research into the suitability of different platforms for developing our application. Broadly, there were two areas in which we could continue: web-based platforms and desktop-based platforms. The group looked into a number of different options regarding each type of platform. For example, an application that extends a social networking website (such as Facebook) could be used. This would mean that a large part of the system would already be present (such as the user sign-up and login/authentication modules).

4.3.1.1 Web Frameworks

If we decide to develop a web-based system, then we could either code the entire system from scratch, or use an existing framework. Frameworks can make development much faster, and could save the group a lot of time – allowing us to implement more advanced features into the final system. However, poorly designed frameworks can inhibit the development effort – if the API is difficult to work with.

4.3.1.1.1 Django

Django is a Python-based web MVC (Model-View-Controller), developed in a newsroom environment where speed is of the essence. It has an elegant API, and includes a template engine and a database ORM – which is compatible with most common database formats (including SQLite, MySQL, PostGres and MS SQL).

Models in Django are represented as simple classes, with attributes of the class representing the database fields. Foreign key and many-to-many relations are simple to describe.

See Appendix 2.1 for Django source code simple model

These model classes define two database tables (objects) – Poll objects and Choice objects. This is the heart of a basic Django application for voting on polls.

Django uses a simple Python function for the view, which is called from a particular URL location (for example, **`/users/rob/`**.

See Appendix 2.1 for a simple view in Django

This view uses Django's ORM to fetch a list of the latest five poll objects from the database, and sends that data to a template named **`index.html`**.

Templates in Django are very simple, and are basically written as HTML, with some basic logic and variable replacement.

See Appendix 2.1 for an example template written in Django

Our "lead software engineer" – Rob – has extensive experience working with Django, and has developed a number of projects using this framework in the past.

4.3.1.1.2 Ruby-on-Rails

Ruby-on-Rails uses the Model-View-Controller (MVC) architecture, like most contemporary web frameworks, to organise the programming of its applications. RoR uses extensive JavaScript libraries, Prototype and Scripts for the Ajax language.

RoR initially used lightweight Simple Object Access Protocol (SOAP) for its web services; however this was replaced by Representational State Transfer (REST) web services. Since version 2 of RoR was released, it offers both HTML and XML as potential output formats by default.

Action View is used to manage your RoR application views. It can be used to create both HTML and XML outputs by default, and allows the rendering of templates, including both nested and partial templates, and includes AJAX language support built-in to the system.

See Appendix 2.2 for the code needed to create a blog application

You need to configure either of a SQLite Database, a MySQL Database and a PostgreSQL Database

See Appendix 2.2 for the code to configure a MySQL database

Migrations are Ruby classes that are designed to make it simple to create and modify database tables.

See Appendix 2.2 for the code for Migration

The syntax of ROR is similar with other languages.

See Appendix 2.2 for an example of the syntax on Ruby-on-Rails

4.3.1.2 Desktop-Based Platforms

4.3.1.2.1 Java Swing

Swing is Model-View-Controller toolkit for Java, used extensively for the development of GUIs. As it is Java based, it has the advantage of being set in a language already hugely familiar with most programmers (it is included in the Java Standard Edition). Mechanisms within Swing allow for the look and feel of an application to be changed with a minimal effect on the code of the application.

Swing is entirely based on components, which are effectively objects with characteristic behaviours that are known to the programmer. Swing provides various extras for components, such as icons and tooltips.

The Swing framework allows fine control over components, in effect making them customizable. For example, users can program to customize Swings standard set of borders, backgrounds etc in order to give the GUI a customized look. It is even possible to arrive at a totally unique visual representation by high-level customisation.

Due to Swings extensive reliance on runtime mechanisms, it can respond to basic changes in its visual settings at a fundamental level. This lets the user of the application further change the look and feel while the app is running. This doesn’t change the application’s code.

Unlike non-visual applications, Swing applications cannot be debugged simply using standard debuggers. This is because Swing generally double-buffers (in a non-displayed buffer) and then prints directly onto the screen, making it impossible to observe the effect of every graphical operation performed by the debugger. This, and other problems related to the printing thread, can result in screen freezes.

4.3.2 Source Code Management

We felt that version/source control would be essential to the smooth operation of our group project. We needed a fast and simple way to share code and documentation – whilst not treading on each other's feet whilst working on different parts of the project.

Using a Source Code Management system allows multiple group members to work on the same file or document at once. When editing binary files, however, (such as Microsoft Office documents), the SCM is not able to "diff" the file - that is, to see the differences between the two. This means that it is left to the user to either overwrite the old file completely, or throw away all their changes when a collision occurs.

This shouldn't be a problem for our group, however, as a single group member is responsible for collating all sections of, and submitting, the documents required in our deliverables. That is, one group member will be actually committing the documents to the repository - though other group members may work on them in a separate fashion.

To this end, we conducted some research into Version Control Systems. The results of this research are included below.

4.3.2.1 Subversion

Subversion (or SVN) is a **client-server** based Source Control Management tool. That is, the repository resides on a central machine (the server) and each user checks out a "working copy" (the client). Each time the user makes a set of changes, they commit them to the server. Commits are stored in a linear fashion, one after the other. In this way, SVN manages the version (or **revision**) of each file using a number, that increments by one on every commit. This allows the user to instantly see which version of a file is the latest – by comparing revision numbers.

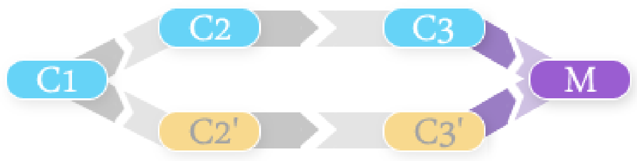
Each document is stored as a hard-links to the most recent revision of said file, keeping the overall size of the repository down.

See Appendix 2.3 for an example of the 3D-Tree Structure SVN uses

4.3.2.2 Mercurial

Mercurial is a **distributed** Source Control Management tool. It differs from Subversion in that there is no central machine on which the repository is stored. Instead, each user has their own copy of the repository, which they commit to locally when their changes are complete. To share changes, users synchronise their local repositories.

This means that the structure of a Mercurial repository is not linear, as with Subversion. Instead, it could be described as a **Directed, Acyclic Graph**. The diagram below shows a representation of this graph.



The diagram above shows the following situation; the repository is copied from state **C1** into state **C2** and **C2’,** by two separate users. These users then make their changes, and commit to their local copes - which results in states **C3** and **C3’** respectively. These two states can then be **merged**, resulting in a hybrid of the two previous states.

The non-linear nature of a Mercurial repository means that there are no revision numbers as in Subversion. Instead, each revision is identified by an **SHA-1 Hash** of the contents in that revision.

See Appendix 2.4 for an example of the hashes in a Mercurial repository

4.3.3 Project Hosting
It was obvious early on in our research that the group would need a centralised location to store our code and documentation, which would be accessible from anywhere. That is, we needed a place to host our project's repository.

4.3.3.1 Google Code
Google provides a free online project hosting service, called **Google Code**.

It offers either a Subversion or Mercurial repository included as standard, though it requires that the code for the project be made open-source. Also, it includes a Wiki section - with all pages contained in the repository - so anything stored on the Wiki is easily backed up by checking out or cloning the repository.

Google code requires all users to have a Google account.

4.3.3.2 SourceForge
SourceForge is a free web-based project hosting site, run by GeekNet.

It offers a huge array of services, including a shared repository, SSH access, Wiki, forums, and downloads. It is, however, a difficult and length process to set up a new project.

4.3.4 Collaboration
Also, the group would need a way to collaborate whilst working on the project - as we would not all be in the same room at all times.

4.3.4.1 Basecamp
Basecamp is an online service offering group collaboration and project management, developed by a company called 37signals.

Basecamp is free for small projects, and allows users to track deadlines and to-do lists, post messages, and chat in real-time. Also, project milestones can be managed using the suite, which can be very helpful when working against the clock.

Perhaps most importantly, Basecamp offers rich email integration. When a message is posted, every member in the group is notified of that message – and can reply via email. This is then interpreted by the Basecamp software, and posted automatically onto the project's page. This not only ensures that group members don't miss an important announcement, but also saves considerable time logging into the web application to post a reply.

4.3.4.2 Project2Manage
Similarly to Basecamp, Project2Manage is an online project-management suite.

It is free for one project, though this does limit the feature set somewhat – as file sharing and time-tracking is not permitted in the free plan. This is assumed to mean that neither deadlines or milestones can be tracked using Project2Manage unless a paid plan is purchased – unlike Basecamp which offers these features in its free plan.

4.4 Summary of Research

After looking into a number of products key to the market we are targeting, especially within the market research section, we can conclude that although there are many products on the market offering a similar feature set, no product offers the democratic nature that we hope to implement within our system. Though we aim to base the system primarily on features we have discussed as a group, features from other popular products may still be considered.

Apart from market research, another important aspect of this section was looking into the technologies available for building our system. The group has extensively researched web-frameworks suitable for the task of building our system, as a web-based platform is currently favoured by the group. We will also have access to free project-hosting and a method for source code management.

5 Implementation Decisions

Firstly, the group had to decide whether to develop our application in one of two ways: a web-based application, or a desktop-based application.

5.1 Web vs. Desktop

5.1.1 Web

Advantages	Disadvantages
Faster development	Limited to one-way communication (request-response)
User-interface layout simplified (HTML + CSS)	Interactive UI is more difficult (requiring JavaScript or Flash)
Does not require installation of a client	
Truly multi-platform	
Possibility of SaaS (Software as a Service)	

5.1.2 Desktop

Advantages	Disadvantages
Two-way communication possible (via sockets)	Slower development (custom protocol may be required)
Implementing an interactive UI is easier	User-interface implementation is more difficult
	Requires installation on all client machines
	Hard to make truly multi-platform application
	SaaS is ruled-out

The group felt that the advantages of a web-based application are so great that our development should focus on this style.

Though the fact that a web-based application is forced to adhere to the request-response nature of HTTP may require some creative programming, we feel that this is an obstacle that can be overcome.

5.2 The Framework

Next, the group needed to decide on a language and/or framework to use for the development of the project. Broadly, we had two options at this point - we could either write the entire application from scratch, or use a web framework to ease the development and speed up the process. However, frameworks can hamper development if they are too inflexible - so this choice is vital.

We researched two popular web frameworks - Django and Ruby on Rails. Rob has prior experience programming using the **Django** web framework, and has done extensive research into its suitability. When this was discussed with the group, no-one had any objections to using Django, or experience with any other framework. Though there was a belief that Django would be suitable for our project – the only way to know for sure was to build a prototype, to act as a "feasibility study" – in order to find out whether we could indeed implement our project using Django. The results of this study are recorded in the next section of this report.

5.3 Development & Tools

To begin development, we had to decide on which tools to use to aid our group. We had already identified the different **types** of tools needed - source code management, collaboration, and project hosting. We weighed up the different tools for each, considering experience that group members had with each - and decided on the following:

- Subversion for source code management
- Basecamp for collaboration
- Google Code for project hosting

On Google Code, we created a Wiki for sharing documents such as the meeting minutes with each other, which we categorised by date allowing us to differentiate between the different meeting minutes, and the additions/changes made to them since the meeting took place. Google code automatically stores the Wiki pages in a directory in the subversion repository - so each revision of every file is tracked in the same place (be it code, documentation, or Wiki pages).

Also, Google Code allows us to browse through the source code (and documentation) in the repository, and view the change log for the project in a browser. This is useful to the group - as we can keep track of changes from anywhere. Also, a contributing factor to our decision is that most of the group members have Google accounts - which are a requirement to use Google Code. This means that for the majority, there was one less step to get up and running with the hosting system.

5.4 Low-Tech Prototyping

In order to nail down some key aspects of how the final system should operate, the group employed a method known as “low-tech prototyping” or “play-testing”. This is where the group plays out the way in which the system may operate on paper, in order to find any faults that may so far have been undiscovered. We produced a number of sets of “rules” for the system, and used these to refine the final interaction.

See Appendix 5.3 for pictures of the low-tech prototyping, and the rule-sets, and the findings from the play-testing process.

We found that the play-testing was an extremely helpful (and also time-efficient) method for prototyping rule sets, and we identified a number of issues that would otherwise have been undiscovered until the system was implemented.

5.5 Initial Implementation Decisions

5.5.1 Database

The first thing that we decided about our implementation was the *models* - or how the database will be designed. To give the most detail, the actual **models.py** file and an Entity-Relationship Diagram is included in the appendix, though a summary is given below:

It was obvious that we would need a database table (a model) for the following items:

- Conference Rooms
- Messages

Obviously, other objects will need to be represented in the database (such as users, votes, etc.), but Django comes with a package of applications that provide common services such as user authentication - so we need not worry about that. Also, Django has an application for voting on a "poll" - called **django-polls**. This feature will not be implemented in the prototype, but we may use this application in our final system if it is deemed suitable. Otherwise, we can write a custom application from scratch to give similar functionality.

See Appendix 5 for model file and ERD.

5.5.2 Views

Next, we needed to decide on the structure of the **views** (commonly referred to as the **controllers** in other MVC frameworks). These define the different "angles" that the user can view the application from - and what data each should use.

We decided on one view for the conference room - that would differentiate between a normal browser request, and an asynchronous (AJAX) request. This would allow us to make requests to the same URL from different places - and retrieve different responses.

Obviously, a view would be required for each of the auxiliary pages (login, logout, conference list) - though these fit nicely into the Django default (and generic) views that are provided with the framework. This means that the programmer can adhere closely to the DRY (Don't Repeat Yourself) principal.

5.5.3 Core Application

Finally, we needed to decide how to combine our code into an application - so that it slots into the Django framework well. It was clear that the "conferencing " functionality could all be grouped into one application, and still make good programming sense. We decided to call this application **conference**, though the name may change in future iterations.

6 Initial Prototyping

In order to find out whether the Django web framework would be suitable for our project, the group decided that we should build a prototype system. It was decided that this system should include only the core functionality of the final system - as a sort of feasibility study. This is because it was still not known whether the request-response nature of the web (and Django itself) would be entirely suitable for this kind of system. To this end, we came up with a very simple specification for the prototype.

This is simply a subset of the list of things that the final system **should** do. The reason for this is because the prototype is intended to find out whether Django is going to present any major problems for our project, and whether the actual system design is actually progressing in the right direction.

6.1 Specification

1. The system should have a web-accessible interface
2. The system should allow users to login and logout
3. The system should present users with a list of available conferences
4. The system should allow users to enter a conference, and chat with others in that room in near-real time. This requires the user to be logged-in.
5. The system should allow users to see who else is in the conference
6. The system should allow users to leave a conference, and return to the list of available rooms
7. The conference system should use an asynchronous method of communication (i.e. the page should not refresh when a new message is available.

The last point above (point 7) is important - as it ensures that the system does not simply refresh the web page in order to check for new messages. Though this would work, it was deemed totally impractical, as it would interrupt the user every time the refresh occurred, not to mention the added load that would be placed on the web server. This would be the real test as to whether the framework has the functionality (and the group has the skills) to develop such a system using web-based technologies.

6.2 Testing

The results of the prototyping are summarised below.

See Appendix 4 for screenshots of the Prototype

Test	No Data	Normal Data	Erroneous Data
The system should have a web-accessible interface	The system has a simple webpage so you can pick which conference you would like to enter. Clicking a link will take you to another page where you are prompted to enter your username and password, if entered correctly it will take you to the room where there is a simple black and white chat room window where you can enter and view messages.		
The system should allow users to login and logout	Tried logging into the system without any data inputted which it didn't allow me to do, this was expected.	Tried logging into the system with a registered username and password and this allowed me into the system.	Tried logging in with an unauthorised username and password and this was unsuccessful which is what should happen.
The system should present users with a list of available conferences	The front page of the system allows you to pick a conference from a list.		
The system should allow users to enter a conference, and chat with others in that room in near-real time. This requires the user to be logged-in.	Once you have entered a conference you can chat with the people in near-real time just by typing a message at the bottom and pressing enter/submit. This doesn't require the page to be refreshed each time a new message is entered by a user.		
The system should allow users to see who else is in the conference	There is a list of users in the conference on the right hand side of the chat room interface.		
The system should allow users to leave a conference, and return to the list of available rooms	There is a 'leave conference' link at the bottom of the chat room page which takes you back to the list of available conferences.		
The conference system should use an asynchronous method of communication (i.e. the page should not refresh when a new message is available.	The chat window automatically refreshes itself without the user noticing, this doesn't involve the whole page being refreshed.		

6.3 Evaluation

The prototype was successfully implemented using the Django framework, and fulfilled all aspects of the specification. The result of this prototype is more than just a simple yes/no answer - as to whether Django is suitable - rather it allowed us to find out about any potential issues we may encounter while working with Django.

The only problem that we may encounter when developing this system is the request-response nature of HTTP. The very core of the system however, that deals with the messaging aspect (checking for new messages, and displaying them on the screen without refreshing) is working in the prototype - so this appears to have been solved early on.

However, this means that each client will be making requests to the web server, very frequently. We may encounter issues with scalability - if the requests start to take longer to process. One thing that should be monitored carefully is the efficiency of the code that is used to check for new messages - as the leaner this can be made, the better.

7 Functional Specification

7.1 Overview

The project (named DemoConf until a final name is chosen) is a service that allows users to participate in a democratic debate or discussion - to achieve a goal of agreement on a particular subject.

The system forces users to vote on a goal at the end of each time period (which is 10 minutes by default). The goal (usually an issue that must be debated) is specified when the conference room is created. If all (or a specified proportion) of the participants agree when the vote is cast, then the goal has been achieved and the conference can be ended. Otherwise, the discussion continues for another period.

The graphics and layout of the system shown in this specification are merely to illustrate the underlying functionality. The actual look and feel will of the final product will be developed over time.

7.2 User Stories/Scenarios

The user stories or scenarios specified in this specification will help to define how users interact with the system.

7.2.1 Scenario 1: John

John is a senior developer on the open-source Ubuntu Linux project. He often needs to discuss important issues regarding the project with other developers, who are located in different countries around the world. At present, this is achieved by using a combination of IRC, and AIM (Internet Relay Chat, and AOL Instant Messenger) to chat online.

Acting on a recommendation from his friend, John signs up for an account with **DemoConf** - the online debate and discussion service. The next time John has to discuss an issue with this peers, he directs them all to the service also, and sets up a conference room. The team join the room and start to discuss.

The team find that the system encourages decision-making, and they reach a decision after 3 periods of 10-minutes. Afterwards, John receives an email from Philip, one of the development team, saying how well the discussion went - and that he feels they should use **DemoConf** every time they want to discuss an issue.

7.2.2 Scenario 2: David

David is a second-year undergraduate university student. He is participating in a group project, in which he has to design and build a software system with five other members of his year - which he did not previously know before the project started.

David’s group need to make a major decision about the direction of their project, but the group aren’t available to meet that day. James, another member of the group, suggests that they use **DemoConf** to discuss the project that evening - and the group agree to give it a try.

That evening, James signs up for an account on DemoConf, and creates a conference room. He sets the objective poll as follows:

Which language should we use to develop our application?

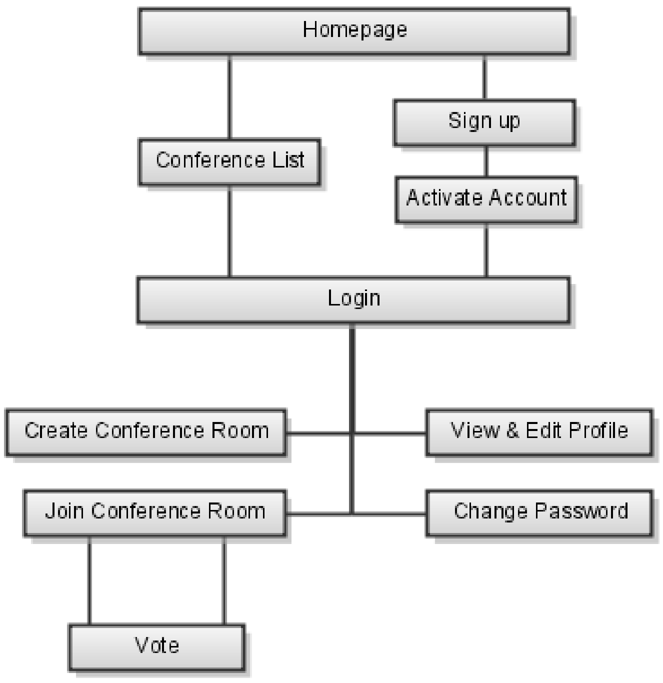
a) Java
b) Python

The group then all sign up for accounts, and join the conference. They begin to discuss the two options open to them, and vote after the first 10-minute period is up. Three of the members choose Java, and three choose Python - so the conference continues.

After 2 more votes, the group is starting to agree that Python is the better option. When the fourth vote is cast, the whole group agrees that the project should be developed in Python, and the goal is achieved. James chooses to end the conference, and everyone says goodbye.

7.3 Flowchart

This flow chart gives a high-level overview of how the user interacts with the system, and vice-versa.



7.4 Functional Requirements

1. The system should have a web-accessible interface
2. The system should allow users to login and logout
3. The system should present users with a list of available conferences
4. The system should allow users to enter a conference, and chat with others in that room in near-real time. This requires the user to be logged-in.
5. The system should allow users to see who else is in the conference
6. The system should allow users to leave a conference, and return to the list of available rooms
7. The conference system should use an asynchronous method of communication (i.e. the page should not refresh when a new message is available.
8. The system should allow users to sign-up for a user account.
    a. Users must choose a username and password
    b. Users must enter their email address
    c. The system should send the user an email, which includes a link that the user must click in order to "activate" their account
    d. Before the user has clicked the link in their email, the account should be inactive - so the user cannot login
9. Users should be able to change their password.
10. Users should be able to change their email address.
11. Users should have a profile, containing their personal information shared with other users.
12. Users should be able to edit the information in their profile.
13. Users should be able to create conferences.
14. When a conference is created, it should not be published immediately - instead residing in a private area until the user chooses to "publish" it and allow other users to join.
15. Users should be able to edit conferences in their private area, before they are "published".
16. Users should be able to delete conferences in their private are, before they are "published".
17. Once a conference is "published", it should not be editable or removable.
18. The system should require users to input a "poll" when creating a conference. This should represent the "objective" of the conference - all members should agree on the issue named in the poll.
19. The system should allow users to choose the length of the "period" when creating a conference (a default value of 10 minutes should be provided).
20. The system should switch to a vote at the end of each specified time period.
21. When in the voting mode, users should not be able to enter messages in the conference. Everyone is required to vote on the poll specified at creation time.
    a. If all members in the conference agree, then the conference can be officially called finished - and the objective achieved (this should be decided by a vote).
    b. If all the members do not agree, another period is entered to try and resolve the disagreement. At the end of each period, the voting mode is entered again.

7.5 Non-Functional Requirements

1. The web interface should be accessible from any platform, using either of four main browsers - Firefox, Google Chrome, Internet Explorer and Opera
2. The system should be secure in that unauthenticated users cannot access conferences
3. The system should be available
4. The system should respond to requests in a reasonable time period

7.6 User Interface

The designs we have created show the potential layout for the conference room screen - as this is the most customised screen in the system. Both implement most of, but not the entire feature set of the system as decisions are constantly being made which affect the features that remain as part of our final specification.

See Appendix 3.2 for the initial User-Interface designs

The first initial design shown is the earliest to be completed. It contains most of the features we planned to implement from the beginning, and although many of these have remained the same throughout the design process, some of them have been discarded. The name of the system shown on this design, for example, was provided before anything had been decided on name-wise; however since then we have used the title 'DemoConf' for all other designs and prototyping ideas.

The second initial UI is very similar to the first in respect to the features implemented in the design process. The structure for the conference room is almost identical, but this is primarily because it is such a well-known interface for chat-room software that it's immediately drawn to when designing your own interface. The users list is located on the left of the design, as is the same with UI #1, and both implement a clear display showing how much time the active speaker has remaining. There is a location difference between these two designs on this particular component however.

The third and fourth designs were created following the idea that we would implement a voting system, where listeners could offer feedback on the active speaker's opinion and whether or not they agree. This is clearly shown by the presence of the up and down arrows on the third design, and the voting symbols on the fourth design, both using the colour green to symbolise positive feedback, and red to symbolise negative feedback. A difference in these designs is that the initial UI#3 has the option of a neutral vote, whereas initial UI#4 would force the listener to abstain from voting if they felt neutral on the topic.

The final design created focuses more on explaining the features introduced in the interface, and less on the aesthetics, such as colour, layout, etc. It introduces new components not considered in previous designs, such as a Help menu, which could be used to explain the interface to new users of the system. Most of the new features the design considers are advanced, and could be considered to give our tool a competitive edge. These features can be seen in the appendix provided, labelled in their respective sections within the interface design.

8 Problems Thus Far and Solutions

- **Functional Specification was far too bare.**

Originally, we missed out key aspects of the technical specification, as it was purely a list of bullet-points that we hoped to expand upon. This has since been amended, in that we've written a functional specification that covers the requirements, and kept the initial set of points for reference within the report.

- **Some members are not always present and/or reachable.**

This issue is directly linked to the amount of work being produced by each person. Unfortunately, some people seem to not bother turning up to meetings most of the time, despite their attendance in lectures being generally high. If they don't attend the meetings, they won't be aware of the situation regarding the work and therefore won't be able to contribute. Those of us who are attending the majority of meetings and contributing the most in both attendance and workload have communicated with the group via a number of methods, but usually without response which makes it extremely difficult to contact other group members.

Unfortunately this means that not all group members were present at the play-testing meeting, which was one of the most important decision-making meetings of all. **(See Appendix 6 for Meeting Minutes & Attendance)**.

- **Initial time plan did not consider the exam period.**

Unfortunately, the first draft of our time plan was extremely rough and did not separate the main tasks (such as interim report / final report) into sub-sections, allowing us to focus on particular points at different times. This caused confusion, but our time plan has since been amended to not only account for the exam period, but now splits up the larger parts of the project into smaller, more manageable sub-headings.

- **Everyone in the group has different levels of programming knowledge.**

This couldn't really be helped, as people began programming at different times and this directly affects their ability in that area. Fortunately, we're close enough as a group to be able to work around this issue, and we will be distributing the work load evenly to cater towards a group member's strength, which allows for certain people to be better at something than others, as you can make up for it in another part of the project.

- **Communication issues due to language-barriers.**

This is a big issue, though we are learning to adapt to this however by talking to the affected group members and ensuring that they know when they need to voice their input to ensure they are heard and that their opinion does not get overlooked.

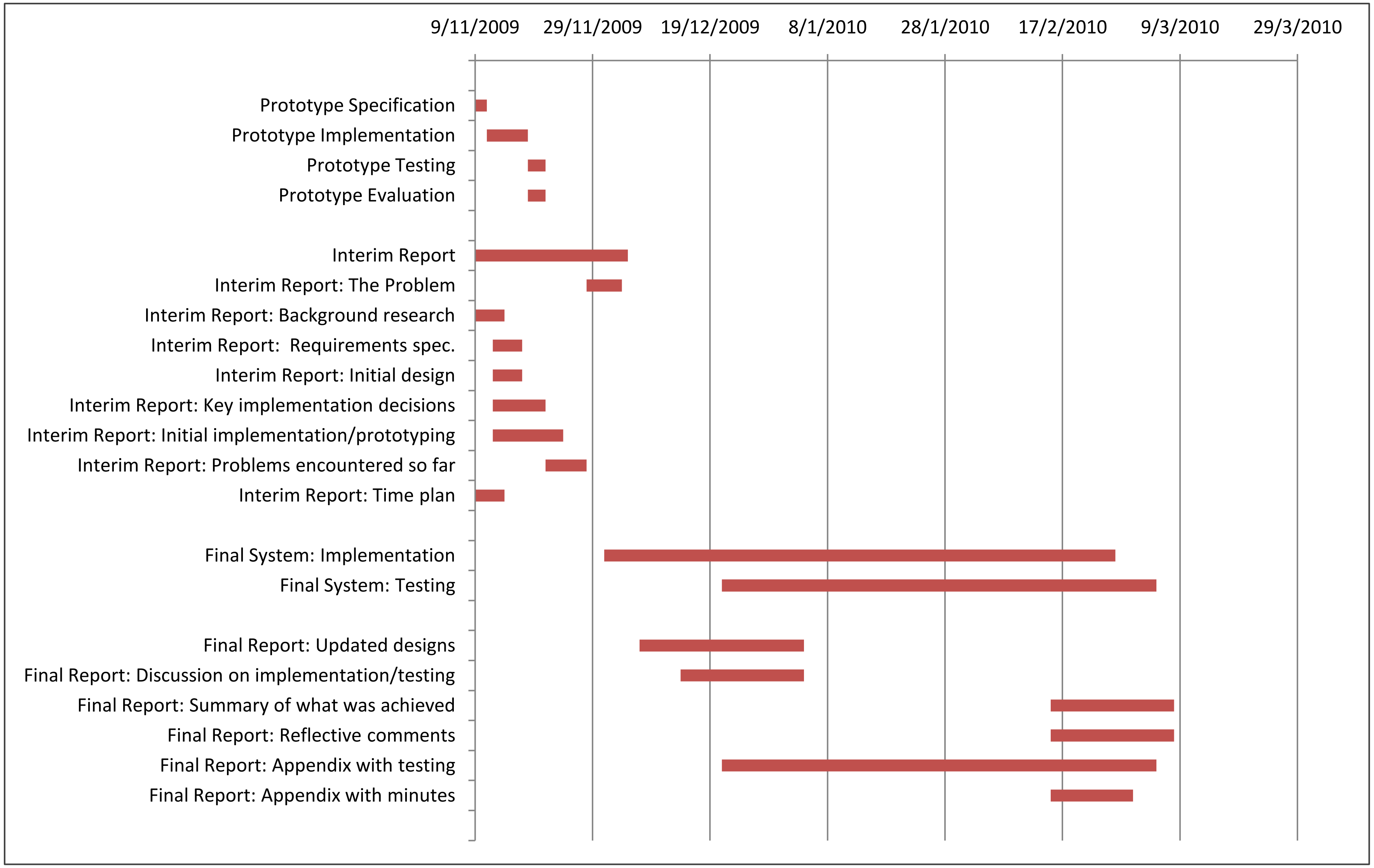
- **Differing amounts of work have been contributed by group members.**

This has been an issue throughout the project time-span; however it has become more obvious as we've progressed towards the end of term. Certain members are still working far harder than others, and we hope this doesn't continue into the second stage of the project, after the break.

9 Time Plan

Task	Start Date	End Date	Group Members
Democratic Conferencing Tool	9/11/2009	21/2/2010	
Prototype Specification	9/11/2009	11/11/2009	Robert, Kit
Prototype Implementation	11/11/2009	18/11/2009	Robert
Prototype Testing	18/11/2009	21/11/2009	Henry
Prototype Evaluation	18/11/2009	21/11/2009	Tammie
Interim Report	9/11/2009	5/12/2009	
Interim Report: The Problem	28/11/2009	4/12/2009	William
Interim Report: Background research	9/11/2009	14/11/2009	Carl, Kit
Interim Report: Requirements spec.	12/11/2009	17/11/2009	Robert
Interim Report: Initial design	12/11/2009	17/11/2009	William, Robert, Tammie, Carl
Interim Report: Key implementation decisions	12/11/2009	21/11/2009	Robert
Interim Report: Initial implementation/prototyping	12/11/2009	24/11/2009	Robert, William
Interim Report: Problems encountered so far	21/11/2009	28/11/2009	Tammie
Interim Report: Time plan	9/11/2009	14/11/2009	Henry
Final System: Implementation	1/12/2009	26/2/2010	Tammie, Robert
Final System: Testing	21/12/2009	5/3/2010	Henry,Carl
Final Report: Updated designs	7/12/2009	4/1/2010	Robert, William
Final Report: Discussion on implementation/testing	14/12/2009	4/1/2010	Robert, Henry, Carl
Final Report: Summary of what was achieved	15/2/2010	8/3/2010	Robert, Billy
Final Report: Reflective comments	15/2/2010	8/3/2010	Robert
Final Report: Appendix with testing	21/12/2009	5/3/2010	Henry, Carl
Final Report: Appendix with minutes	15/2/2010	1/3/2010	Tammie

9.1 Gantt Chart


9/11/2009
29/11/2009
19/12/2009
8/1/2010
28/1/2010
17/2/2010
9/3/2010
29/3/2010
Prototype Specification
Prototype Implementation
Prototype Testing
Prototype Evaluation
Interim Report
Interim Report: The Problem
Interim Report: Background research
Interim Report: Requirements spec.
Interim Report: Initial design
Interim Report: Key implementation decisions
Interim Report: Initial implementation/prototyping
Interim Report: Problems encountered so far
Interim Report: Time plan
Final System: Implementation
Final System: Testing
Final Report: Updated designs
Final Report: Discussion on implementation/testing
Final Report: Summary of what was achieved
Final Report: Reflective comments
Final Report: Appendix with testing
Final Report: Appendix with minutes