**G52GRP**

**Democratic Conferencing Tool**

**Interim Report – November 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

[Table of Contents 2](#_Toc247097335)

[1 The Problem 3](#_Toc247097336)

[2 Background Information & Research 4](#_Toc247097337)

[2.1 Survey of Existing Systems 4](#_Toc247097338)

[2.2 Market Research 4](#_Toc247097339)

[2.3 Technical Research, Platforms & Tools 4](#_Toc247097340)

[2.3.1 Source Code Management 4](#_Toc247097341)

[2.3.2 Project Hosting 7](#_Toc247097342)

[2.3.3 Collaboration 8](#_Toc247097343)

[3 Requirements Specification 9](#_Toc247097344)

[3.1 Functional Requirements 9](#_Toc247097345)

[3.2 Non-Functional Requirements 10](#_Toc247097346)

[4 Initial Design & UI 11](#_Toc247097347)

[5 The Platform/Implementation 12](#_Toc247097348)

[5.1 Web-Based 12](#_Toc247097349)

[5.2 Desktop-Based 12](#_Toc247097350)

[6 Initial Implementation/Prototyping 13](#_Toc247097351)

[6.1 Prototype Specification 13](#_Toc247097352)

[7 Problems Encountered 13](#_Toc247097353)

[8 Time Plan 13](#_Toc247097354)

G52GRP Interim Report

# The Problem

To clarify, gp09-sdb’s project is to design and create a Democratic Conferencing Tool. Our primary aim is to introduce a new product to an existing market, with ideas both new and familiar to those who are interested in such products, as we feel it is important to research other successful systems whilst we intend to implement our own original features on top. In essence, the system should provide a platform for users to collaborate and participate in debates, in a democratic way. That is, the outcome of the conference should be decided by a vote.

Voting will be a central aspect of the final system, as this is the basis for a democratic environment. The system may also include limits on the amount of content a single user can input into the conference - though the mechanism for this is to be decided upon in the planning stages of the project.

# Background Information & Research

## 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.

## Market Research

## Technical Research, Platforms & Tools

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).

### 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.

#### 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.



#### 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. The hashes in a Mercurial repository look like this:

1:1ef7872431f9c64908c732f0bcd4db5700b4cb70

This means that unlike Subversion, two revision identifiers cannot be easily compared to see which is newer.

### 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.

#### Google Code

GoogleGoogle 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.

#### 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.

### 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.

#### Basecamp

Basecamp is an online service offering group collaboration and project management services. It is free for small projects



Alongside SVN, we have also implemented Google Code into our plan for file sharing. In short, Google code offers users access to various developer tools and products. Google also has Project Hosting, which has enabled us to create a directory for our group project to store all documentation and code that is created. As this doesn’t allow for updates to be shared instantly with each member while working, it will only be used to share documentation upon completion, whereas SVN can be used for all ‘in-between’ updates.

On Google Code, we currently have a directory specifically for sharing files with the other group members, a ‘Wiki’ for typing and sharing documents such as the meeting minutes with each other, which comes with a date sub-heading allowing us to differentiate between the different meeting minutes, and the additions/changes made to them since.

Something else that is important for our project is possibly utilizing Google App Engine, which Google uses for hosting web applications created by its users. As we have decided upon creating our project as a web application, rather than desktop-based, this is an important feature provided by Google for us.

Also, the group chose to utilize the free Subversion hosting offered by Google, in the form of **Google Code**. Using this service meant that the group had a central location to store code and documentation, that would always be available and, perhaps more importantly, accessible through a web interface. This gives members the ability to browse the code repository without actually checking out a copy, which is very useful for quick status-updates.

# Requirements Specification

## 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.
   1. Users must choose a username and password
   2. Users must enter their email address
   3. The system should send the user an email, which includes a link that the user must click in order to “activate” their account
   4. 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.
    1. 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).
    2. 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.

## 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

# Initial Design & UI

Here is an initial UI design, drawn up before a number of the features were decided upon:



This is of course, not a finished design, however it includes the basic features we wish to include – Users online list, chat window, user status, etc. The list of features has grown substantially since this design was created, and the next versions will reflect that.

# The Platform/Implementation

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.

## Web-Based

|  |  |
| --- | --- |
| 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) |  |

## Desktop-Based

|  |  |
| --- | --- |
| 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.

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.

There are a number of web frameworks that would be suitable for a project such as this - Ruby on Rails, CakePHP, and Django to name but a few. However, Rob has prior experience programming using the **Django** web framework, which is written using the Python programming language. 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.

# Initial Implementation/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.

## Prototype 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.

# Problems Encountered

# Time Plan

Still to be added:

More recent UI designs, information about play-testing session, research notes, summary of progress to date.