University of Cambridge

Computer Science Tripos, Part IB

Group Project Team Echo

Multi-touch Conference Project plan

Authors:
Mona Niknafs (mn407)
Yojan Patel (yp242)
Alexandru Tache (at628)
Philip Thomson (prt28)
Petar Veličković (pv273)

Client:
Catherine White, BT

28 January 2014

Contents

1 Project Plan

1.1 Introduction

The main purpose of this document is to present a more detailed breakdown of the technical work that is required to be undertaken for the Multi-touch Conference group project, as described in *Appendix A* (Management Strategy) of the functional specification document. As most of the functionalities are already described in that document, the module descriptions here are brief, and more emphasis is given to the amount of people/time needed to implement them, testing strategies, and diagrams that will enhance the reader's understanding of the module structures and dependencies. Note that the UML diagrams shown here will represent only the bare minimum of methods we would expect the listed classes to have; it is highly likely that the final project will have a different abstract class layout (which is still based upon the diagrams given here).

1.2 Modules

The project will be split in the following **five** modules:

- the server;
- a client base library for communicating with the server, and its extensions:
  - the touchscreen client;
  - the Android client;
  - the computer client.

There is an additional package (the *"data"* package) containing all the classes that represent units of data stored within or in movement between the other modules. This section describes each of those modules in turn, with accompanying UML diagrams to aid clarity.

1.2.1 Server

The **server** will be the core component of our project. It can be split into two large components:

- a **client handler**, that handles and multiplexes all of the messages received from clients and produces appropriate replies to the relevant client(s);
- a **data analyst**, that upon request processes conversation data in order to produce various kinds of output, f.ex. keywords for individual conversations, conversations sorted by popularity, searching for particular words, and many other possibilities.

Since the server will store all the conference data it must be resilient to failure, thus all the data must be stored in a database on disk. A more detailed structure of the server is shown on *Figure 1.1*.

The server is the largest and hardest module to implement and as such **two** group members are assigned to work on it (with the work roughly split between the client handling and data analysis tools). Testing the client handler will involve stressing it with different kinds of client load scenarios and observing the response latency, and the data analyst will be tested by feeding pre-made conversation data (f.ex. scripts from movies) and observing the usefulness of the output (f.ex. whether or not the keywords extracted correspond to the movie's theme, etc). The entire work regarding implementing and testing the server should not take more than 30h/person, if no major difficulties arise.

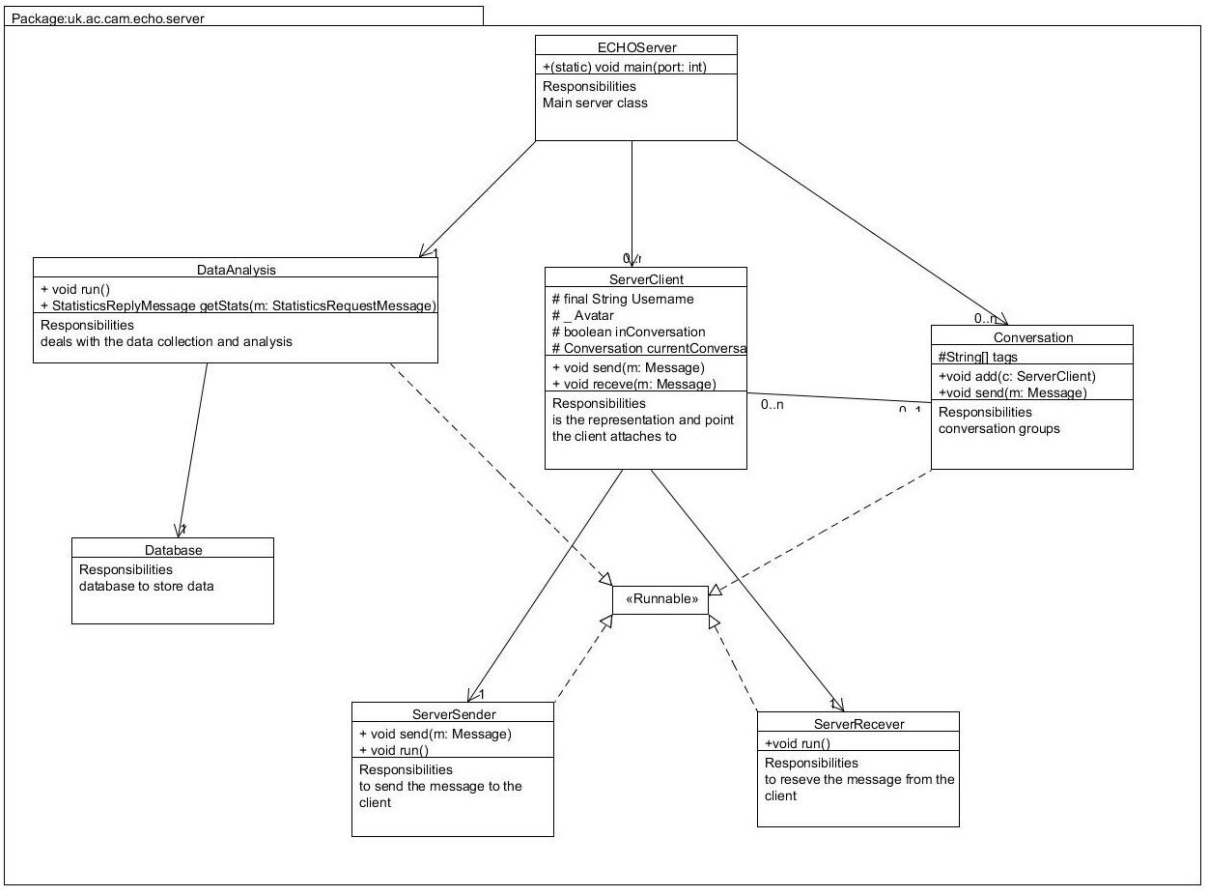

Figure 1.1: UML diagram depicting a more detailed structure of the server.

1.2.2 Client

In order for other modules to easily operate with the server, a **client library** that handles all the server communication needs to be provided. Its structure is relatively simple, because it only serves to provide all the clients a single type of base class that will directly contact the server on their behalf (this involves implementing methods for sending and receiving messages to/from the server and handling basic events); all the additional interesting processing of the received raw data from the server is handled in the modules that extend it. Because of that, we anticipate that the base client should be a simple module to implement and as such, only **one** person is assigned to its implementation. It should not take more than 5h of work to have a stable implementation, and testing will be performed in conjunction with the server.

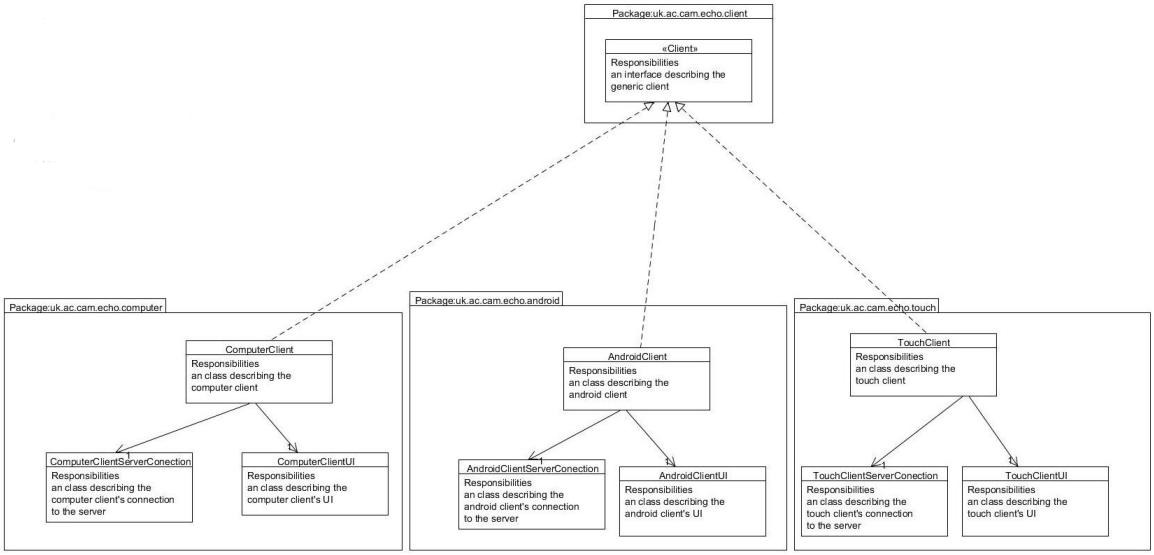

Figure 1.2: UML diagram depicting a more detailed structure of the client and its extensions.

1.2.2.1 Touchscreen client

The **touchscreen client** will be the main user interface. The application will run on a PC running Windows 8.1, with a 27+ inch multi-touch screen that supports at least 10 simultaneous touches. Its role will be to allow multiple users to follow and contribute to different conversations simultaneously using the touch capabilities of the screen. It should also provide users with suggestions about other relevant conversations they might be interested in joining, as well as personalised information for the users that are currently connected with the screen. As part of this effort the application will need to communicate with the server using the library specified in subsection 1.2.2, and interpret the received data to provide useful output for the users; also, it should be able to generate QR codes that the client mobile applications will be able to scan in order to connect to a specific conversation.

This module clearly has the largest demonstrative value, as its proper implementation will display exactly how a multi-touch screen can enhance the conferencing experience, which is the key distinctive feature of this project. It is also the module where we expect the most implementation challenges, depending on how many problems arise from the touchscreen API. To reflect its importance and possible difficulties, **two** group members will be assigned to the implementation this module at all times, with additional members joining in for assistance as needed. Tests will involve issuing different kinds of request batches to the server and observing the output, as well as scheduling "mock conferences" among group members in order to investigate its conference-enhancing value. We estimate this module to require roughly up to 30h of work per person, however this is subject to change.

1.2.2.2 Android client

The **Android client** will be implemented as an application that will provide a simple interface that will enable individuals to contribute to group conversations using their personal smartphone/tablet devices. As the touchscreen client the application should use the library from subsection 1.2.2 in order to communicate messages and requests to the server. It should periodically display notifications to the user about interesting topics, give suggestions for conversations to join, allow for keyword/tag searching of conversations, and enable QR code scanning to integrate with the multi-touch screen's interface if desired.

This is a very important module as it will likely be the dominant manner of communication between users in most scenarios, however its implementation should be fairly straightforward, as well as testing (as there are only a handful of features to test, and they can be done by running the application on any Android smartphone/tablet and observing the behaviour). Hence only **one** group member will be assigned to work on this module and a reasonably stable application should be expected within 20h of work.

1.2.2.3 Computer client

As an additional extension, we will implement a **computer client** that is designed to run on a PC with a generic monitor without touchscreen capabilities. Using a personal computer during a conference falls out of the usage model we have predicted for this application, and we have opted to design it solely for the purpose of an additional means of testing the final product. As such its interface will be minimal, with controlling the application and receiving the output done either via the terminal or with a very simple graphical interface. **One** group member is assigned to this module, and we expect it in a stable state within 3h of work.

1.2.3 Data

This package contains all the various kinds of **custom data classes** that satisfy any of the following criteria:

- are stored within any of the modules;
- are transferred between modules in any way (f.ex. for message passing);
- are stored in the server database and can be queried upon.

We have initially identified a few essential data classes that are bound to be used in multiple modules (such as User, Message, Conversation, Conference...) and it is likely that more classes will be added as development proceeds. As the need for these classes arises during development, no particular group member is assigned to implement this part; whoever discovers the need for a new data class will be assumed

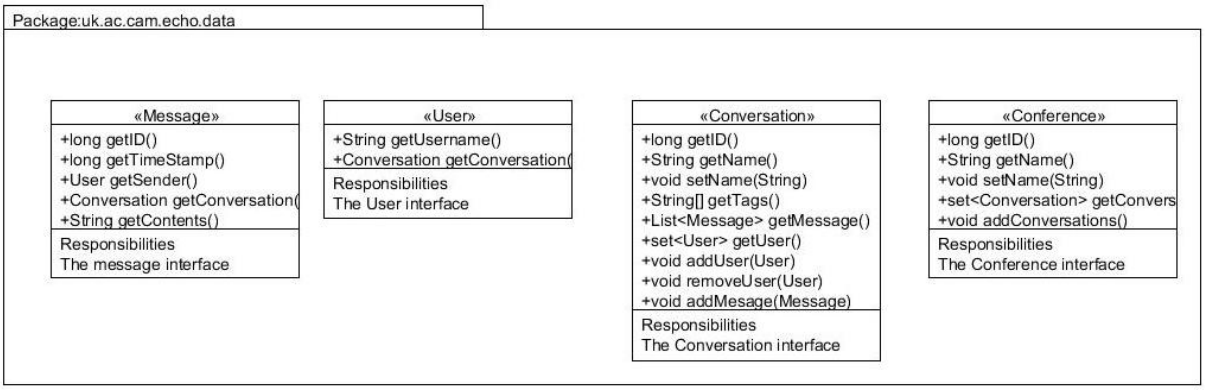

Figure 1.3: UML diagram depicting a more detailed structure of the data package.

responsible to provide at least an interface that the remainder of the group can use, and if possible an implementation. The data will be modeled to be **persistent** and easily **storable** in a database.

1.3 Module dependencies

All modules described above are dependent on the "data" package to provide a unanimous representation of data within the project; otherwise:

- the server will have no dependencies;
- the client library will depend on the server implementation (determining the manner in which the client must communicate to it);
- the client extensions will depend on the client library implementation (that they will rely on to do the low-level server communication on their behalf).

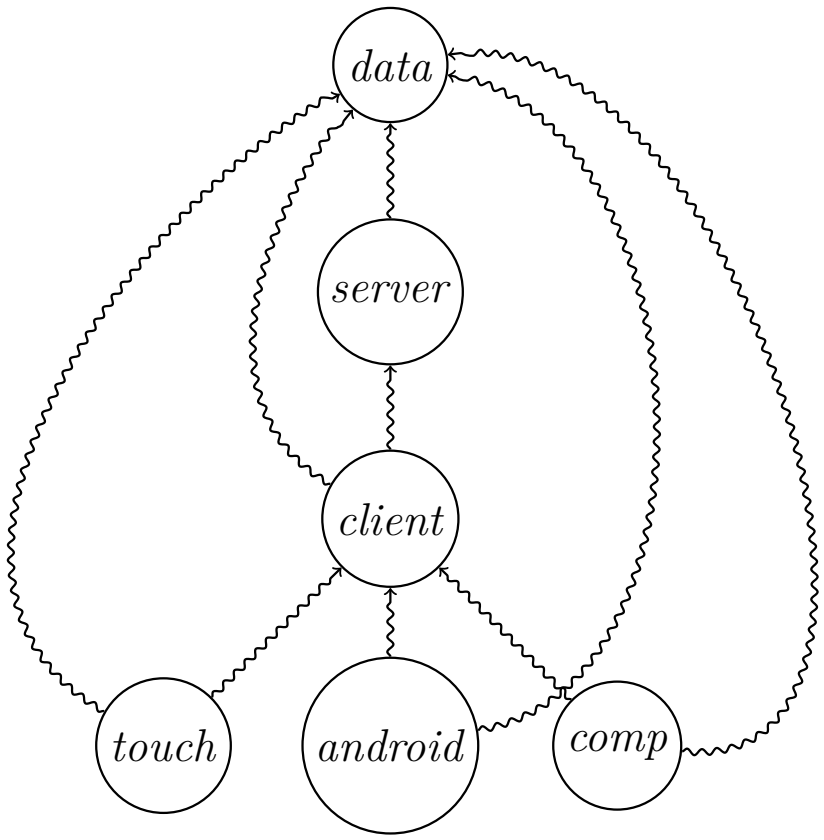

Figure 1.4: Graph diagram depicting the dependencies between modules in the project.