1 The System

The Virtual Disease application is divided into two modules: a client application, which runs on a phone and a server application. The client application simulates the transmission of diseases between devices. It also gathers information such as Bluetooth and GPS data and sends it to the server. The server application receives this data and stores it in a database for further processing.

1.1 Disease Model

The disease model with which the spreading of diseases is simulated is a simple SEIR model. In this model, each device is originally susceptible to a disease. Once it is infected by another device, it becomes exposed for a specified time. Whilst it is exposed, a device has the disease but cannot yet infect other devices. Once the exposed duration has run out, the device becomes infectious for a specific time. Whilst it is infectious, the device can infect other devices – each disease has an associated infection probability which indicates the likelihood that another device will be infected. Once the infectious duration has run out the device is recovered from the disease and cannot be reinfected.

The SEIR model can be used to simulate SI diseases by setting the exposed duration to 0 and the infectious duration to a really large number (for example, 2000 years). SIR disease can be simulated by simply setting the exposed duration to 0.

2 Development

2.1 Directory Structure

All code was developed using the NetBeans IDE (version 6.8) and is stored as a NetBeans Project. The client and server applications are stored as two separate NetBeans projects: `Virtual Disease`, which is a Java ME project and contains the code of the client application and `Virtual Disease Server` which is a standard Java SE project and contains all code required for the server.

Each project directory contains several subdirectories. These are:

- `build/` contains temporary files used for compiling the application
- `dist/` contains distributable content such as a JAR file of the application, extra libraries and documentation
- `nbproject/` contains project meta-data
- `src/` contains the source code of the application

2.2 Setting Up NetBeans

When developing the application, using NetBeans is recommend. NetBeans is available for free from `http://netbeans.org`. When selecting the version to download, download the 'Java' version (not 'Java SE') as this comes bundled with everything needed for mobile development using Java ME such as the Sun Wireless Toolkit. Simply download the executable and open it to begin the installation process. When prompted whether you wish to install Glassfish and/or Tomcat, select `no` as these are not required. Once the application is installed, open the project by selecting *File*, *Open Project* and the location where the application is stored.

2.3 Compiling

To compile the project, open the *Projects* tab (available in the *Windows* menu at the top), right-click the project and select *Clean & Build*. A distributable version will then be compiled and available in the `dist/` folder.

2.4 Signing

After compiling the client application, it is recommended that the application is signed before it is deployed as this will allow the user to grant more privileges to the application and thus reduce annoying prompts on the screen.

In the root of the client project folder, a keystore containing the Computer Lab Thawte Certificate is located under the name `keystore`. Signing should be enabled by default – when selecting *Clean & Build* a prompt should appear asking for the store password and afterwards the key password. The password for the store is `gg7sdg2j` and the password for the key is `uh873dhddf`.

To change any signing related settings in NetBeans, right-click the project, select *Properties*, *Build*, *Signing*. Here signing can be disabled or an alternate store or key selected.

2.5 Deployment & Execution

2.5.1 Client

To deploy the phone client application, copy it from the `dist/` folder to applications folder on the device. On Nokia 6730 the folder is located under `C:/Data/Installs`; it can easily be accessed via Bluetooth. After copying it onto the device, install the application. On a Nokia 6730, this can be done by accessing *Menu*, *Applications*, *Installations*, *App. mgr.*, *Installation Files* and selecting the application. Follow the instructions on the installation screen until the application is installed. After installation, the application can be launched by opening *Menu*, *Applications*, *Installations*, *Virtual Disease*.

2.5.2 Server

To deploy the server application, secure copy (scp) can be used. Simply change into the directory containing the distributable (ie. `dist/`, open a terminal window and enter the command `scp -R * www-fluphone:<target directory>`. Then type `ssh www-fluphone` and type `cd <target directory>`. Then run the command `java -jar <application name>` and the server will run.

2.6 Code Documentation

The code of both applications is documented using inline comments as well as a HTML based Javadoc that can be viewed in any browser. Javadoc refers to documentation that is automatically generated from specially marked comments and formatted in HTML as well as the tool used to generate it. The Javadoc of each project is located in `dist/doc`. Open the `index.html` file to open the start page of the documentation.

Comments which contribute to the Javadoc are indicated by appending an extra `*` behind `/*`. The code snippet below illustrates how Javadoc comments are used.

```java
/**
 * The foo method which does something useful.
 * @param bar some bar object
 * @return true if foo is bar; false otherwise
 */
public void foo(Object bar)
{
...
}
```

Anytime Javadoc comments are changed, the Javadoc has to be recompiled. This can be done in NetBeans by right-clicking the project and selecting *Generate Javadoc*. A detailed explanation of Javadoc is out of the scope of this document; to find out more about Javadoc refer to `http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html`.

3 The Database

The server application stores all the data it gathers in a database for further processing.

3.1 Current Set-up

The database currently in use to store the data is a Microsoft SQL Server 2008 database. We connect to this using the open source jTDS driver. The log-in data used for this database is:

- Database: `victoria.ad.cl.cam.ac.uk:1433/FLUPHONE`
- Username: `fluphone`
- Password: `getwell00`

These credentials can easily be changed by editing the relevant constants in the `thread.SessionHandler` class.

3.2 Structure

This section illustrates the structure of the database. The database uses four tables: `contact`, `disease`, `infection` and `position`. The structure of the tables is explained in this section.

3.2.1 `contact`

Records any bluetooth encounters made. Contains 3 fields: `timestamp` (`bigint`); `device` (`varchar`); `address` (`varchar`).

3.2.2 `disease`

Records information about various diseases. Contains 4 fields: `name` (`varchar`); `exposed-duration` (`bigint`); `infection-duration` (`bigint`); `infectious-probability` (`decimal`).

3.2.3 `infection`

Records infections between devices. Only the first time a specific disease is transmitted to a specific phone is valid. Further transmissions of the same disease to the same device are irrelevant. Contains 4 fields: `timestamp` (`bigint`); `device` (`varchar`); `address` (`varchar`); `disease` (`varchar`).

3.2.4 `position`

Records the location of a device at a given time. Contains 4 fields: `timestamp` (`bigint`); `device` (`varchar`); `longitude` (`decimal`); `latitude` (`decimal`).

3.3 Changing the Database Driver

The application supports any relational database that has a JDBC compliant driver. The set up an alternate driver, simply change the relevant log-in credentials in the `thread.SessionHandler` and alter the contents of the String `DRIVER` to contain the class path of the new driver.

4 Data Logs

The application keeps a copy of all its logged data in the directory `E:/log.xml`. Additionally, it stores information about all diseases it has received in the file `E:/diseases.log`.

4.1 `log.xml`

This file stores all GPS and Bluetooth measurements made by the device, as well as any devices it has infected.

The file begins with the root tag `log`. The `mac` attribute specifies the MAC address of the device that the application is running on. The `log` tag has can have any number of three types of child tags - `measurement`, `infection-received` and `infection-sent`.

The `measurement` tag indicates the values of a measurement: this can include either GPS or Bluetooth data or both. If the element contains GPS data, the attributes `longitude` and `latitude` will be set to GPS coordinates respectively. If the element contains Bluetooth data, it will have numerous child elements of type `mac` indicating the MAC addresses of Bluetooth devices that were observed by this device. Every `measurement` tag must have an attribute named `timestamp`, which indicates when the measurement was taken in milliseconds from 01.01.1970.

The `infection-received` tag indicates that the device has received a disease. This tag has 5 attributes: `timestamp`, which indicates when an infection was received (in milliseconds from 01.01.1970); `name`, which indicates the name of the received disease; `exposed-duration`, which indicates how long the disease will remain in the exposed state before becoming infectious, in milliseconds; `infectious-duration` which indicates how long disease remains infectious after leaving the exposed state, in milliseconds; `infection-probability` which indicates how like the disease is to infect any device in range. If the first fie

Finally, the `infection-sent` tag indicates that the device has successfully infected other devices, with one or more diseases. The `infection-sent` element has one type of child element, of which it can have one or more, named `disease`. The `disease` element contains information about a specific disease. It has one or more child elements of type `mac` which contain the MAC addresses of devices which were infected with the disease. Each `mac` element has a `timestamp` attribute indicating when the infection occurred, in milliseconds from 01.01.1970. The `disease` element has 4 attributes describing the disease, in very similar way to infection `infection-sent`: `name`, which indicates the name of the received disease; `exposed-duration`, which indicates how long the disease will remain in the exposed state before becoming infectious, in milliseconds; `infectious-duration` which indicates how long disease remains infectious after leaving the exposed state, in milliseconds; `infection-probability` which indicates how like the disease is to infect any device in range.

4.2 `diseases.log`

This file is essentially a flat file log which contains information about any diseases received in the form of a semicolon delimited list. If the file exists but is empty, the device is healthy. Disease are entered into the file in the following form: `<time of disease reception>:<disease name>|<exposed duration>|<infectious duration>|<infection probability>|<disease history>;`

`<disease history>` is a comma delimited list, which each item is the form `<device>@<timestamp>`. Each item indicates that the disease was received at that device (from the previous device in the list) at the the given time.

4.3 Transmission

The logged data can optionally be transmitted to a server but does not have to be, allowing for both centralised and decentralised experiments. To disable the transmission of data, simply select *Options*, *Data Transmission*, *Disable 3G* in the application.

To alter the URL and port of the server the client application sends its data to, edit the String `URL` in the class `thread.DataTransmission`.

5 Advanced Application Functionality

5.1 Automatic Restart

One way to significantly reduce the strain on the Bluetooth chip and thus avoid the notorious `OutOfMemoryError` (see *Known Issues*) is to let it restart itself using the push registry. If this feature is enabled, the application will periodically register itself to be started after a specified time in the future. It will then quit and eventually, when the specified time is reached automatically start itself again.

To enable this feature, set the constant boolean `ENABLE_AUTO_RESTART` in the `thread.DataCollection` class to `true`. The time after which the feature will be invoked and to adjust how long the gap is between closing and restarting the applications can be set by adjusting the constant integers `RESET_TIMEOUT` and `RESET_WAITING_TIME`.

Note: the application does not currently store its settings, so we recommend that the options menu is disabled when this feature is used.

5.2 Disabling the Options Menu

Occasionally, it may be better if the user cannot changed any of the options the applications offers. To disable the options menu, on line 184 of `gui.WindowManager` simply comment out the line

```
mainMenu.addCommand(OPTIONS_COMMAND);
```

You may wish to change the default settings for the available options. This can be done by changing the values of the booleans on lines 181–184 in the constructor of `thread.DataCollection`:

```
// Set default settings.
enableLowPowerMode = false;
enableGPS = true;
enable3G = true;
```

5.3 Hardcoded Diseases

Diseases can be hardcoded into the application and set to spawn on the first run on the device with a specified probability. This is very useful for decentralised experiments as it allows us to reliably spawn diseases on devices.

5.3.1 Adding a Disease

To add a disease, the method `void loadDiseases()` in `thread.DiseaseHandler` needs to be edited. Simply fill the disease information (`<name>`, `<exposed duration>`, `<infectious duration>`, `<infection probability>`) into the code snipped below:

```
if (random.nextDouble() < SEED_PROBABILITY)
{
  long timestamp = new Date().getTime();

  DiseaseInformation d = new DiseaseInformation("<name>",
    <exposed duration>, <infectious duration>,
    <infection probability>, deviceMac, timestamp);

  ActiveDisease a = new ActiveDisease(d, timestamp);

  activeDiseases.addElement(a);
}
```

Copy the code snippet and paste it into the section beginning on line 382 of `thread.DiseaseHandler` as illustrated below.

```
if (!file.exists())
{
  // Paste hardcoded disease(s) here.
}
```

5.3.2 Changing the Seed Probability

To alter the probability with which a disease will be spawned on a device, simply change the `SEED_PROBABILITY` in the `thread.DiseaseHandler` class. To disable the seeding of hardcoded devices, simply set the value of this constant to 0.

5.3.3 Currently Hardcoded Diseases

Table 1 lists the diseases that are currently hardcoded into the application.

Disease Name	Exposed Duration	Infectious Duration	Infection Probability
Base line	0	31536000000	1.0
SARS	86400000	108000000	0.8
Flu	172800000	216000000	0.4
Cold	259200000	432000000	0.2

Table 1: Currently hardcoded diseases. Durations are in milliseconds.

6 Known Issues

6.1 Out Of Memory Error

Due to a problem with Symbian OS, the Bluetooth chip will run out of memory after several hundred connections and/or service scans and refuse any further connections by simply throwing an `OutOfMemoryError`. Unfortunately, there is no fix for this at Java ME level. After this error occurs once, it will occur again and again: simply restarting the application is not sufficient. Instead, the Bluetooth chip itself needs to be fully reset which is only possible by restarting the phone itself.

7 Future Improvement

7.1 Adding cryptography

The client application currently sends all it data in plaintext to the server. This is of course not optimal. One option is to add HTTPS. Whilst this is easy to implement on the client side (as all the developer needs to do is add an 's' on the client side, the Java ME handles the rest), on the server side this requires the use of third party libaries etc. and significantly increases traffic. An alternative would be to just have the client generate a symmetric session key and encrypt it under public key cryptography and send it to the server. After the server has decrypted it, both sides could use the symmetric key to communicate. Advantages of this approach are that most phones support symmetric key cryptography. Unfortunately, RSA or a similar public key cryptography scheme would have to be implemented from scratch or through a third party library. Another problem is that phones lack secure random number generators: libraries that provide this functionality seem to be very sparse.

7.2 Improving Database Connection Handling

Currently every thread in the server application creates and eventually closes a database connection. Whilst this is fine for trials that only contain 50 to 100 phones, it does not scale very well. The easiest way of improving the scalability is by adding a thread pool.

7.3 Improved Error Handling

The server application currently just exits if an error occurs. Future versions of the application should implement better error handling.

7.4 Adding Persistent Storage of Settings

The settings selected in the phone application are currently not stored persistently, which means a user will have to reselect them every time they open the application. Future versions of the application should store settings.