Distributed and Mobile Programming Paradigms
Academic Year 2011-2012
Exam Requirements for the WPO part of the course

A urban game for collaborative noise mapping

Deadline 11 June 2012.

Delivery Both the documentation and the code should be delivered via the drop box of PointCarre in the form of a <name>-project.zip file on the day of the deadline **before 16:00**.

Material The project material is available as usual in the Documents folder at PointCarre, and at `http://soft.vub.ac.be/amop/teaching/dmpp`.

1 Assignment

The purpose of this project is to implement a game for *collaborative noise mapping*. The game consists of two teams which have to map the noise pollution of a particular area of a city at a particular time of the day (e.g. the VUB campus between 9 and 10 in the morning). The teams compete to cover the area in the least possible time and earning the largest number of points. Points are scored by visiting —and measuring the noise of— the reachable locations of the area, called "spots". Only the first *player* who visits a spot receives a point. Extra points can be scored by reaching a few "join spots" which are predefined spots randomly placed over the area at the beginning of the game. A join spot gives points only to the members of the first *team* that reaches it (i.e. all the members of the team have to visit the join spot at the same or different times).

The teams start at the same spot and their participants can start walking in random directions. During the game, the players collect the measurements and points they score when visiting the different spots of the game area. The game ends when the map is fully covered by the two teams and/or when the duration of the game is reached.

To measure the noise, you can assume that the game will use a participatory sensing application such as NoiseTube. NoiseTube allows mobile users to measure noise with their Android phone while they move about. The application associates noise data with GPS coordinates, and displays it on a map as shown in Figure 1 (red dots = loud, green dots = quiet). More information about NoiseTube can be found at its website, `http://noisetube.net`. As we explain in Section 3, in this project you will either use NoiseTube or a simulation of it depending on the profile you choose.

2 Requirements

The game is a multi-player mobile application. Each participant of the game has a (mobile) device running the application. Here are the requirements for the distributed design of the game:

Peer-to-peer communication Interactions among participants should happen in a peer-to-peer fashion. In this project, you can assume that a WiFi network will be available all over the game area. However, two devices are said to be connected only if they are at the same spot at the same time. Figure 1 illustrates the normal and join spots for a game at the VUB campus. Every spot is defined as a square delimited by two GPS coordinates indicating

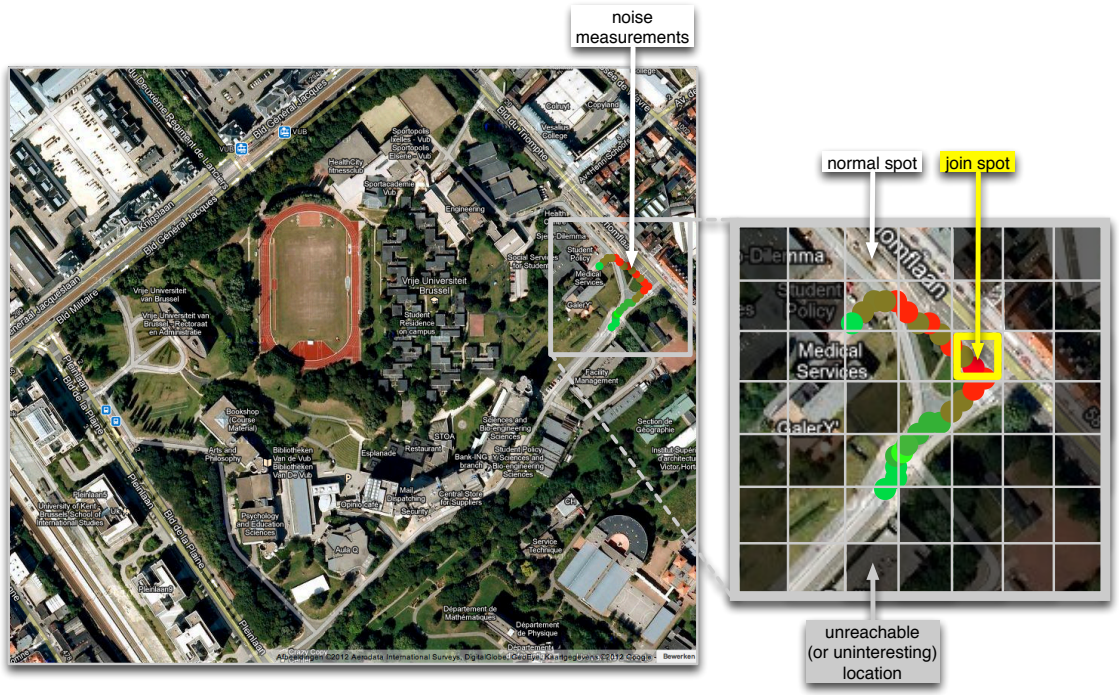

Figure 1: A game for collaborative noise mapping at the VUB campus.

its beginning (X, Y) and its end $(X + \Delta X, Y + \Delta Y)$. The information about spots will be calculated by the application before starting the game. Join spots are randomly put on the map. The application only reveals join sports when the players actually visit those spots (i.e. players will not know up front where the join spots are located).

There is no global knowledge of the area. The participants can only know about each other's collected data when they meet at some spot. Only then they can exchange data, allowing each participant to construct a (partial) map of the measured area. As such, the participants can share information about visited spots and learn where join spots are located. This also allows the participants to correct their score as they may learn that somebody else reached a spot first, and to calculate how many members of their teams have already passed by a join spot.

You can assume that when the game is over, the participants come back together at the location where the game started. All measurements are collected and the data is put together so that final gaps and scores become clear.

Network failure handling You must assume that *every* computational unit in the network can fail at *any* point in time. Team members meeting at some spot will take the time to fully synchronise their collected data. In this case, interruptions of the data exchange due to disconnections should not hamper the normal functioning of the game application.

2.1 Extra requirements

Extra requirements are not compulsory but rewarded with extra bonus marks. Some examples are the following:

- You can add additional kinds of spots or objects to the game which benefit or damage the players who reach them. Some of these objects can be "contagious" in the sense that they propagate to other players (indefinitely, or within some time or space frame).

- You can also add a recommendation service which suggests what to visit next according to the player's collected information.

Being creative and implementing additional requirements of your own, or any of the extra requirements to the project is appreciated.

2.2 Technology

The game should be implemented in AmbientTalk. You can use any of the language abstractions seen in class or during the lab sessions for designing the application. You can also exploit the symbiosis between Java and AmbientTalk to make use of Java classes for e.g. your data structures such as a Hashmap. However, all distributed communication **must** be implemented in AmbientTalk. In other words, you **cannot** use other technologies (e.g. Java RMI) as your distributed computing framework.

3 Project Profile

In order to implement the application, you will choose one of the following profiles:

Profile A: Scientific Profile. The distributed design is the most important part of your project, and the hardware platform where the application will be deployed is not that relevant. The emphasis is on the design of the application, and the correctness of the distributed interactions. You will use the version of AmbientTalk for desktop machines, e.g. by means of the Eclipse plugin as done in the lab sessions.

In this project profile, you can use the map given in lab session 5 of the course as a replacement for simulating the recording noise of an actual participatory sensing application. Recall that the Java GUI given during the lab session consists of a map in which your location is simulated while you drag the mouse. You will need to update this GUI to attach mock noise data to each location.

Profile B: Applied Profile. The distributed design is an important part of your project, but the application will be effectively deployed on the Android platform. The emphasis is on the correct working of the application on the Android platform and a usable user interface. You will extend the NoiseTube application with the gaming functionalities previously described. This means that you can reuse noise measurements that the application produces as well as its map GUI. We will provide you with an Android project from which you can start. The project already contains NoiseTube and the AmbientTalk android core library, it is already setup in a similar way as in lab session 3. Please read the appendix of this document to get further details about the installation and use of this Android project.

In order to facilitate the testing, you should adapt the application to work with simulated location information (i.e. you will not be asked to go outside to test the application, but you can do so if you are a concerned citizen ;). The simulated location information can be obtained at least in two ways:

1. By extending NoiseTube's map GUI to indicate your location using the device's touch screen. This is similar to the interactions with the Java GUI that we described in Profile A, but you will simulate that you change your location as you draw on the map with a finger.
2. By using NoiseTube's map GUI as it is now implemented, and loading mock location data through the Emulator Control of Android's Dalvik Debug Monitor Server (DDMS). DDMS allows you to load locations one by one or in a batch file (GPX).

You should mail no later than **16 April 2012** which project profile you would like to do. If you choose Profile B, please let us also know if you owe an Android device so that we can distribute devices if necessary.

4 Testing and Report

You should define a number of test cases for your application. Keep in mind that these test scenarios should test distributed interactions (e.g. testing that players receive each other's collected measurements). In order to test your applications in the face of network disconnections, you can use the different AmbientTalk constructs used during the lab sessions (e.g. the `disconnect:` form).

You should write a small report about this project. This report must be **no longer than 7 pages** (excluding figures) and serves as a guide for understanding your project. The report should include:

1. An overview of your application and how the implementation fulfils the requirements.
2. The important cases and design choices (w.r.t. distributed aspects) considered in the project.
3. Description of test scenarios and behaviour of your application.
4. A small "manual" about how to run and test your application.
5. Which IdeAT plugin version or AmbientTalk build you used (if you choose Profile A).

5 Evaluation

The project will be evaluated mostly on good distributed design. Each profile will be evaluated as follows:

Profile A: distributed design 80% stability and usability 20%
Profile B: distributed design 50% stability and usability 50%

What is **important** for both profiles:

- You have to display the collected data of other players you encounter in the game area. You are encouraged to reuse the aforementioned Java GUI or Android project to this end. We strongly advise **against** building your own GUI as the focus of the project is the distributed design.
- You should aim to fully exploit a peer-to-peer architecture and a fault-tolerant design.
- The testing should be focused on trying the application from a distributed point of view.
- Quality and structure of the code is **very** important.
- Make sure that Java code (if any) is compatible with **version 1.5**.
- This assignment is to be made **individually**. Cooperation is not allowed in any form.
- Problems with the software tools (e.g. Eclipse plugin, Android setup, etc.) and devices should be reported **no later than** two weeks before the deadline, i.e. **28 May 2012**.

Please do not hesitate to contact the assistants for further questions.
Elisa Gonzalez Boix, e-mail: egonzale@vub.ac.be, office: 10F731
Jorge Vallejos, e-mail: jvallejo@vub.ac.be, office: 10F724

For further information about NoiseTube:
Ellie D'Hondt, e:mail: eldhondt@vub.ac.be, office: 10F730

Good luck!

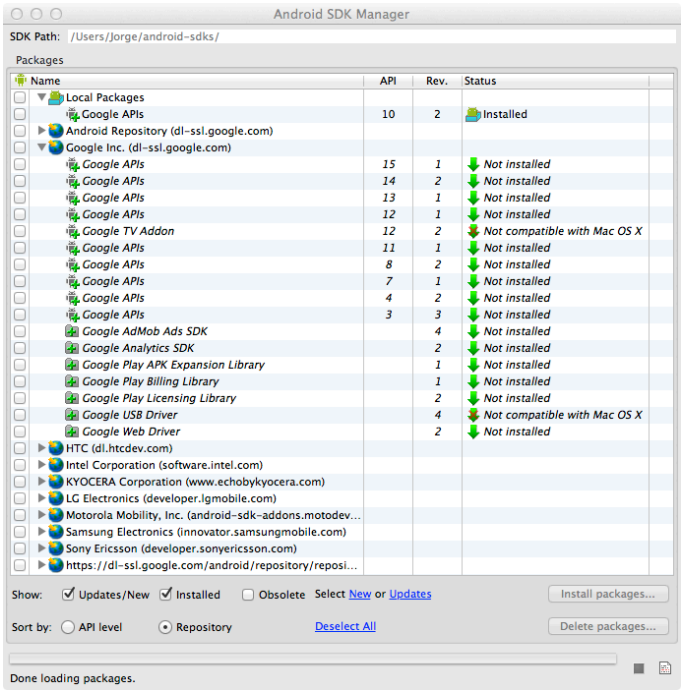

Figure 2: Installing Google APIs.

A The NoiseTube AmbientTalk Android project

This appendix contains some extra information (only relevant for those choosing Profile B) about the Android project from which you can start.

A.1 Installing the Android project

You need to import the project into Eclipse as done in lab session 3. Note that the project contains the android-core AmbientTalk library. If you use the same Eclipse workspace than during the sessions you will not need to import it again.

The Android project requires using Google APIs as the Project Build Target. This is because NoiseTube uses a Google Map View for showing a map to the user, which is only included in the Google APIs. The project is currently set to Google API 2.3.3, but if you have not installed this API the project will give compilation errors.

To install Google APIs you need to make sure that you are running the latest version of the ADT plugin(17.0.0.v201203161636-291853) and that in your SDK manager you have installed the latest version of the Android SDK tools (Rev. 17). If you have installed the latest version of the plugin, the Google APIs can be installed as an add-on using the Android SDK Manager. Figure 2 shows where to find this package when you select the option "Repository" in the bottom left corner of the Manager's window. You can also install them when the SDK Manager shows the packages corresponding to the API levels by selecting it within a particular API level.

Choose Google API level 10 or another one which is supported by your Android device. Once installed, you will need to set the properties of the project to use it (Properties -> Android -> choose the Google API level) as shown in Figure 3.

You can find more information about how to install Google APIs at `http://code.google.com/android/add-ons/google-apis/index.html`.

If you are unable to run the project after installing the Google APIs and properly setting the project properties, please contact us. We have detected misbehavior of the application on older

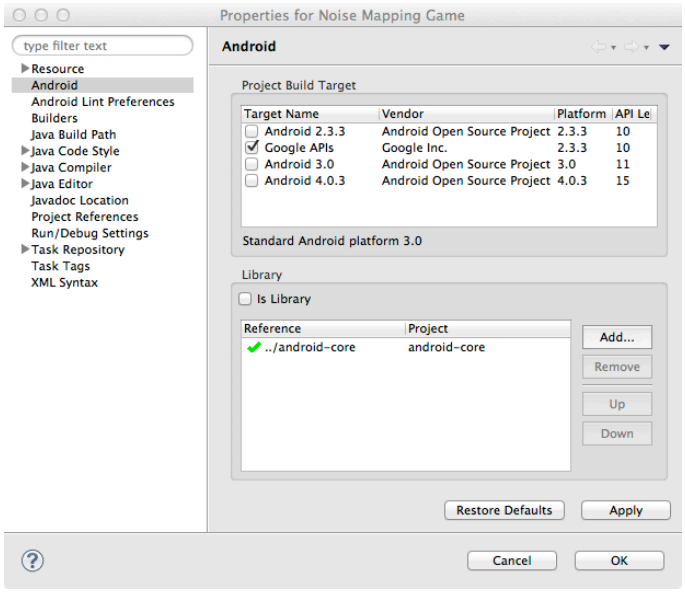

Figure 3: Using Google APIs as Project Build Target.

Android devices (e.g. Xperia Sony Ericson running version 2.1).

A.2 Using the Android project

We now discuss the relevant parts of the NoiseTube AmbientTalk Android project you will need to use/adapt:

MainActivity.java As its name says, it is the main activity of the application. It contains the necessary code to load the **noiseMappingGame.at** file. This activity already stores a reference to the AmbientTalk application in a variable called `game`, so that Android can communicate with AmbientTalk. After calling the `makeNoiseMappingGame` function from **noiseMappingGame.at**, this activity starts recording noise measurements from your phone.

noiseMappingGame.at It is the AmbientTalk file you need to implement to interact with the GUI and remote players. It is stored under the `/assets/atlib/game` folder. noiseMappingGame.at includes the `makeNoiseMappingGame` constructor function that returns a `localInterface` object with the 3 methods described by the **ATNoiseMappingGame.java** interface. The `registerActivity` method setups the connection between Android and AmbientTalk. It is called once AmbientTalk is launched and has evaluated the `makeNoiseMappingGame` function receiving as parameter the activity representing the GUI to which AmbientTalk objects communicate. Right now, it receives a reference to a `MainActivity` instance, but you may need to change this in a later stage of your project, i.e. call it with another activity.

TabMapActivity.java NoiseTube use tabs to organise its different views. The map is defined in the `TabMapActivity.java` class which creates a map-viewing activity which overlays the noise measurements taken by the application. You will need to adapt this activity to display the noise tube measurements of other players in the neighbourhood. In order to facilitate the testing of your application (without having to go outside and record measurements), you can adapt the map view to be able to display a "simulated" path. As explained before, there will be two options to do this:

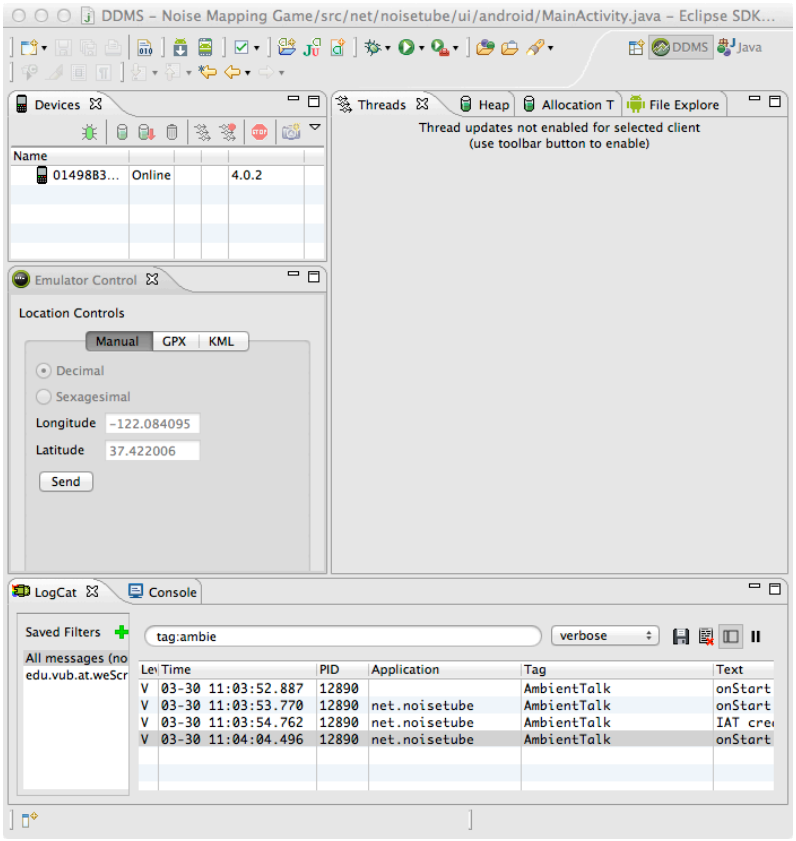

Figure 4: Using DDMS' location emulator.

1. Similar to the GUI of exercise session 5, you could draw a path on the map with your finger. This path represents the locations you are measuring. To this end, have a look at the documentation of the `onDraw` method in the `MapView.java` class.
2. Your location can also be simulated by using Android's DDMS (`http://developer.android.com/guide/developing/debugging/ddms.html`). DDMS provides a location emulator as shown in Figure 4. More details on how to simulate location data can be found at `http://developer.android.com/guide/topics/location/obtaining-user-location.html#MockData`.

ATNoiseMappingGame.java and JNoiseMappingGame.java Similar to the WeScribble project used in the lab session 3, these two classes are the Java interfaces defining the methods used for the communication between AmbientTalk and Android. Currently, it only contains the minimum methods to set up the application. You should complete these interfaces with the necessary methods. We are going to look at them during the evaluation of your project in order to be able to understand your code.