Aether’s notebook

Introduction

Aether's notebook is a logging application for connectivity information. The aim of this project was to develop the software infrastructure to enable the crowdsourcing of such information from Android mobile phones.

Architecture

Aether's notebook is a typical client-server application. The client, running on an Android mobile phone pulls network connectivity information and submits it to the central server which is a cloud hosted application, running on the Google App Engine. The server is responsible for the accumulation of the crowdsourced data.

Figure 1: Client-server architecture

Client

The centre of the client system is a number of sensors that are responsible for collecting different connectivity information. These Sensor Logging Services run on their own threads, periodically scanning for new information.

Any new information is sent to the main Sensor Service (`SensorService`) that controls the sensors and coordinates the logging. This service is responsible for sending the information to a number of logging listeners that can be registered, using the exposed interface, to intercept logging messages.

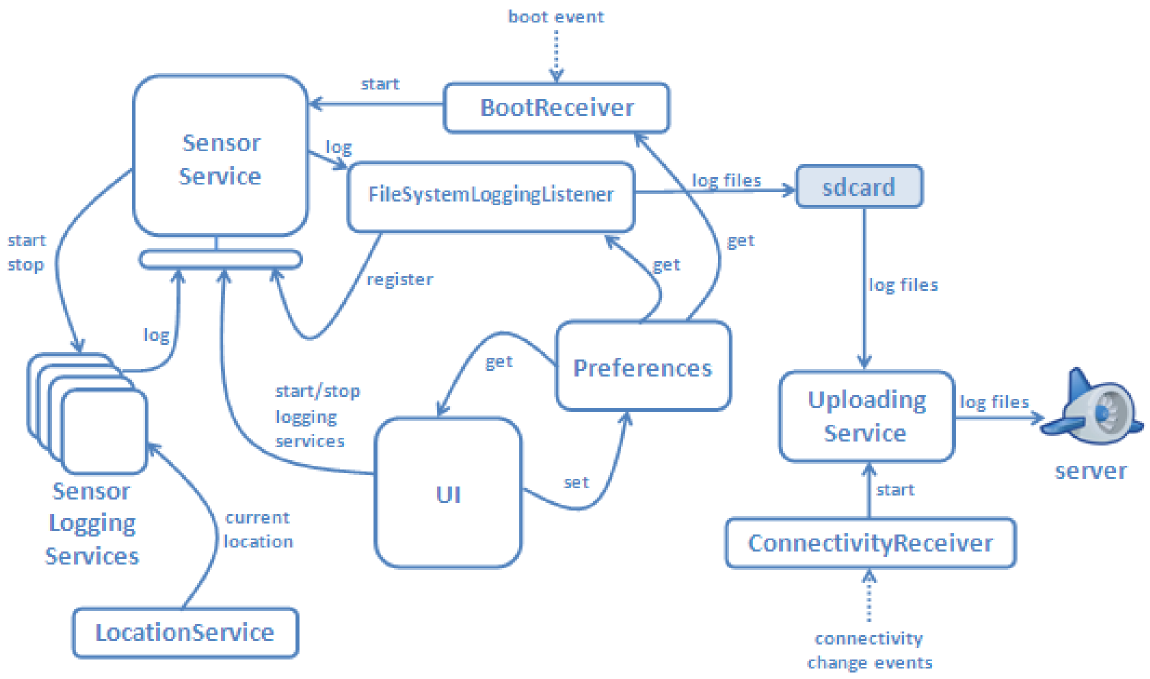


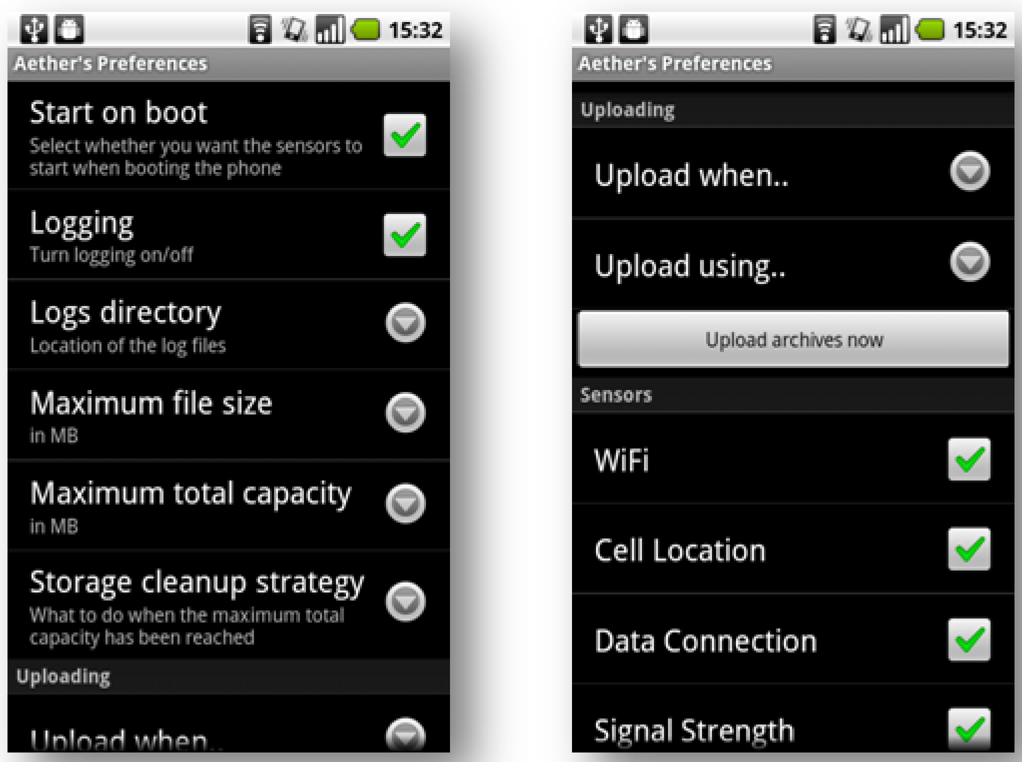
Figure 2: Client design

The default logging listener is the `FileSyStemLoggingListener` service that saves data on the sdcard. This service is also responsible for compressing and archiving old log files.

The compressed log files will end up in the server with the help of the Uploading Service (`UploadingService`) that makes POST requests to the server. A broadcast receiver (`ConnectivityReceiver`) keeps track of the connectivity changes and notifies the Uploading Service when the client has a connection that can be used to upload files.

Another broadcast receiver (`BootReceiver`) is used to start the logging when the phone is booted by listening to the relevant event. This feature can be switched off if the user wishes so.

Preferences


15:32
Aether's Preferences

Start on boot ✓
Select whether you want the sensors to start when booting the phone

Logging ✓
Turn logging on/off

Logs directory
Location of the log files

Maximum file size
in MB

Maximum total capacity
in MB

Storage cleanup strategy
What to do when the maximum total capacity has been reached

Uploading

Upload when..

15:32
Aether's Preferences

Uploading

Upload when..

Upload using..

Upload archives now

Sensors

WiFi ✓

Cell Location ✓

Data Connection ✓

Signal Strength ✓

Figure 3: Preferences screen

A preferences screen allows the user to change a number of settings using the following:

- **Start on boot:** An on/off button that lets the user select whether they want the application to start when the device is booted.
- **Logging:** An on/off button setting the status of the logging.
- **Logs directory**: A text field where the user selects the directory of the log files.
- **Maximum file size**: A text field to set the maximum size the log file can have (set in MB).
- **Maximum total capacity**: A text field to set the maximum size of the logs directory.

- **Storage cleanup strategy**: A list of possible strategies to be followed to cleanup the logs directory in case the maximum capacity is almost reached. Currently, only one strategy is implemented that will simply delete old files.

- **Uploading:**
    - **Upload when**: A list of choices where the user can select when to upload archives to the server. The files will be uploaded either when the user presses the "Upload archives now button", when there is at least one file or when the maximum storage capacity is almost reached.
    - **Upload using**: A list of choices for the user to select which connection to use for uploading files to the server (3G/Wi-Fi).

- **Sensors**: A list of checkboxes to switch the sensors on and off.

Exposed interfaces

Sensors Service

The Sensors service exposes two interfaces:

1. The `SensorServiceControl` interface that provides methods to control the sensors:
    - `List<LoggingServiceDescriptor> getLoggers():` Returns the list of loggers.
    - `void stopLogger(LoggingServiceDescriptor descriptor):` Stops a logger given by the descriptor
    - `void startLogger(LoggingServiceDescriptor descriptor):` Starts a logger given by the descriptor
    - `boolean getLoggerStatus(LoggingServiceDescriptor descriptor):` Gets the status of a logger (on/off).
    - `void startLogging():` Starts the loggers that are
    - `void stopLogging():` Stops all loggers.

2. The `SensorServiceLogger` interface that provides the following methods:
    - `void log(long timestamp, String identifier, String dataBlob):` Receives a log entry
    - `void registerListener(LoggingListenerService listener):` Receives a logging listener to be added in the list of listeners.

Uploading Service

The uploading service exposes a simple `IUploadingService` interface that exposes a single method:

- `boolean uploadFile(String filePath, String serverUrl):` Uploads a file to a server.

Server

The server is responsible for receiving logging data and saving it to a database. For this purpose, a Spring MVC application was created that receives compressed log files, uncompresses them and parses them to retrieve the logging data and, finally, persist them in the database.

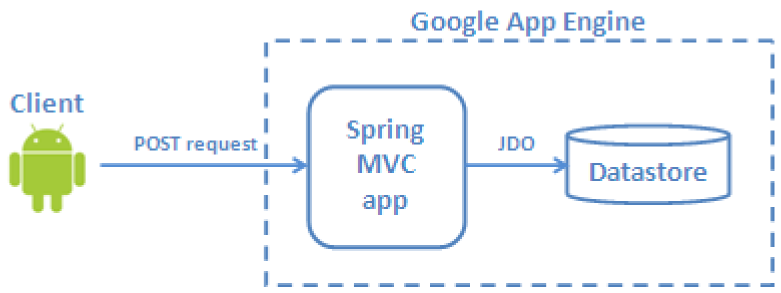


Figure 4: Server design

The Jackson Processor was used to parse the logging data and map them to objects that get persisted in the database.

Model

The main entity of the model is an “Entry”. Entry objects represent a logging entry received from a client and is described by the timestamp of the entry (`timestamp`), the type of entry (`identifier`), the geographical location (`location`) and the data blob (`dataBlob`).

A location is represented by an object of the `Location` class which has the following fields:
`accuracy, altitude, bearing, latitude, longitude, provider, speed, extras`

The data blobs can be a number of different types that implement the `IBlob` interface:
`CellLocationBlob, DataConnectionStateBlob, ServiceStateBlob, SignalStrengthBlob, TelephonyStateBlob, WifiBlob`

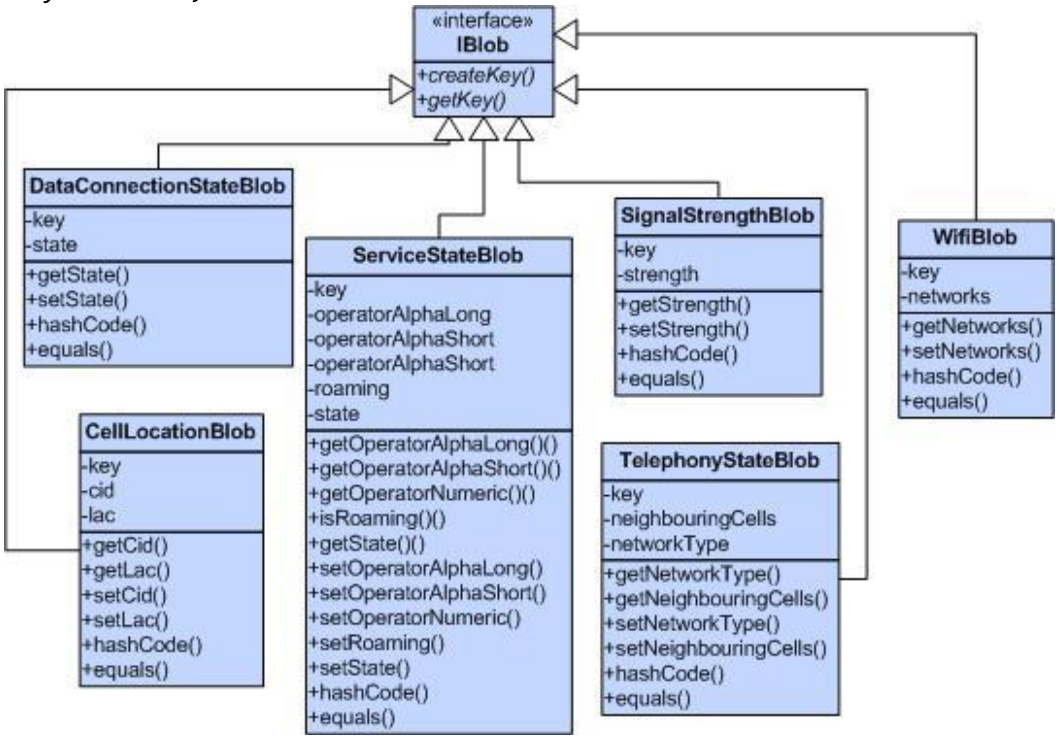


Figure 5: Blob hierarchy

View

[TODO]

Controller

[TODO]