Realtime Project

Mosaddegh, Keiwan
Faculty of Computer Science,
Lund University
ke2476mo-s@student.lu.se

Borglund, Filip
Faculty of Computer Science,
Lund University
fi5685bo-s@student.lu.se

Noorzadeh, Sepehr
Faculty of Computer Science,
Lund University
se1711no-s@student.lu.se

Schön, Maximilian
Faculty of Computer Science,
Lund University
ma5176sc-s@student.lu.se

December 2018

1 Introduction

This is a manual that covers the design of the realtime project, implemented for the course EDAF55. The manual is split up into a server side section and a client side section. These sections are then split up into subsections in which the threads and monitors are described.

2 Usage

2.1 Running the server and client

The server (with main class named `main.java` in package `server`) can be run with 2 arguments: first argument must be an integer number, which will be the port that the server uses. The second argument is the address of the proxy camera (for example `argus-2.student.lth.se`). The camera must have a proxy

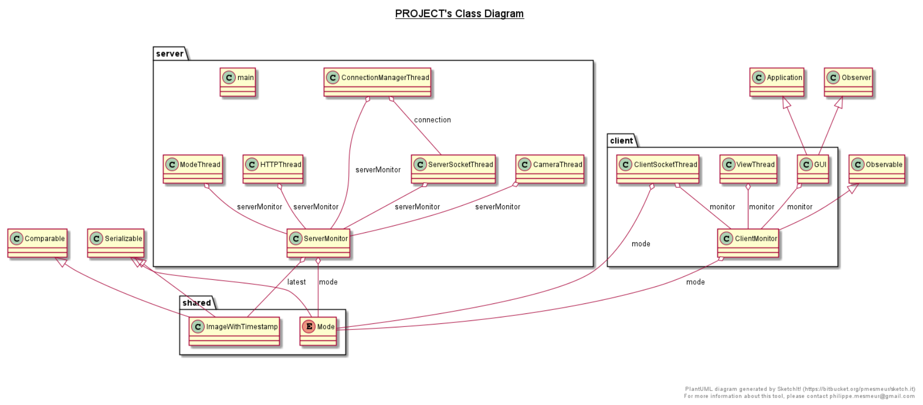

Figure 1: UML Diagram

server running on port 8888 for the server to successfully connect. The client can be run from the class `GUI.java` in the package `client`.

2.2 Using the GUI

The address of the server and the port, separated by a colon, must first be entered in the text field at the top left. The buttons 'Connect server 1' och 'Connect server 2' can be used to connect to the server and display the images in the left or right side of the client. 'Disconnect server 1' and 'Disconnect server 2' disconnects the left/right feed from the server. The radio buttons 'Auto', 'Movie' and 'Idle' update the mode on the client and server. The radio buttons 'Auto-sync', 'Sync' and 'Async' change the sync behaviour on the clientside.

3 Overview

There are three packages, client, server and shared. The shared package is used by both the client and the server. The server package is responsible for the connection with the camera and sending the images to the client. The client is responsible for displaying the images. The two packages communicate with the ModeThread & ServerSocketThread on the server side and the ClientSocketThread on the client side. For an overview diagram see figure 1.

4 Shared

In the package named shared in the project, the data shared between client- and server-side is found. This consists of the images, which are sent from the server side to client side, but also information about what mode (idle, movie, and auto) is set. The data about what mode is set is sent both from client to server, and conversely; that is, from server to client. This is in order to enable the mode auto, where the camera automatically changes modes if it detects movement, to notify the client of the mode changes.

The shared data results in the following classes in the shared package:

- ImageWithTimestamp
- Mode

4.1 ImageWithTimestamp

The name of the class is fairly self explanatory. This class consists of an Image object, and a time stamp. The time stamp, represented by a long with the time when the image was received in milliseconds, is used in synchronization solutions on the client side, but also to compute the delay when transferring an image from server to client. ImageWithTimestamp implements Serializable, which enables the java class to be sent as a bytestream, thus allowing it to become serialized, transferred between server and client, and ultimately deserialized as the same class.

4.2 Mode

Mode consists of *MOVIE*, *IDLE*, and *AUTO*; three named values (enumerators) that are intended to replace for example a collection of integer constants. Mode also implements Serializable with the same reason as for ImageWithTimestamp.

5 Server

The Server package consists of two different types of classes to make sure that we avoid concurrency problems. These two different types are Threads and a Monitor. The Threads offer the concurrency part, while the monitor makes sure that the data these use threads are not accessed simultaneously.

5.1 Threads

On the server side there are five threads in use. These threads are:

- CameraThread
- ConnectionManagerThread
- HTTPThread
- ModeThread
- ServerSocketThread

5.1.1 CameraThread

The CameraThread is the thread with a connection to the camera. The thread fetches pictures either as fast as it can - or it waits for 5 seconds between pictures. It does this by calling the *Thread.sleep()* function with 5000 ms as its' parameter.

5.1.2 ConnectionManagerThread

The ConnectionManagerThread manages the connections from clients. It does this by constantly calling the blocking function *Socket.accept()*. Once it gets a connection it creates a new ServerSocketThread which handles the individual connection.

5.1.3 HTTPThread

The HTTPThread is the thread acting as a HTTP server. What it does is listens for GET-requests and returns appropriate header and image-data. All other requests receive a header explaining that the server did not have the implementation to handle received request.

5.1.4 ModeThread

The ModeThread waits to receive something from the client, namely, a mode. The client only sends Modes, so what we receive from the server is assumed (and typechecked) to be a Mode. The ModeThread calls the method *Object-InputStream.readUnshared()* which is blocking, so the thread is in the blocked state most of the time.

5.2 ServerSocketThread

ServerSocketThread handles sending objects to the client. It receives a socket from the ConnectionManagerThread and uses this for sending the objects. The objects being sent is either of the type Mode or ImageWithTimestamp, which both implement the **Serializable** interface. This allows the objects to be sent using *ObjectInputStream.writeUnshared()*.

5.3 ServerMonitor

The ServerMonitor class protects all of the data that the threads on the server side use. This is mainly the items being sent to and from the client and images received from the camera. To do this we use the functions:

- getImage: Blocking, returns the oldest image from the camera, put in an ArrayDeque.
- putImage: Blocking, adds a new image to the ArrayDeque of Images.

- getLatest: Returns the latest image added into the ArrayDeque.
- newMode: Returns whether the mode has changed and we have not sent the new mode to the client. This happens when the server is in Mode.AUTO and receive movement from the camera.
- setNewMode: States that the mode have changed.
- setMode: Changes the current Mode.
- getMode: Returns the current Mode.

6 Client

6.1 Threads

On the client side of the project two threads are used. These threads are called ClientSocketThread and ViewThread.

6.1.1 ClientSocketThread

The ClientSocketThread has two main purposes. ClientSocketThread establishes a connection with the Server through the use of Java's Socket class. This is done once for every instance of the thread. Once the connection has been established the thread continues to work on its second main purpose. That is receiving images and modes from the server. The thread also sends modes from the client to the server. The images and modes received are stored in the ClientMonitor. Each of the method calls to the monitor are blocking.

6.1.2 ViewThread

The purpose of ViewThread is to take images, delays and timestamps from the monitor and display these in the GUI. All the method calls to the monitor are blocking to ensure ViewThread gets exclusive access while fetching resources.

6.2 ClientMonitor

The ClientMonitor is a subclass of Observable. This is to make sure the GUI updates the view as soon as the monitor receives new images from the server. The monitor stores images in a HashMap mapping from an integer, representing which view (0 or 1, but expandable for more views) the images are to be displayed on, to an ArrayDeque containing images with timestamps. The images are fetched by the blocking method *getImage*. This method also takes care of clearing the image buffers in case synchronization is not necessary. In case synchronization is necessary, the method makes sure the the fetched images have the appropriate timestamp.

Other methods in the monitor involve methods to connect and disconnect to servers, put images in the monitor, change the mode in which the client is running, set and get synch and delay. All of the methods in the Monitor are using the keyword *synchronized* to make sure the code can be accessed concurrently.