EDA040 Camera Project
Reference Manual

Tim Hedstrom (int15the)
Michael Noukhovitch (int15mno)

User Guide

How to Start

To use the proxy camera security system, first start the camera servers:

1. choose a camera number (1-8) which we'll refer to as X and ssh into argus-X.student.lth.se

   ```
   ssh argus-5.student.lth.se
   ```

2. in the camera server's terminal start `proxyserver` on a chosen port (we recommend port 700X where X is your camera number)

   ```
   ./proxyserver 7005
   ```

3. repeat steps 1-2 for another camera (different camera number X)

Then start the client program

4. open a terminal and navigate to the directory with `cameraproject.jar`

   ```
   cd ~/path/to/cameraproject/
   ```

5. add execution privileges to `cameraproject.jar`

   ```
   chmod +x cameraproject.jar
   ```

6. run `cameraproject.jar` with the camera numbers and camera ports that you chose

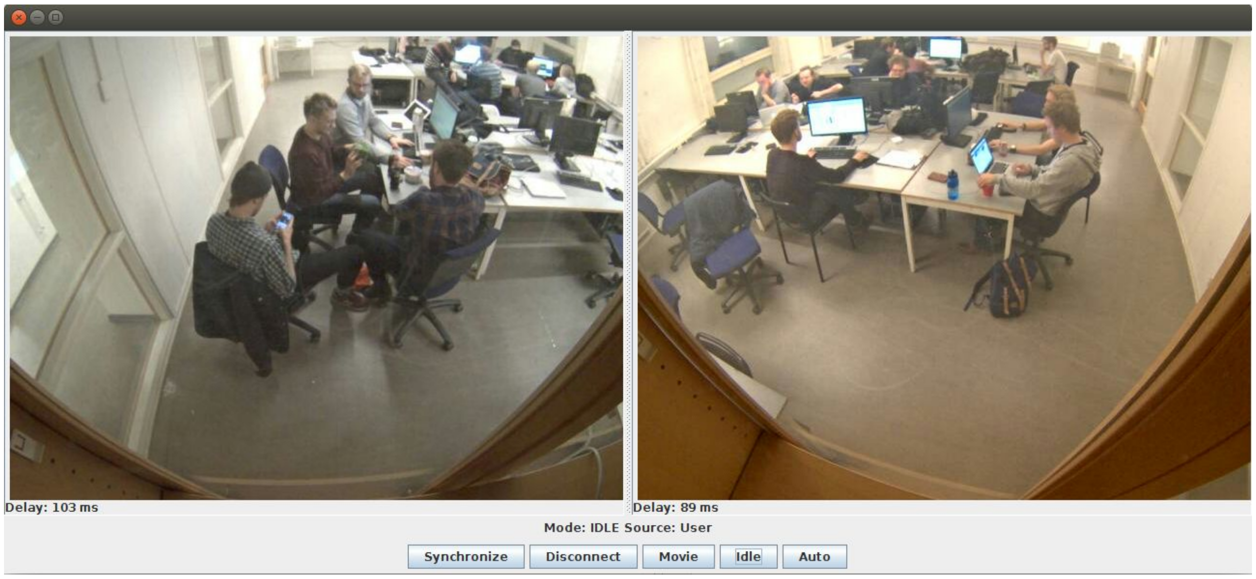
   ```
   ./cameraproject.jar 3 7003 5 7005
   ```

Check the connection using the automatically started test client

7. a test client will be started and will show just an image grabbed from the first camera and have a button to get the next image. click the button and the next image from that camera should appear.

8. if the next image appears, your connection is stable and you can close the test client

How to Use



Connect

1. click `connect` to connect to the cameras and start the stream of images to your client, the button name will then change to `disconnect`
2. click `disconnect` to disconnect from the camera (note that you can reconnect after disconnecting)
3. while connected you can close the window to shut down the cameras (note that if you're not connected to the cameras, you cannot correctly shut them down and release their resources)

Modes

1. there are three modes that you can choose from:
   a. `IDLE`: camera sends images once every 5s
   b. `MOVIE`: camera sends images once every 83ms (approx 12fps)
   c. `AUTO`: camera is in `IDLE` mode until it detects motion, then switches to `MOVIE`
2. the current mode is indicated just under the two images, above the buttons

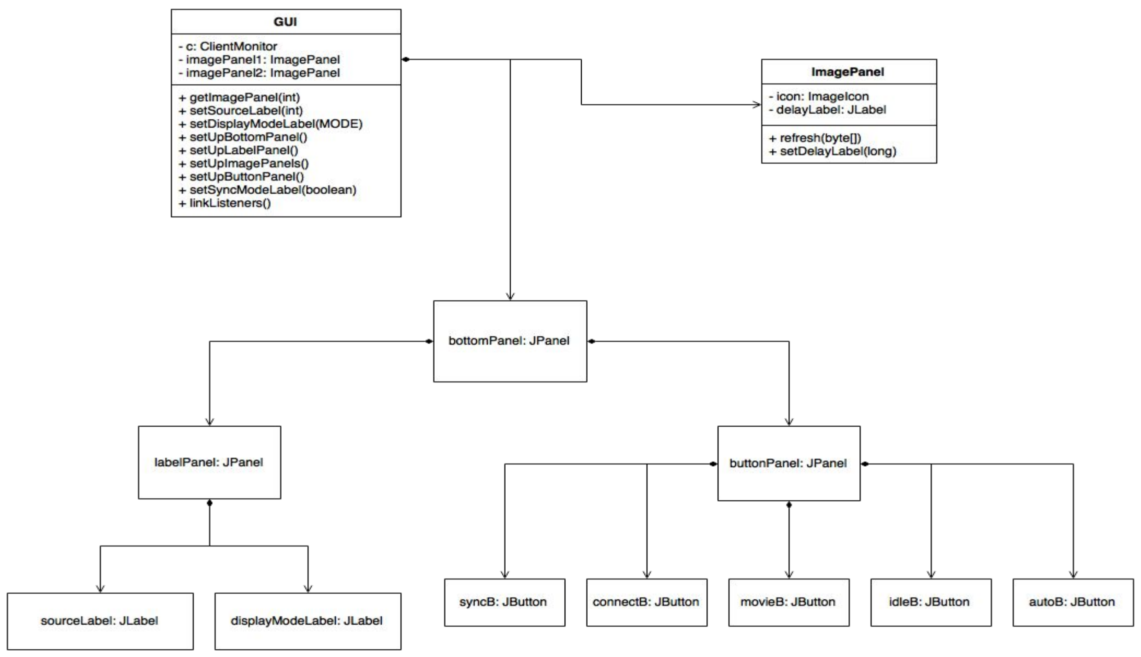
3. the source of the mode (also indicated there) is one of four options:
    a. `INITIAL`: initially set as default
    b. `USER`: the user pressed the corresponding button
    c. `CAMERA 1`: camera 1 detected motion and was in `AUTO` mode, changing the current mode for everything to `MOVIE`
    d. `CAMERA 2`: camera 2 detected motion and was in `AUTO` mode, changing the current mode for everything to `MOVIE`
4. the user can change the current mode by selecting the appropriate button
5. the mode set initially is `AUTO`

Synchronization

1. the two cameras can be in one of two states:
    a. Synchronized: images from the two cameras will appear within 200ms of each other's real time clock. what this means is that if two cameras are trained on the same spot, setting this will make any action in that spot happen synchronously on both camera's outputs
    b. Unsynchronized: images from the cameras are sent to output as soon as they are received
2. If the delay between the cameras outputs is larger than 200ms for three outputs in a row, synchronization will automatically be disabled (since synchronization would seem to be impossible in that situation)
3. Clicking on `synchronize` will change the button to `unsynchronize`, so you can always tell what mode you are in by the text of the button
4. Initially, synchronization is disabled

Description of System

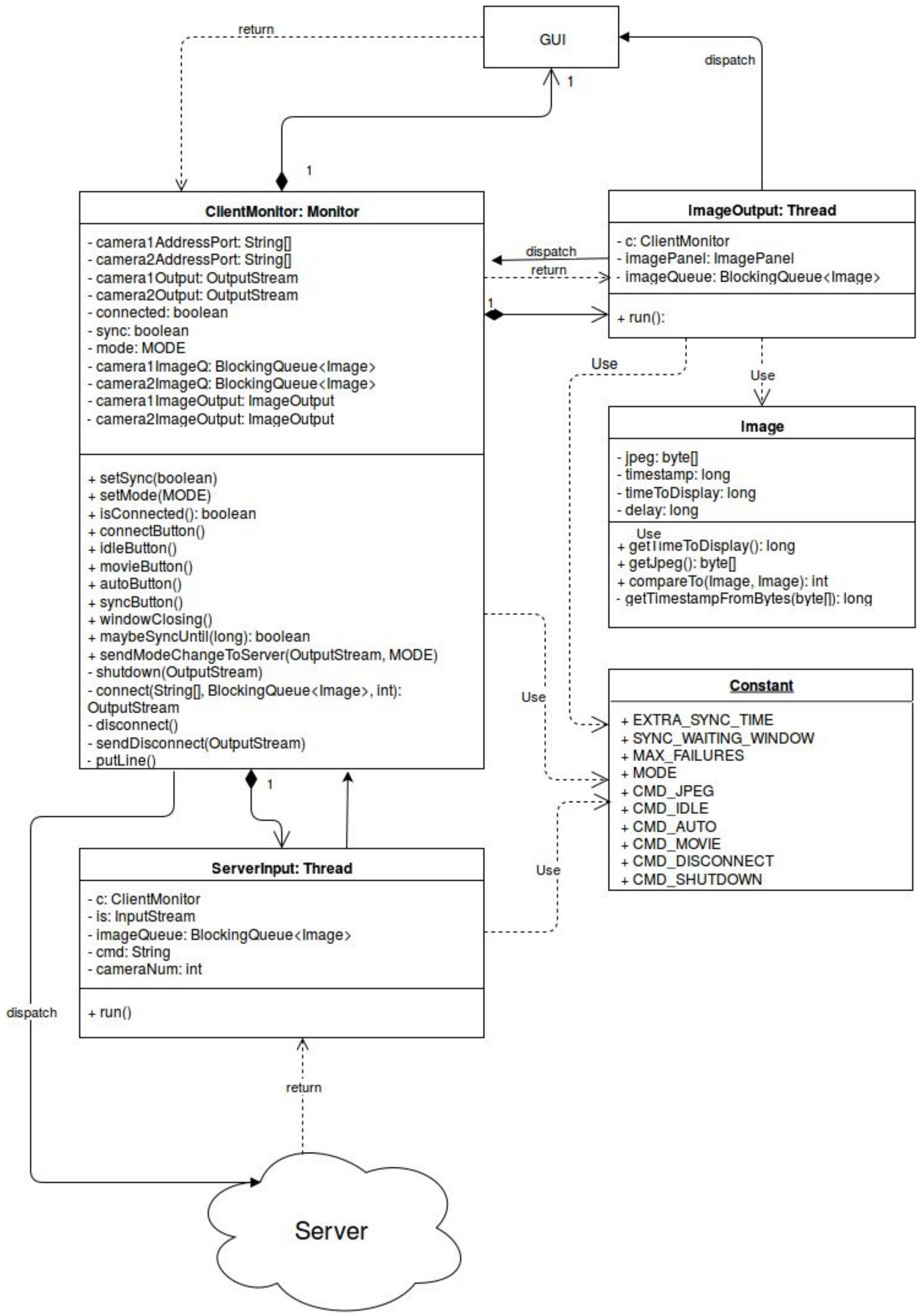
GUI



Upon initialization, the GUI object will create a window with two image panels for displaying the video feed, buttons and labels, and will attach ActionListeners to each of the buttons to interact with the system. The labels and buttons in the GUI are each organized into their own panels. The two images are displayed through two ImagePanel objects. Each ImagePanel displays an image (initially blank, further images displayed by a call to refresh(byte[])) and contains a label intended to display the delay, set by calling setDelayLabel(long). Images will not display until the connect button is clicked. When clicked, the connect button will call connectButton() on its client object, which upon successfully connecting will cause the video feed and delay times to be displayed in the ImagePanels. Clicking the sync button will interact with the client and cause the system to go into synchronous mode, and change the text of the button to “Unsynchronize”.

When the system is in synchronous mode, clicking the sync button, now labeled “Unsynchronize”, will switch the system into asynchronous mode and change the button text back to “Synchronize”. The client is checked to ensure the button text properly reflects the state of the system. The movie, auto, and idle buttons each call on the ClientMonitor to switch to their respective modes when clicked.

Client



ClientMonitor

The client is initialized in ClientMonitor which will set the initial state of the application, start the GUI, start the camera, connect to the server, starts the ServerInput, creates the queues to keep the images sent from the server, and finally starts the ImageOutput processes.

The monitor also allows for processes to send messages to the server (to change the mode, disconnect, or shutdown), and facilitates other processes' wait (specifically the ImageOutput's syncing). The monitor also is what maintains the statefullness.

ServerInput

There is one of these processes for every server (camera), it waits for input from the server (in the form of an InputStream) and handles the two possible commands from the server.

For CMD_MOVIE, it sets the current mode of the client to MOVIE and then propagates that mode to the remaining camera.

FOR CMD_IMAGE, it takes the jpeg and corresponding timestamp sent from the camera and creates an Image object which is placed on the image queue (for that camera) to be used by ImageOutput.

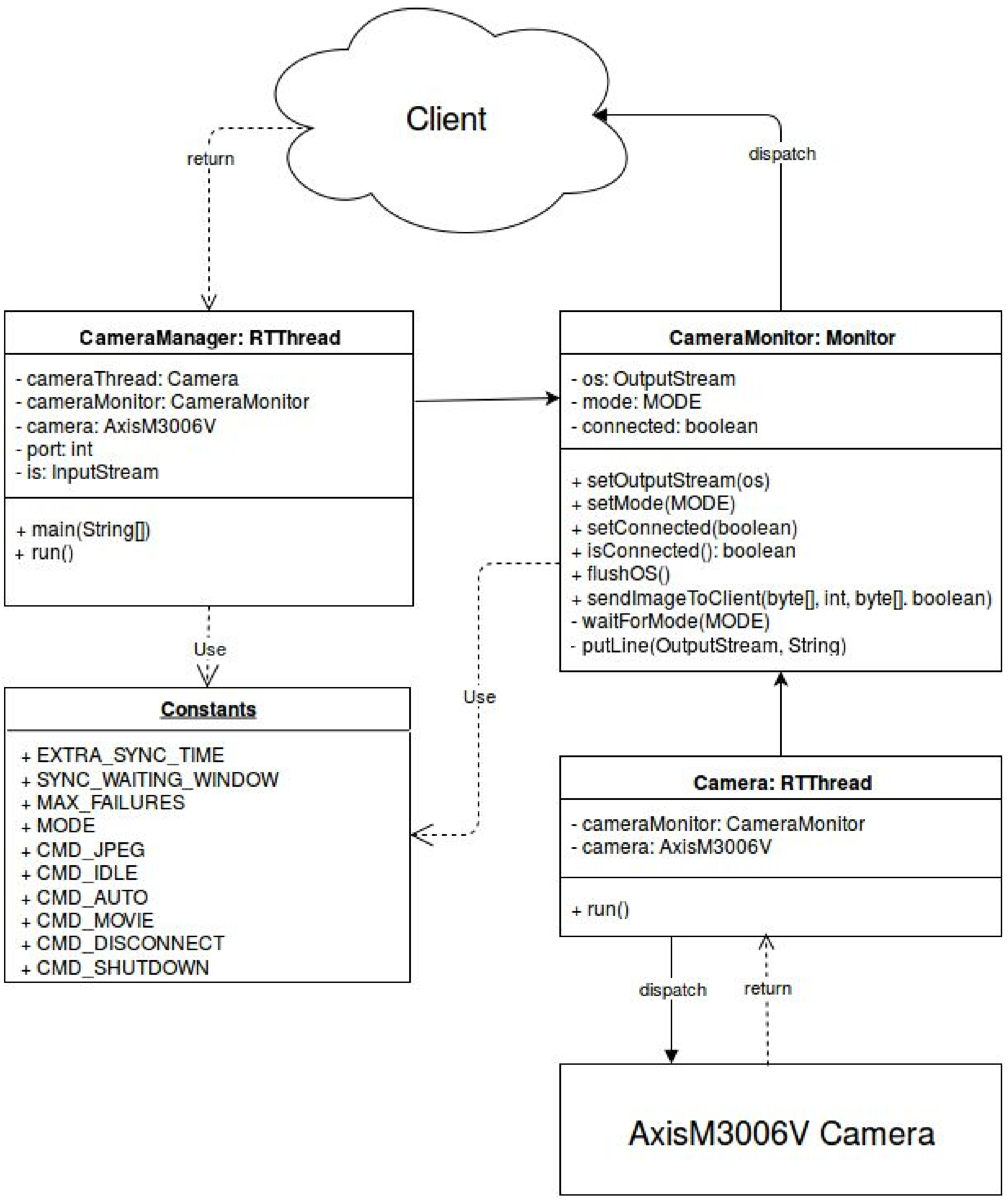
ImageOutput

This is the process responsible for taking images from the image queue and displaying them in the GUI through the ImagePanel. It can use the timeToDisplay from the Image to wait the appropriate time in CameraMonitor and synchronize the outputs. There is one ImageOutput per camera.

Image

This is the class used for storing meta information about the jpeg (timestamp, delay) and using it to calculate the time for it to display if it is to be synchronized

Server



CameraManager

This class includes the initial main method that is started. The main method starts this thread and it proceeds to create a proxy server that will communicate with the client. If this is a proxycamera it will also connect to the camera through a proxy and then pass the camera object into the newly created Camera thread. This is also the thread that will be waiting on input from the client and handling the possible commands.

For a mode change (CMD_IDLE, CMD_MOVIE, CMD_AUTO), it will simply call the monitor to update the state.

For CMD_DISCONNECT, it will close the connection with the client and restart looking for an acceptance to its server, setting the connection to false in the monitor as well as stopping the camera thread (restarting upon reconnection).

For CMD_SHUTDOWN, it will close the connection, shutdown the server, close the camera and then call camera.destroy to release any held resources, and finally closing itself (by finishing running).

CameraMonitor

The CameraMonitor is responsible for keeping the state of the camera system (mode, connected) as well as facilitating the output to the client through the OutputStream. It will also allow for waiting the correct period of time (for the mode) in between getting an image to create a periodic stream of images sent to the client.

Camera

The camera thread calls the camera to get the jpeg and timestamp which are then sent to the client via the CameraMonitor. The CameraMonitor facilitates the correct waiting.

Communication protocol

1. Header ending in empty line
2. command as a line (one of the commands listed)

if the command is CMD_JPEG

3. line containing the length of the image as an int N
4. N bytes of the jpeg
5. 8 bytes of the jpeg’s timestamp