Software Design Descriptions

for

OpenFlexure Microscope

Mehmet Alper Yilmaz - 2405611

Alparslan Yesilkaya - 2237923

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1 Purpose of the System

For decades, microscopy has become the cornerstone of scientific imaging and research in many fields, including medicine, biology, and physics. Standard commercial microscopes can have accurate motorized and high quality imaging features, but these devices are usually very costly. In the developing world, acquiring a commercial microscope is even less likely due to their inflated price, lack of customer support and lack of spare part availability. Open source hardware is aimed at empowering the delivery of scientific instruments, affecting science, local production, and education. OpenFlexure microscope is a full scale, motororized, laboratory level, 3D, open source microscope. The OFM seeks to make microscopy available to both low-budget programs in developed lavratories and emerging countries seeking to adapt more modern scientific research technology.

1.2 Scope

This report illuminates the project's architectural characteristics and patterns in accordance with IEEE standards. The study provides solutions to certain basic design limitations as well as adaptations. To the greatest extent possible, UML diagrams are used to improve the comprehensiveness of the document.

In this project, users will be able to use 3D printed microscopes thanks to Web of Things (WoT) over the internet, utilizing many client side environments with the support of open source software. Scientists will be able to extend and modify its source code due to its open source nature. More specific objectives include the following.

- Device will include a raspberry pi computer containing device control code for the operation of the camera, leds, motor movements, cabled connections, networking, image processing to enable client and server side applications functioning.

- The system will have GUI and APIs for scripting experiments which provide simultaneous connection of users to microscope, real-time camera feed, sample manipulation, data acquisition, data analysis.

- The system will enable a single client to access multiple microscopes. As a result; multiple image samples obliquely, increment in the throughput will be obtained.

- The system will be built as a server client architecture. As a result, client side applications can use any modern programming language.

- The system will provide a web application, a desktop application, a Python client, and a Matlab client.

- The system will be able to live-stream real time Motion JPEG (MJPEG) with auto focus features, data compression,image correction, and post- processing algorithms.

- The system will also provide automatic calibration features.

1.3 Stakeholders and their Concerns

End users are the only stakeholders of this project due to its open source nature. Thanks to its open source and extensible architecture, OFM adapts itself to novel technologies.

End User: Since the OFM is designed to be an open source project, anyone with or without the technical skills can use the device. Users can update, create, remove the project functionalities themselves if they have the skillset.

2. References

This document is prepared with respect to the specifications of IEEE 1016-2009 standard:
IEEE standard for information technology--systems design--software design descriptions. (2009). New York, NY: Institute of Electrical and Electronics Engineers.https://standards.ieee.org/standard/1016-2009.html

Other sources:
Collins, Joel & Knapper, Joe & Stirling, Julian & McDermott, Samuel & Bowman, Richard. (2021). Modern Microscopy with the Web of Things: The OpenFlexure Microscope Software Stack.

Collins, Joel & Knapper, Joe & Stirling, Julian & Mduda, Joram & Mkindi, Catherine & Mayagaya, Valeriana & Anyelwisye, Grace & Nyakyi, Paul & Sanga, Valerian & Carbery,

Dave & White, Leah & Dale, Sara & Lim, Zhen Jieh & Baumberg, Jeremy & Cicuta, Pietro & McDermott, Samuel & Vodenicharski, Boyko & Bowman, Richard. (2020). Robotic microscopy for everyone: the OpenFlexure Microscope. Biomedical Optics Express. 11. 10.1364/BOE.385729.

Stirling, Julian & Sanga, Valerian & Nyakyi, Paul & Mwakajinga, Grace & Collins, Joel & Bumke, Kaspar & Knapper, Joe & Meng, Qingxin & McDermott, Samuel & Bowman, Richard. (2020). The OpenFlexure Project. The technical challenges of Co-Developing a microscope in the UK and Tanzania. 1-4. 10.1109/GHTC46280.2020.9342860.

3. Glossary

Term	**Definition**
App	Abbreviation for Application
CRUD	Abbreviation for Create, Read, Update, Delete, operations used in database
HTTPS	Abbreviation for Hyper Transfer Protocol Secure
MYSQL	An open source database management system providing access to the data by multiple users
OFM	Abbreviation for Open Flexure Microscope
TCP	Abbreviation for Transmission Control Protocol
IOT	Abbreviation for Internet of things
WOT	Abbreviation for Web of Things
API	Abbreviation for Application Programming Interface

Table 3.1: Glossary

4. Architectural Views

4.1 Context View

The context of the project, including actors, are described comprehensively. In the context diagram, actors and their interactions with the OFM will be illustrated widely. Use case diagrams and their extensive definitions will be given below the context diagram.

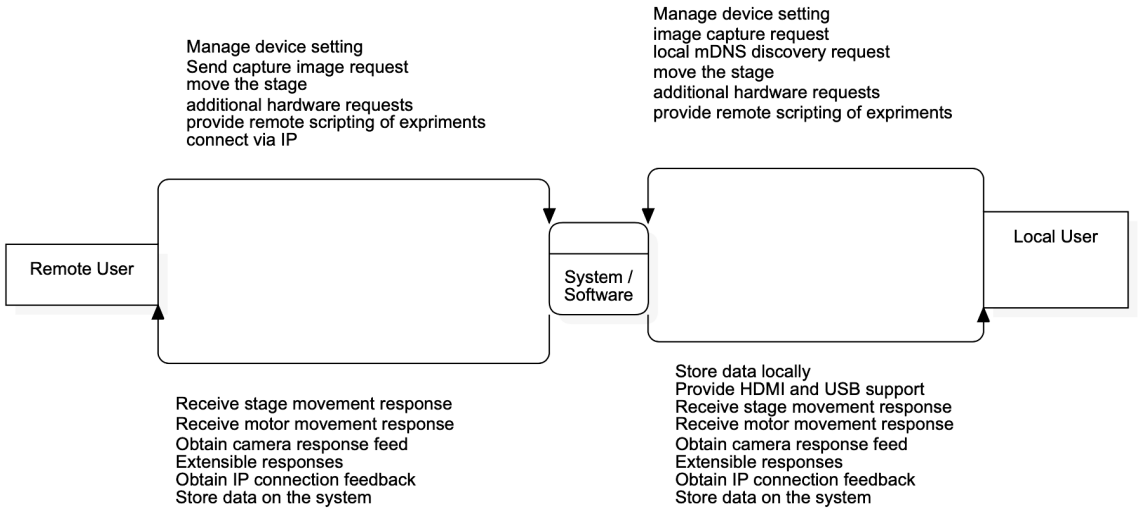

Figure 4.1: Context Diagram

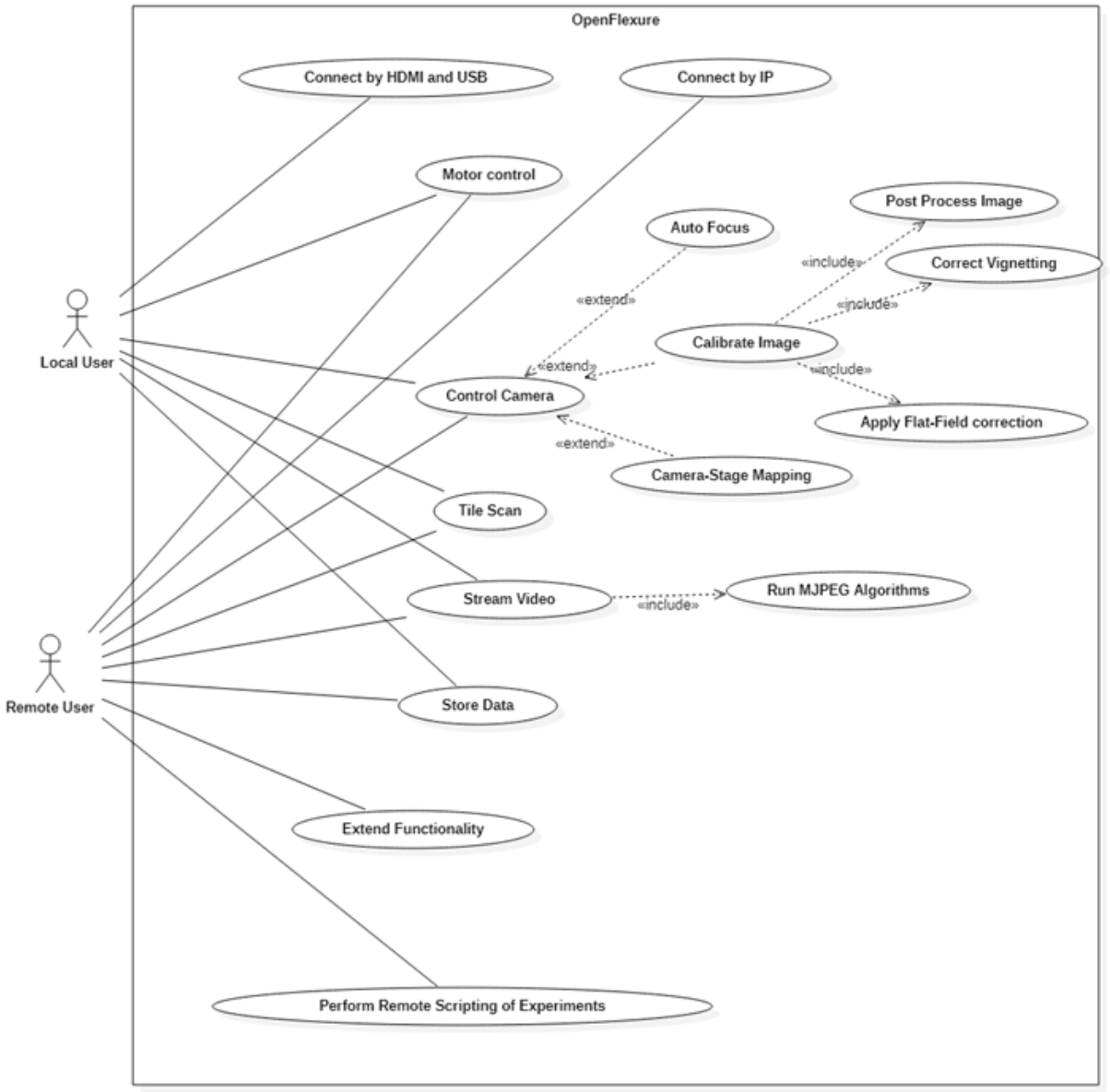

Figure 4.2: Use-case Diagram

Table 4.1: Use Case 1

Use Case ID	1

Use Case Name	Connect by HDMI and USB
Actors	Local User
Descriptions	The user connects to the system in order to use its functionalities.
Preconditions	The user has the access to the microscope physically
Postconditions	The user is successfully connected to the OpenFlexure Microscope system.
Normal Flow	1. Microscope is found using mDNS queries in the user's local network or the user directly connects to the microscope hardware 2. System accepts user connection
Exceptions	1a. If the user's LAN does not provide a mDNS support system will not answer and client side applications will provide error messages. 1a. In case of a hardware failure between the end systems connection fails.

Table 4.2: Use Case 2

Use Case ID	2
Use Case Name	Connect by IP

Actors	Remote User
Descriptions	The user can connect to the microscope control server application over the IP network.
Preconditions	Server application should be ready for connection.
Postconditions	The user is successfully connected to the server application and now the user can send HTTP request packet to the server and can receive HTTP response packet from the server.
Normal Flow	1. The client sends a small TCP segment to the IP address of the server application. 2. The server application acknowledges back to the client and it means TCP connection is established.
Exceptions	2a. If the connection is lost with the web application, the user can't control the microscope.

Table 4.3: Use Case 3

Use Case ID	3

Use Case Name	Control Motors
Actors	Local User and Remote User
Descriptions	The user controls motors on the OFM by interacting with one of the APIs or with a local client.
Preconditions	Connection is established with the OFM by either remotely or locally
Postconditions	Motors are moved as desired.
Normal Flow	1. The user sends movement data by using APIs or the local server. 2. Server side receives the data 3. Server side processes the data. According to this process, the server sends signals to the motors.
Exceptions	3a. If another user that is currently using the OFM already sent a signal to the motors, the signal which is sent later is dropped. User is informed accordingly.

Table 4.4: Use Case 4

Use Case ID	4
Use Case Name	Control the Camera
Actors	Local User, Remote User
Descriptions	The user controls camera specific actions
Preconditions	The user is connected to the OFM.

Postconditions	Tasks such as auto focus, calibration are performed.
Normal Flow	1. In the graphical interfaces user clicks the related button for updating camera settings. In the text based interfaces such as Python scripting API, client calls related functions to control the camera. 2. OFM processes the action internally. 3. If the action is valid, camera control actions are performed.
Exceptions	1a. Network errors result in error notifications on both server side and client side. 3a. Actions that require reentrant locks result in exceptions sent to the client side.

Table 4.5: Use Case 5

Use Case ID	5
Use Case Name	AutoFocus
Actors	Local User, Remote User
Descriptions	The Camera focuses on the sample and much better image and video quality is obtained.
Preconditions	The user is connected to the OFM. Use case number 4 is performed.
Postconditions	A focused and clearer camera feed level is achieved.

Normal Flow	1. Capture z stack images. 2. Z stack images are converted to gray scale. 3. By assigning higher values to higher spatial brightness variance, Laplacian convolution is applied. 4. Values are raised to fourth power and summed over image to provide sharpness value. 5. Returned to the z position with the highest sharpness.
Exceptions	1a. By capturing z stack images, lock for stage movement is required. If lock cannot be obtained causing an exception. 3a. If a malfunction of the led bulb that is used to light the sample surface occurs, brightness value cannot be computed correctly. As a result, incorrect autofocusing occurs.

Table 4.6: Use Case 6

Use Case ID	6
Use Case Name	Calibrate Image
Actors	Local User, Remote User
Descriptions	Certain corrections techniques applied to remove imperfections on the image.
Preconditions	Camera must be functioning properly. The user must satisfy the use case 4.
Postconditions	An image is obtained which is cleaned from

	negative effects of the Sony IMX 219 image sensor.
Normal Flow	1. Take an image 2. Apply flat field correction 3. Correct Vignetting 4. Perform post processing
Exceptions	1a. Taken image must be focused. If they are not focused, images must be retaken.

Table 4.7: Use Case 7

Use Case ID	7
Use Case Name	Post Process the Image
Actors	Local User, Remote User
Descriptions	The image that is acquired using the camera is post processed to correct its saturation.
Preconditions	The image must be stored internally or remotely.
Postconditions	Reduction in saturation at the edges of the image is eliminated.
Normal Flow	1. Acquire and store the image. 2. Enter the interface of control camera either graphically or using scripting 3. Run the post processing algorithm.
Exceptions	1a. failure to access the image causes an error.

Table 4.8: Use Case 8

Use Case ID	8
Use Case Name	Correct Vignetting
Actors	Local User, Remote User
Descriptions	Live streams and raw images that are captured by OFM suffer from vignetting. Vignetting is corrected by some techniques in software.
Preconditions	The image must be present or real time streaming must be started.
Postconditions	Vignetting is removed from the image or real-time preview.
Normal Flow	1. The image is taken or streaming is started 2. Access to the lens shading table in the camera's GPU-based image processing pipeline by the help of "picamera" library. 3. Correct vignetting by flat-field correction techniques.
Exceptions	2a. Shading table access requires reentrant locks. If the lock has not been acquired, the sequence results in an exception.

Table 4.9 : Use Case 9

Use Case ID	9

Use Case Name	Apply Flat-Field Correction
Actors	Local User and Remote User
Descriptions	Optical artifacts such as shading or lens cast are removed from the image.
Preconditions	The image must be present or real time streaming must be started.
Postconditions	Optical artifacts are successfully removed from the image.
Normal Flow	1. A flat-field array of the image, which consists of each pixel's dark current and their gain, is created. 2. Using a flat-field array, the correction equation is applied for each pixel. 3. At the end, resulting image will be calibrated image
Exceptions	1a. In case flat-field array isn't created properly, optical artifacts in the image may still exist.

Table 4.10: Use Case 10

Use Case ID	10
Use Case Name	Camera-stage Mapping
Actors	Local User, Remote User
Descriptions	The relationship between the axes and step size of the translation stage is calibrated and the stage is mapped to camera location accordingly.

Preconditions	The user is connected to the OFM. Use case number 4 is performed.
Postconditions	The translation stage is successfully mapped to the camera location.
Normal Flow	1. The translation stage is moved back and forth along its x and y axes in order. 2. The displacement in the image ,captured by camera, is analyzed and translation stage is calibrated. 3. The calibrations are combined into a 2x2 affine transformation matrix which is used in mapping stage to camera coordinates.
Exceptions	3a. In case the stage isn't appropriately mapped to camera location, the user can't move the camera in the desired direction.

Table 4.11: Use Case 11

Use Case ID	11
Use Case Name	Scan (Tile Scan)
Actors	Local User and Remote User
Descriptions	Tile scan provides images with larger fields of view constructed by combining normal images for users.
Preconditions	The user is connected to the OFM. Auto focus, Image calibration and camera calibration works very well.

Postconditions	An image with a large field of view is created.
Normal Flow	1. Camera moves around the sample. 2. Before taking each image, the microscope is brought back into focus. 3. Camera captures images at a series of locations. 4. The system combines the captured images and reconstructs a complete image by using overlapped regions.
Exceptions	1a. In case the camera doesn't move to the appropriate location, the camera can't take pictures in these areas. This might cause some problems in creating the final image. 2a. If the microscope is not brought back to the focus, the captured image might be of poor quality. 3a. In case the camera doesn't take images in some locations, this might cause some problems in creating the final image. 4a.If there are some locations in which camera doesn't take image : 1. The final image may not be created. 2. The final image might include some gaps. 4b. If there are some low-quality images: 1. The final image may not be created. 2. The final image might include poor quality regions.

Table 4.12: Use Case 12

Use Case ID	12
Use Case Name	Stream Video
Actors	Local User and Remote User
Descriptions	The OFM host provides a real-time MJPEG live stream of the camera for the users.
Preconditions	The user is connected to the OFM.
Postconditions	The users can display live video streams from the microscope interface.
Normal Flow	1. A background thread recording JPEG (MJPEG) frames from the camera into a buffer is run on startup. 2. In case users connect to the system, the server starts to send a multi-part stream of frames, stored in the buffer, to users.
Exceptions	1a. In case the thread doesn't run or it is terminated, then JPEG frames can't be recorded. 2a. In case user disconnected from the server, live stream stops

Table 4.13: Use Case 13

Use Case ID	12
Use Case Name	Run MJPEG Algorithms
Actors	Local User and Remote User
Descriptions	MJPEG is a high quality video compression format in which each frame is compressed as a JPEG image.
Preconditions	The OFM streams video.
Postconditions	OFM successfully streams video as MJPEG format.
Normal Flow	1. Each frame in the video is compressed separately as a JPEG image format. 2. Resulting JPEG images are combined and the resulting video will be in MJPEG format.
Exceptions	1a. If video frames couldn't be compressed in JPEG format, MJPEG video won't be created.

Table 4.14: Use Case 14

Use Case ID	13
Use Case Name	Store Data
Actors	Local User and Remote User
Descriptions	The OFM system stores images and microscope state data for analysis and the user can access this data through HTTP

	API or data cable.
Preconditions	The user is connected to the OFM.
Postconditions	The User successfully accesses the every available data he/she wants.
Normal Flow	1. In case a picture is captured by the camera, the OFM system stores metadata about the state of the microscope during capturing picture including stage position,camera settings, calibration data, and custom metadata added by the user. The metadata is stored as JSON (JavaScript Object Notation) formatted string in the “UserComment” EXIF field. Furthermore, JPEG images and raw 8MP Bayar data collected from the camera are also stored for advanced analysis. Raspberry Pi stores this data on SD card or external USB storage device. 2. Through a data cable or HTTP API, the stored data can be accessed by users.
Exceptions	1a. In case of data corruption or hardware failure, users can’t access the data.

Table 4.15: Use Case 15

Use Case ID	14
Use Case Name	Extend Functionality

Actors	Remote User
Descriptions	Microscope Functionalities other than the most basic ones are provided with extensions.
Preconditions	The user is connected to the OFM.
Postconditions	OFM has a new functionality
Normal Flow	1. A Python script providing different functionality for the OFM is written. 2. OFM runs the script and now it has additional functionality.
Exceptions	2a. In case the script includes a command which OSM can't perform, desired functionality can't be added.

Table 4.16: Use Case 16

Use Case ID	15
Use Case Name	Perform Remote Scripting of Experiments
Actors	Remote User
Descriptions	Python/MATLAB clients can transform web API into native Python/MATLAB functions. In that way, experiments can be performed remotely via scripts.
Preconditions	The user is connected to the OFM.
Postconditions	The User successfully makes OSM perform particular experiments via scripts .
Normal Flow	1. The user writes a python/MATLAB

	script which will perform a particular experiment. 2. The user sends the script into the server application. 3. OFM receives the script, processes it, and performs the desired experiment.
Exceptions	2a. If the client lost connection between the server application while sending the packet, experiments can't be performed 3a.In case the script includes a command which OSM can't perform, the experiment fails.

4.2 Composition View

The component and interface viewpoints are represented using UML Component Diagram. Figure below shows a diagram that enables to see the overall picture of the system by minimizing ambiguity and providing a more abstract perspective of the system

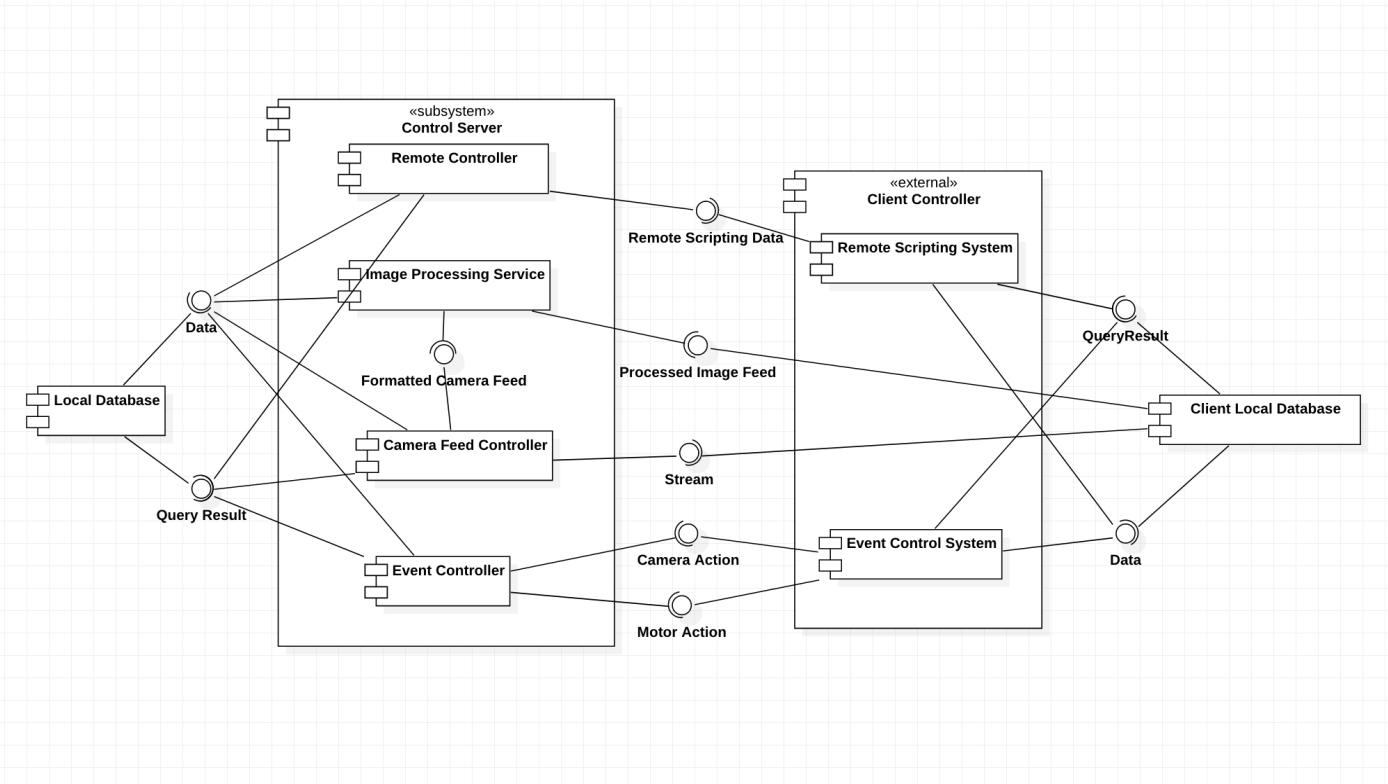

Figure 4.3: Component Diagram

Design Rationale:

- Control Server controls components having external interfaces. For security reasons and as a good design practice the data passed through the Control Server.
- Control Server stores beside image and video data as well as functionality specific data. For example, by storing image data, it links this data with the user.
- Control server keeps temporary frame data. When the communication fails frames are dropped. As a result, it fails to store to the database.
- Initial data manipulation is performed in the Control Server. Data is stored to the database later on for performance reasons.
- Client components connect to the Control Server to perform their functionality.
- Client components may or may not be on a separate device.
- One client application can connect to multiple Control Servers and one Control Server accepts multiple client connections.

- Database is not a separate server but a local disk or drive located on either client side or server side or both.

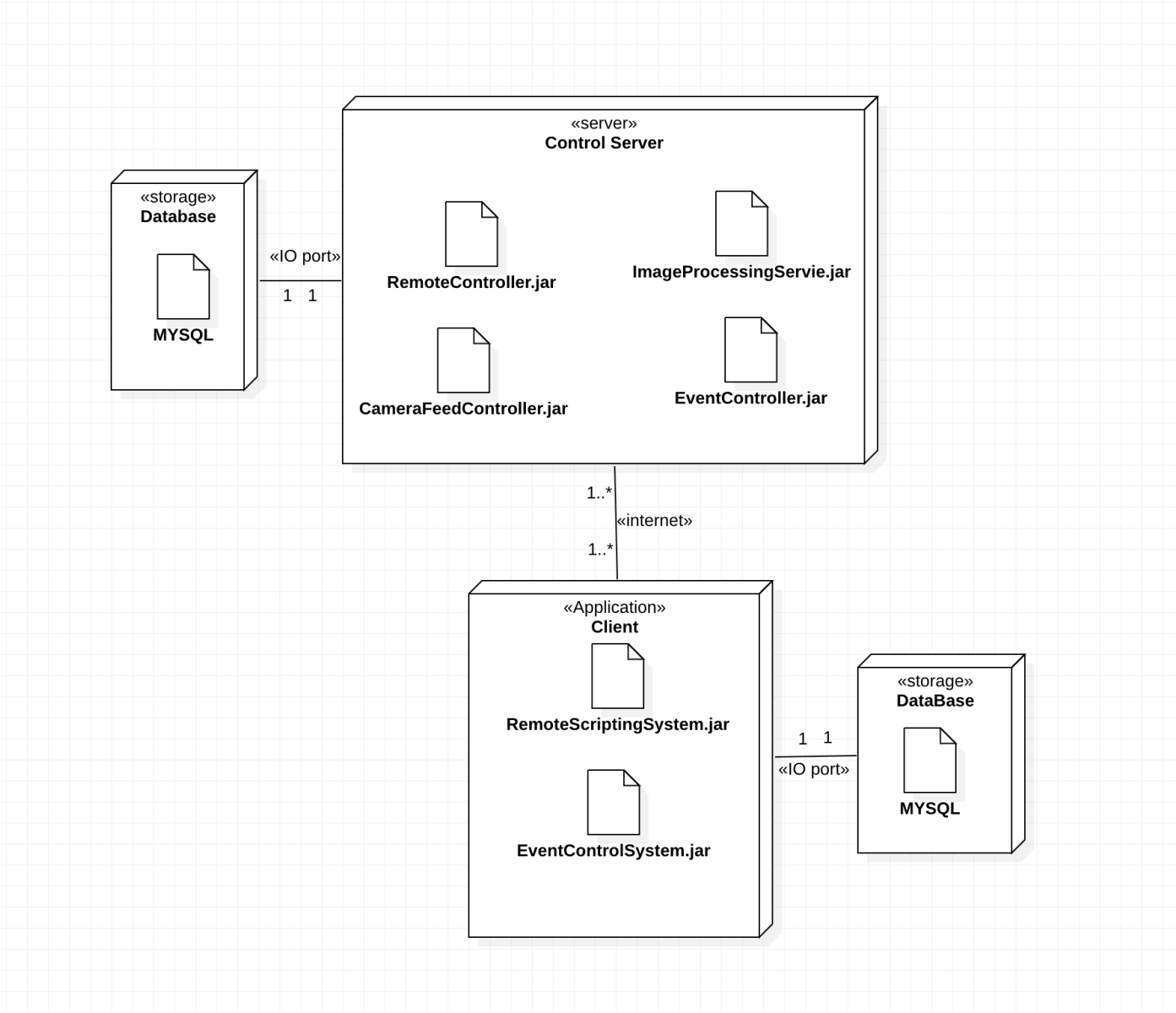

Figure 4.4: Deployment Diagram

Design Rationale:

- Control server is developed and running Java. Database is managed with MySQL

- Client applications will communicate with the Control server either locally or remotely to access or manipulate the data that is acquired by the microscope.
- There are two types of separate database storages. These are local client databases, microscope local databases.
- Only the remote systems use the internet connection due to the disadvantages of latency and connection problems.
- All the connections that are over the internet are encrypted and use SSL and HTTPS protocols.

4.3 Information View

This viewpoint expresses the elements which are transmitted between various components of the system. These structures will be examined in terms of their manipulation, storage in local collections, and transmission during different stages.

4.3.1 Interfaces

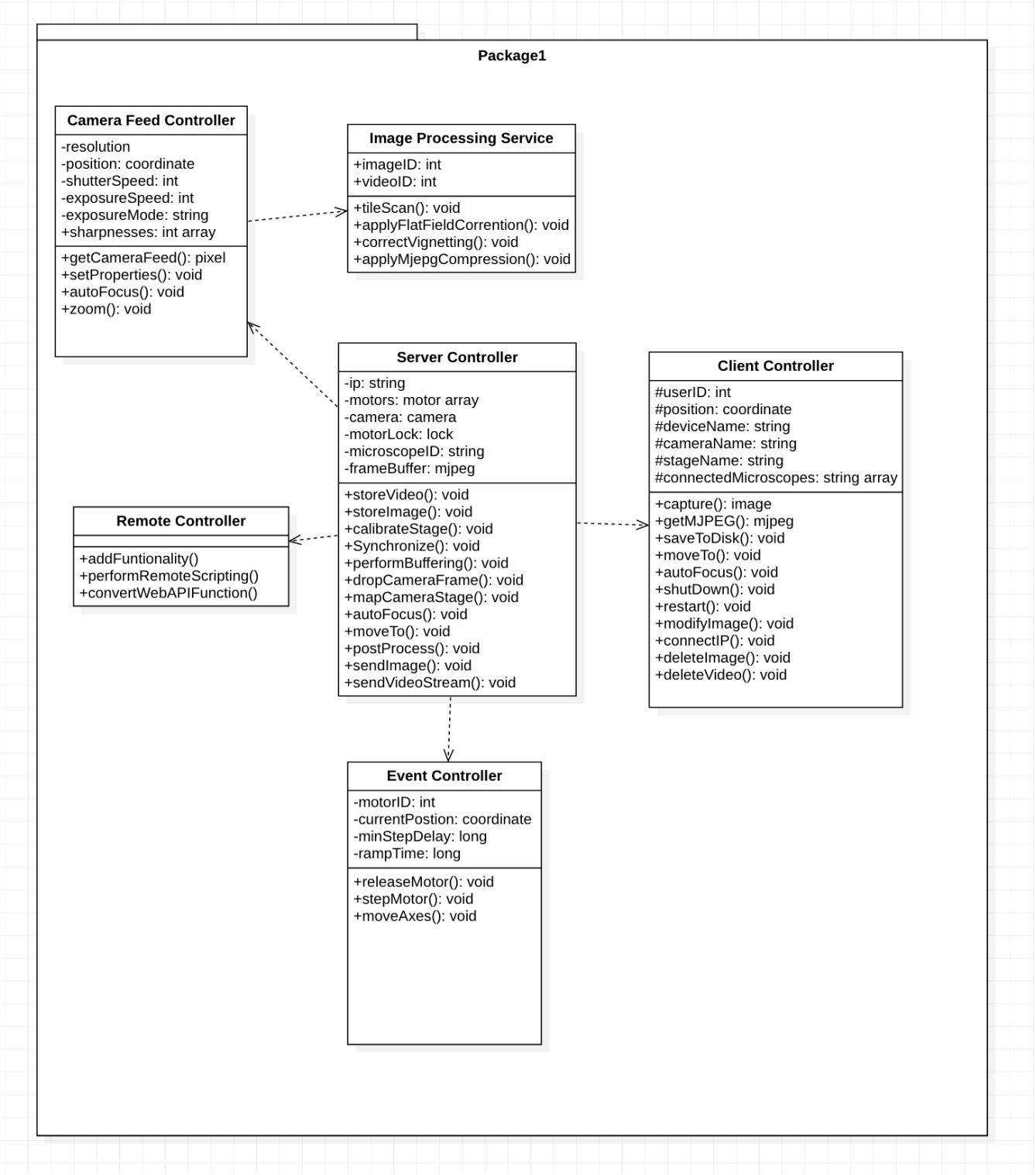

Figure 4.5: Interface Class Diagram

getCameraFeed	Camera feed is obtained by the Raspberry PI.
setProperties	Given properties set updating attributes .
autoFocus	Image is focused by applying an auto focusing algorithm.
zoom	Camera zooms.
releaseMotor	Motor is moved to its initial position.
stepMotor	Motor is moved away from its original position.
moveAxes	Axes moved using stepMotor and releaseMotor.
storeVideo	Video is stored in the local database.
storeImage	Image is stored in the local database.
calibrateStage	Image is stored using a calibration algorithm.
tileScan	Multiple tiles are scanned and combined.
synchronize	Multiple users prevented from executing critical sections at the same time.
performBuffering	Video or image feed is stored temporarily for later processing.
dropCameraFrame	Certain camera frames are deleted.
mapCameraStage	Camera stage is mapped to its logical plane.
autoFocus	Auto focus signal sent to the camera.
moveTo	Stage is moved to a certain point.
applyFlatFieldCorrection	Flat field correction algorithm is applied.
postProcess	Camera feed is further processed to remove imperfections.
correctVignetting	Vignetting correction algorithm is performed.
sendImage	Image is sent to the client.

sendVideoStream	Video stream is sent to the client.
applyMjpegCompression	Mjpeg data is compressed for transmission.
capture	Image capture signal is sent to the server.
getMJPEG	A signal for receiving Mjpeg data is sent.
saveToDisk	Acquired data is saved to the local database.
moveTo	A point is specified and sent to the OFM.
autoFocus	A signal for initiating auto focus is sent to the OFM .
shutDown	Client closes the connection.
restart	Client restarts itself.
modifImage	Image modifications performed.
connectIP	Connected to the OFM using IP.
deleteImage	Acquired image is deleted.
deleteVideo	Acquired video is deleted.
performGUI	Web client provides GUI.
addFunctionality	Python client combines building blocks of functions to extend functionality.
performRemoteScripting	Python client enables users to experiment using scripting features of Python.
convertWebApiFunction	Python client maps Web API to scripting functions.
performRemoteScripting	MATLAB client enables users to experiment using scripting features of Python.
convertWebApiFunction	MATLAB client maps Web API to scripting functions.
addFunctionality	MATLAB client combines building blocks of functions to extend functionality.
connectHDMI	Desktop client connects to the OFM by HDMI and USB.
findMicroscopes	Desktop client performs microscope search in its LAN by the help of mDNS queries.
performGUI	Desktop client provides GUI.

addCommonlyUsedDevices	A device is added to the list of commonly used devices.
removeCommonlyUsedDevices	A device is removed from the list of commonly used devices.

Table 4.17: Operation Description

Operation	**Inputs**	**Outputs**	**Exceptions**
getCameraFeed	- cameraFeed	MJPEG if operation is successful, 0 otherwise	- The camera is already providing the feed - Illumination is not sufficient
setProperties	- resolution - position - shutterSpeed - exposureSpeed - sharpnesses	- Trues if operation is successful, False otherwise	- An invalid input value is given
autoFocus		- Camera feed is focused accordingly, 0 otherwise	- If the camera is already focused - If the camera stage is moving - If another user has the camera lock
zoom	- zoomLevel	- Camera is zoomed if successful, 0 otherwise	- Camera lock belongs to another user - Invalid zoom level
releaseMotor	- releaseAmount	- Motors are moved accordingly if successful, 0 otherwise	- Invalid release amount - Motor lock belongs to another user.
stepMotor	- stepAmount	- Motors are moved accordingly if successful, 0 otherwise	- Invalid step amount - Motor lock belongs to another user
moveAxes	- position	- Motors are moved using stepMotor and releaseMotor if successful, 0 otherwise	- Invalid position - Motor lock belongs to another user
storeVideo	- videoFile	- Video is stored in local database of the user if successful, 0	- No available space for storage - Invalid video file name. - Invalid video file location

		otherwise	
storeImage	- imageFile	- Image is stored in local database of the user if successful, 0 otherwise	- No available space for storage - Invalid image file name - Invalid image file location
calibrateStage		- Camera stage calibrated if successful, 0 otherwise	- Motor lock belongs to another user
tileScan	- tileDimensions	- perform scanning if successful, 0 otherwise	- Motor and camera locks belong to other users - Invalid dimension
synchronize	- cameraLock - motorLock	- Critical section is synchronized if successful, otherwise waiting	- Thread exception
performBuffering		- Data is stored in the buffer temporarily if successful, otherwise 0	- Buffer is not empty
dropCameraFrame	- frameNumber	- Specified frame is removed from the buffer, otherwise 0	- Invalid frame number
mapCameraStage	- stageCoordinates	- Camera coordinates and stage coordinates are mapped if successful, otherwise 0	- Motors are busy
autoFocus		- Auto focus signal is sent to the camera if successful, otherwise wait	- Another thread possess the camera lock
moveTo	- coordinate	- moveAxes signal is sent to the camera if successful, otherwise 0	- moveAxes signal is not successful - Invalid coordinate
applyFlatFieldCorrection	- cameraFeed	- Image corrected if successful, otherwise 0	- Corrupted camera feed is given
postProcess	- cameraFeed	- Saturation decrease on the edges is removed if	- Corrupted camera feed is given - Low illumination

		successful, 0 otherwise	
correctVignetting	- cameraFeed	- Vignetting is reduced if successful, 0 otherwise	- Corrupted camera feed is given - Low illumination
sendImage	- image	- image is sent to the requesting client if successful, otherwise try resending	- Network connection problems occurred - Client is closed the connection
sendVideoStream	- video	- If successful, video stream is sent to the requesting client, 0 otherwise	- Network connection problems occured - Client is closed the connection
applyMjpegCompression	- dataToBeTransmitted	- If successful, data is compressed. Otherwise 0 is returned	- Corrupted data type
capture		- Capture signal is sent to the server if successful. Otherwise 0 is returned	- Network connection problems occured
getMJPEG		- A signal requesting MJPEG data is sent to the server if successful. Otherwise 0	- Network connection problems occured
moveTo	- coordinate	- If successful, a signal requesting the camera to move to the specified coordinate is sent. Otherwise 0	- Network connection problems occured. - Invalid coordinate
autoFocus		- If successful, a signal requesting the camera to be auto focused sent to the server. 0 otherwise	- Network connection problems occured. - Unsuccessful autoFocus calls happened in the server
shutDown	- connectionCredentials	- Client shuts down the connection if successful. 0 otherwise	- Invalid connection credentials

restart	- connectionCredentials	- Client restarts if successful. 0 otherwise.	- Invalid connection credentials
modifyImage	- image - modifications	- image is modified according to modifications if successful. Otherwise 0	- Invalid image type - Image does not exist - Wrong image path
connectIP	- ipAddress	- Client is connected to the IP address if successful. Otherwise 0	- Invalid IP adress
deleteImage	- imageName	- Image is deleted from the local database if successful. Otherwise 0.	- Invalid image name - Image does not exist at given path
deleteVideo	- videoName	- Video is deleted from the local database if successful. Otherwise 0	- Invalid video file name - Video file does not exist at given path
performGUI		- For web client; GUI is set up for interaction if successful. Otherwise 0.	- Server does not response - Network connection problems
addFunctionality	- functionsToBeExtended	- For the Python clientI; if successful, given functions are combined to produce new functionalities. Otherwise 0	- Invalid functionality
performRemoteScripting		- The Python client performs experiments using scripting If successful. Otherwise 0	- Network connection problems
convertWebApiFunction		- Web API methods are mapped to the Python scripting functions If successful, otherwise 0	- Invalid functionality

performRemoteScripting		- The MatLab client performs experiments using scripting If successful. Otherwise 0.	- Network connection problems
convertWebApiFunction		- Web API methods are mapped to the MatLab scripting functions If successful. Otherwise 0	- Invalid functionality
addFunctionality	- functionsToBeExtended	- For MatLab client; if successful, given functions are combined to produce new functionalities. Otherwise 0	- Invalid functionality
connectHDMI		- If successful, the desktop client is connected to the OFM using HDMI and USB. Otherwise 0.	- USB connection problems - HDMI connection problems
performGUI		- For desktop client; GUI is set up for interaction if successful. Otherwise 0.	- Server does not response - Network connection problems
addCommonlyUsedDevices	- deviceNumber	- If successful, device number is added to the list of commonly used OFMs. Otherwise 0	- Invalid device number - Device is already in the commonly used OFMs list.
removeCommonlyUsedDevices	- deviceNumber	- If successful, device number is added to the list of commonly used OFMs. Otherwise 0	- Invalid device number - Device does not exist in the list of commonly used OFMs.

Table 4.18: Operation Design

Design Rationale:

- Server supports multiple client connections and serves data concurrently.
- Motor and Camera classes are used inside of the Microscope class in a composited manner.
- Camera and stage are mapped with each other. This is done moving the stage in x and y axes then displacement in the image is observed. After that, the acquired calibration setting is combined using a 2x2 affine transformation matrix. This matrix maps the stage to the camera coordinates.
- Core architecture is written as stand alone libraries instead of specific libraries to the OFM.

4.3.2 Database Operations

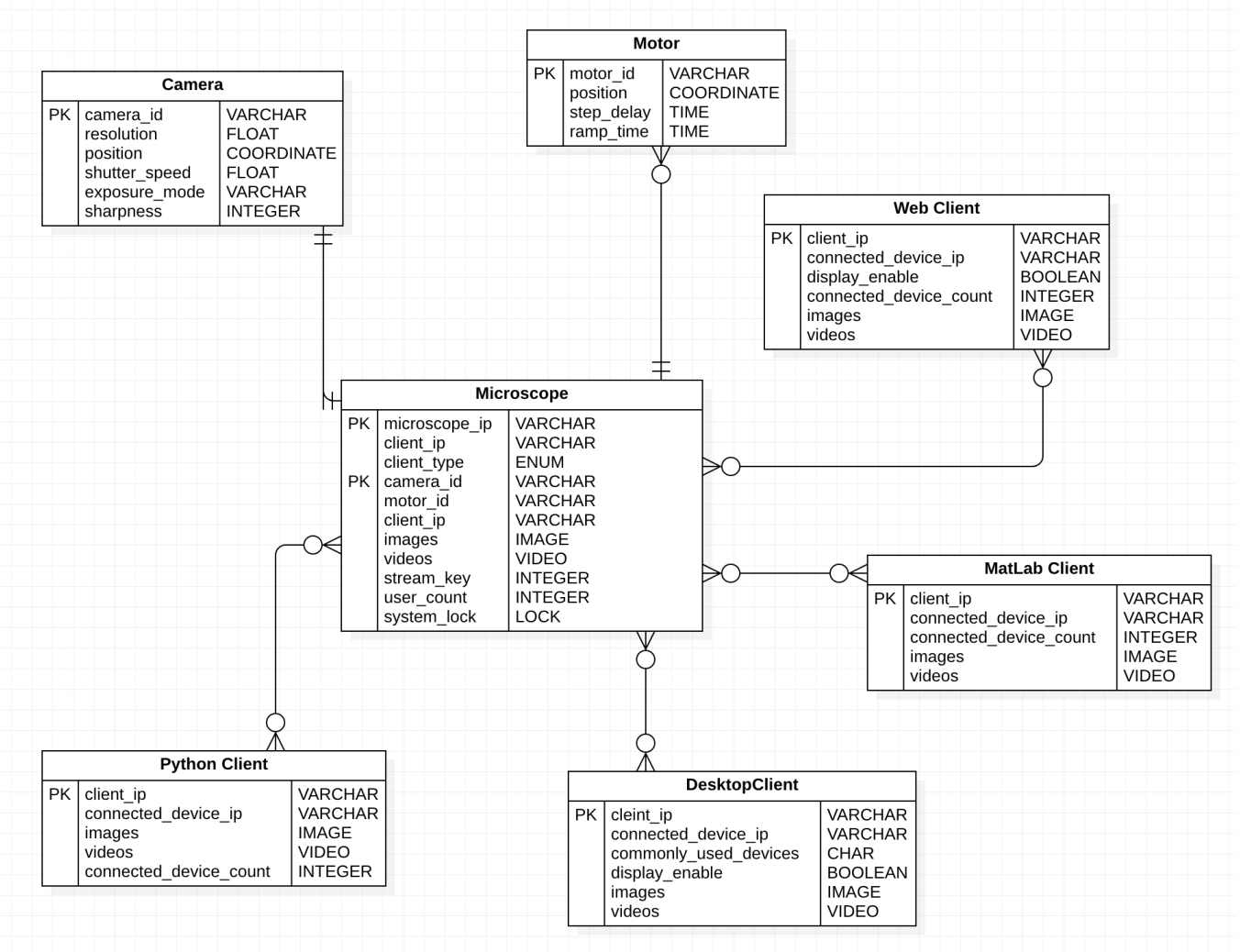

Figure 4.6: Database Class Diagram

Operation	CRUD Operation
getCameraFeed	Create: Read: Camera Update: Microscope, Clients Delete:
setProperties	Create: Read: Update: Camera Delete:
autoFocus	Create: Read: Camera Update: Camera Delete:
stepMotor	Create: Read: Motor Update: Motor Delete:
moveAxes	Create: Read: Microscope Update: Camera, Microscope Delete:
storeVideo	Create: Read: Camera Update: Microscope Delete:
storeImage	Create: Read: Camera Update: Microscope Delete:
calibrateStage	Create: Read: Motor Update: Motor Delete:
tileScan	Create: Read: Camera Update: Motor, Camera Delete:
synchronize	Create: Read: Microscope Update: Microscope Delete:
performBuffering	Create: Microscope Read:

	Update: Microscope Delete:
dropCameraFrame	Create: Read: Update: Microscope Delete: Microscope
mapCameraStage	Create: Read: Camera Update: Camera, Motor Delete:
autoFocus	Create: Read: Camera Update: Microscope Delete:
moveTo	Create: Read: Camera Update: Motor, Microscope Delete:
applyFlatFieldCorrection	Create: Read: Update: Microscope Delete:
postProcess	Create: Read: Update: Microscope Delete:
correctVignetting	Create: Read: Update: Microscope Delete:
sendImage	Create: Clients Read: Update: Microscope, Clients Delete:
sendVideoStream	Create: Clients Read: Update: Clients, Microscope Delete:
applyMjpegCompression	Create: Microscope Read: Update: Delete: Microscope
capture	Create: Clients Read: Camera, Microscope

	Update: Delete:
getMJPEG	Create: Clients Read: Camera. Microscope Update: Clients Delete:
saveToDisk	Create: Read: Microscope, Camera Update: Clients Delete:
moveTo	Create: Read: Camera Update: Camera, Microscope Delete:
autoFocus	Create: Read: Camera Update: Camera Delete:
shutDown	Create:- Read:- Update:- Delete:-
restart	Create:- Read:- Update:- Delete:-
modifImage	Create: Read: Update: Client Delete:
connectIP	Create: Read: Microscope Update: Delete:
deleteImage	Create: Read: Update: Delete: Clients, Microscope
deleteVideo	Create: Read: Update: Delete: Clients, Microscope
performGUI	Create:

	Read: Microscope Update: Client Delete:
addFunctionality	Create: Read: Update: Microscope Delete:
performRemoteScripting	Create: Read: Microscope Update: Microscope Delete:
convertWebApiFunction	Create: Read: Update: Microscope Delete:
convertWebApiFunction	Create: Read: Update: Microscope Delete:
addCommonlyUsedDevices	Create: Read: Microscope Update: Clients Delete:
removeCommonlyUsedDevices	Create: Read: Microscope Update: Delete: Clients

Table 4.19: CRUD Operations

Design Rationale:

- MYSQL database used for the system.
- A microscope table has a list of clients tables added. In case of new connection or losing connection this table is updated.
- A microscope table has multiple motors and only one camera.
- All the clients are in terms of the logical database model the same. However, their functionality differs in the implementation.

4.4 Interface View

In this view, the internal interfaces and the external client interfaces will be specified in detail with design rationale.

4.4.1 Internal Interfaces

Interface between Database and Remote Scripting Handler:
Database will be updated by the operations that are carried out in the Remote Scripting Handler. Remote Scripting Handler implements certain functionalities to communicate with the local database. When the Python and Matlab clients connect to the Control Server, they operate using this interface.

Design Rationale:

- Web client and Desktop client cannot access this interface.
- Multiple clients can access this interface concurrently provided that their operations do not require a system lock.
- If an operation requires the system lock, access to the database is performed in order to prevent race conditions and also to keep the system consistent and secure.

Interfaces between Server Controller and Database:
Server Controller implements the required functionality to interact with the database. Data is passed to the database from the ServerController for security reasons and architectural concerns. Query results that are retrieved from the database passed to the Server Controller. After that, the server controller redirects the data to corresponding Controllers. Server controller stages the data before passing it to the database. This way, fast data access and manipulation is satisfied.

Design Rationale:

- In case of system failure, query results are stored in the database
- If an operation requires the system lock, access to the database is performed in order to prevent race conditions and also to keep the system consistent and secure.
- Server Control runs on multiple threads to provide service to multiple subsystems as well as multiple client instances.
- In order to prevent starvation of the threads Server Controller uses a priority scheduling algorithm with aging enabled.

Interface between Camera Feed Controller and Server Controller:

Camera feed is acquired by Camera Feed Controller. If camera feed is needed to be stored to the database, it is placed there by the help of the Server Controller. Camera Controller implements the functionalities that are desired by Server Controller.

Design Rationale:

- Acquired image or video is staged in the Server Controller. In case of a save request data is loaded to the database.

- Camera Feed Controller serves ready and processed data to the Server Controller

Interfaces between ServerController and Event Controller:

Event controller waits for an event to happen. Camera movement, motor movement, data retrieval from the local database, positioning, calibration, tile scans can be given as examples of events. Whenever an event occurs it signals the Server Controller to take the necessary actions. If the query that is given by the Server Controller to the database is successful, the Event Controller is informed accordingly.

Design Rationale:

- Event Controller has priority in the database since it may require time critical actions. Therefore, in case of a system lock requiring action, other threads are preempted.

- In case of other threads requiring time critical actions at the same time, First Come First Serve algorithm is used, since these operations do not have priority over each other.

Interfaces between Camera Feed Controller and Image Processing Service:
Raw camera feed can be processed to serve clients and store in the databases. Image Processing Service unit performs a set of operations on camera feed to remove imperfections. Flat field correction, removal of vignetting, further post processing can be given as examples of these operations. Camera Feed Controller implements the functionalities that are provided by Image Processing Service.

Design Rationale:

- In case of network delay, Camera Feed Controller drops frames in order to catch up with the real time image feed requirement.

- Camera feed must be queued for the processing of the Image Processing Service since multiple clients can use this service concurrently.

- In order to satisfy performance requirements, Image Processing Service runs multiple threads on a single process of Image Processing Service. For every client a new Image PRocessing Service process is created.

4.4.2 External Interfaces

4.4.2.1 User Interfaces

The OpenFlexure Microscope system has interfaces for Web Application, Desktop Application, Python Client and MATLAB client. Python and MATLAB clients are very different from Web and the Desktop application due to their primary functional concerns. Detailed explanations for each interface will be explained in further sections.

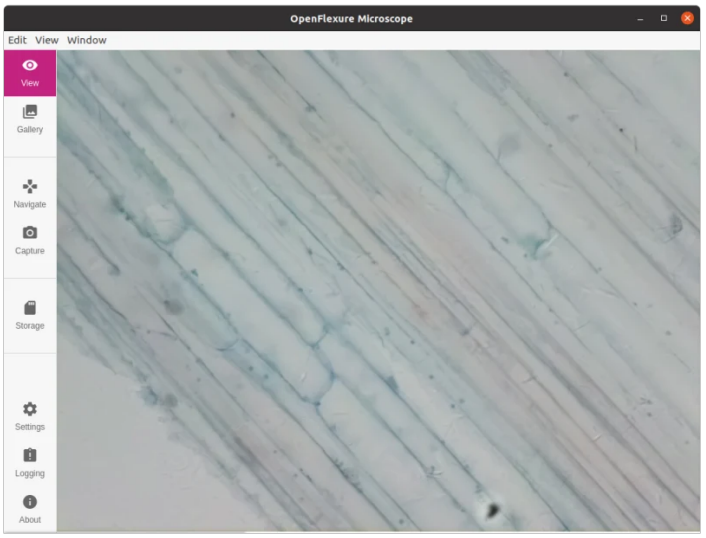

Figure 4.7 OFM Desktop Application Interface

Web Application Interface

This interface provides GUI to the users. Thanks to its being a web application it can be accessed by different client systems. It also includes live stream of camera, image capture, data management, and extension support functionalities.

Users can access the OpenFlexure graphical interface from any web browser which is either connected to the same local network or the internet. If the microscoscope is in the local network, the user can access it by typing “http://microscope.local:5000” to the web browser. If the OFM is in the external network, the user can access it by typing “http:/microscopesIPadress”, for example “http://144.122.171.83:5000”. The interface of the web application runs on port 5000.

Design Rationale:

- The interface should be compatible with multiple web browsers.
- If there are multiple microscopes in the local network, users should make sure that each server has different port numbers.
- Video files are saved as ".mp4" format, and images can be saved as ".jpeg" or ".png" format.

Desktop Application Interface

OpenFlexure Connect desktop application provides this interface with GUI support. It provides mDNS discovery and manual connection features. It also saves commonly accessed microscopes in a list. This interface is responsive with general purpose interface frameworks.

Design Rationale:

- Desktop Application should be compatible with different operating systems like Linux, Macos, Windows.
- Users are restricted with functionality of the interface. Users can only perform the allowed actions.

Python Client Interface

This interface allows users to interact with the remote scripting system which enables users with experiment scripting. It achieves its goal by mapping web API functions to python functions. As a result, interactive sessions are possible through scripting. Thanks to this interface, the microscope can be controlled from a Python script that can be run on the Raspberry Pi itself or remotely. Jupyter notebook can be used to run the script remotely allowing users to plot graphs and display images.

Design Rationale:

- Users should make sure that the written python script should have the same python version that the server code has.
- In case of run-time errors, the remote controller notifies the remote scripting system of the error and the remote scripting system informs the user with this interface.

MATLAB Client Interface

Users can access and control the microscope through MATLAB over a network or locally. Users are able to control the motors and access to the camera feed as well as extend its functionalities.

Design Rationale:

- Users should make sure that the used MATLAB version should be compatible with the server side.
- In case of run-time errors, the remote controller notifies the remote scripting system of the error and the remote scripting system informs the user with this interface.

4.4.2.2 System Interfaces

Interface Between Remote Scripting System and Remote Controller (Remote Scripting Experiments)

Remote Scripting system is where the user sends its python/MATLAB scripts in the first place. After checking compile errors and putting required client local data in its place, the script is sent to the remote controller. Remote controller executes the delivered script and makes the OFM perform desired actions by directly communicating with Raspberry Pi.

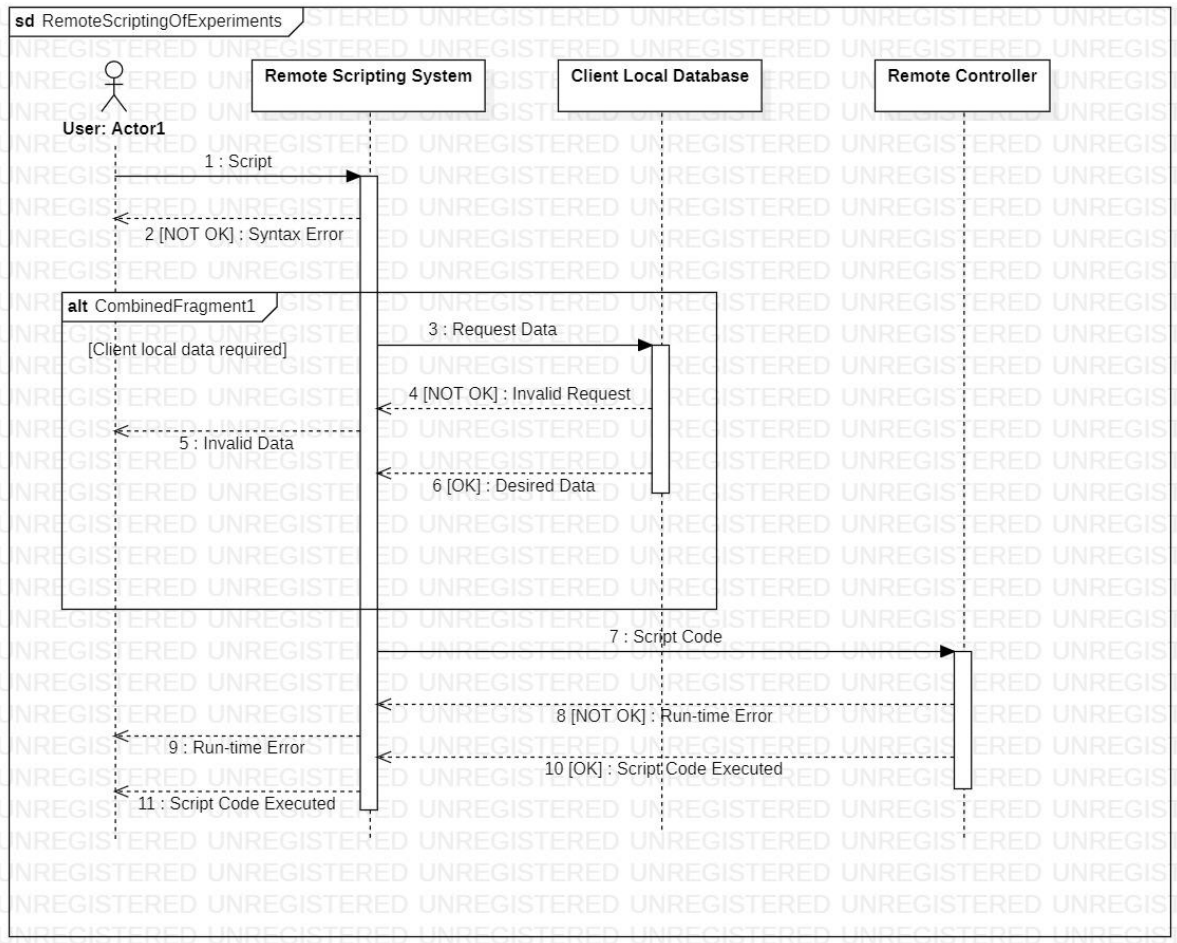

Figure 4.11: Sequence diagram showing the interface between Remote Scripting System and Remote Controller

Design Rationale:

- In case required local data doesn't exist, the remote scripting system terminates and informs the user.
- In case run-time error occurred in the remote controller, the remote controller terminates the program and informs the remote scripting system.

Interface Between Client Local Database and Image Processing Service (Processed Image Feed)

Image Processing Service performs operations on raw camera feed to remove imperfections. These operations can be Flat field correction, removal of vignetting or further post processing. In case the user wants to access processed camera feed, Client controller sends a request packet, indicating image processing methods and its parameters, to the server and then Image Processing Service sends the requested processed camera feed to

Client. After response arrives, delivered camera feeds are stored in the Client Local Database.

Design Rationale:

- Processing the images can take some time and multiple clients can request processed images. Therefore, requests are queued on the server side.
- In order to satisfy performance requirements, Image Processing Service runs multiple threads on a single process of Image Processing Service. For every client a new Image PRocessing Service process is created.

Interface Between Client Local Database and Camera Feed Controller (Stream Video)
The OFM server uses Camera Feed Controller to record video streams and users can watch the stream by accessing the client local database. These two systems communicate with each other by means of this interface. Firstly, the client local database accesses the camera and then the Camera Feed Controller continuously sends the video stream into the client local database until the client application stops it or hardware error occurs on the server side.

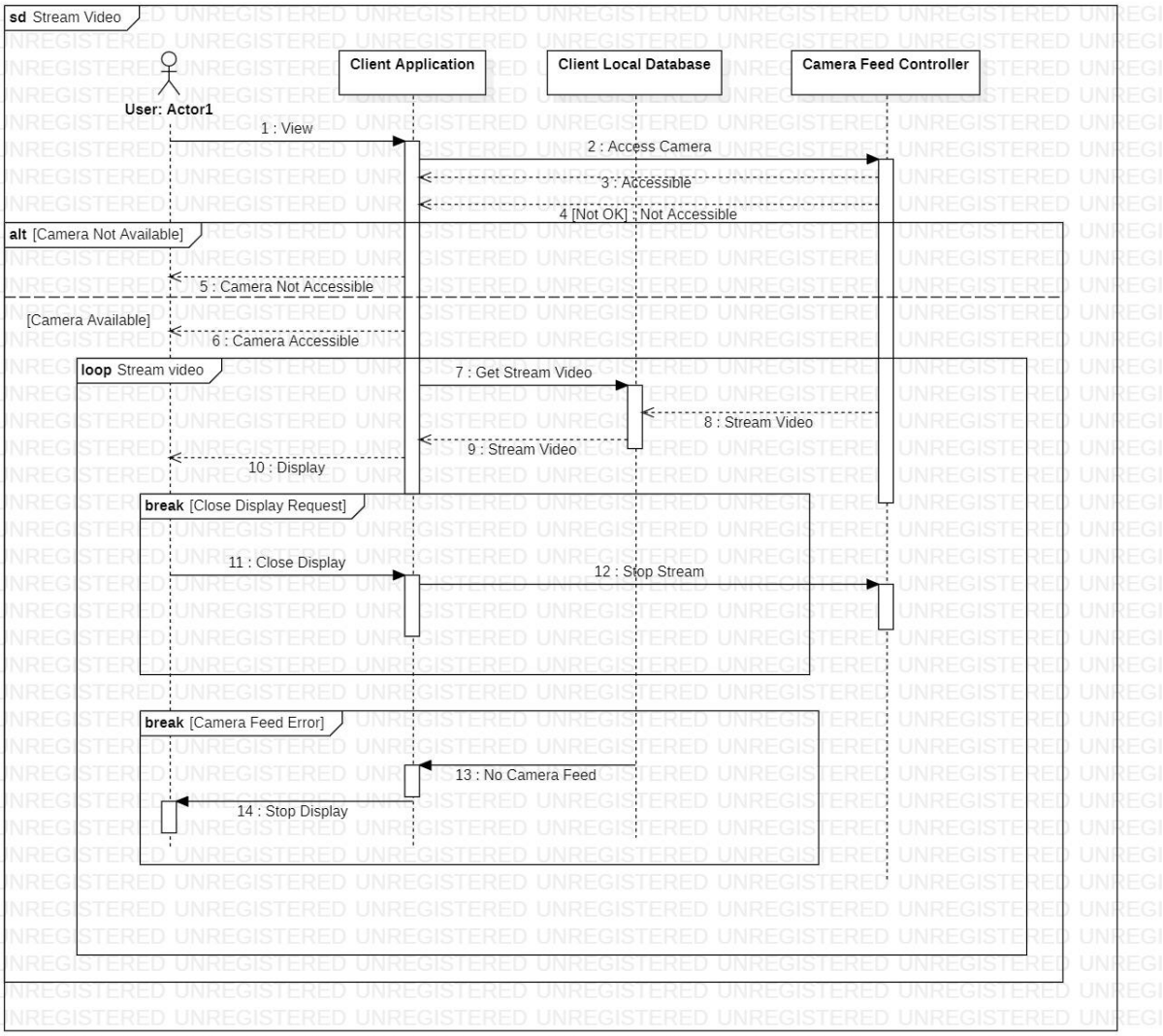

Figure 4.12: Sequence diagram showing the interface between Client Local Database and Camera Feed Controller

Design Rationale:

- This interface requires high bandwidth internet connection. Otherwise, high live streaming latency can mislead users and affect the experiments.
- In case of high network latency, the client application should warn the user.

Interface Between Event Control System and Event Controller (Camera Control)
The Event Control System allows the user to control the camera by communicating with the Event Controller on the server side. Users can capture an image, configure camera settings or get the camera settings. Event Controller System in client application makes desired request from server and event controller in server side sends the desired data to client. After that, received data is stored in the Client Local Database.

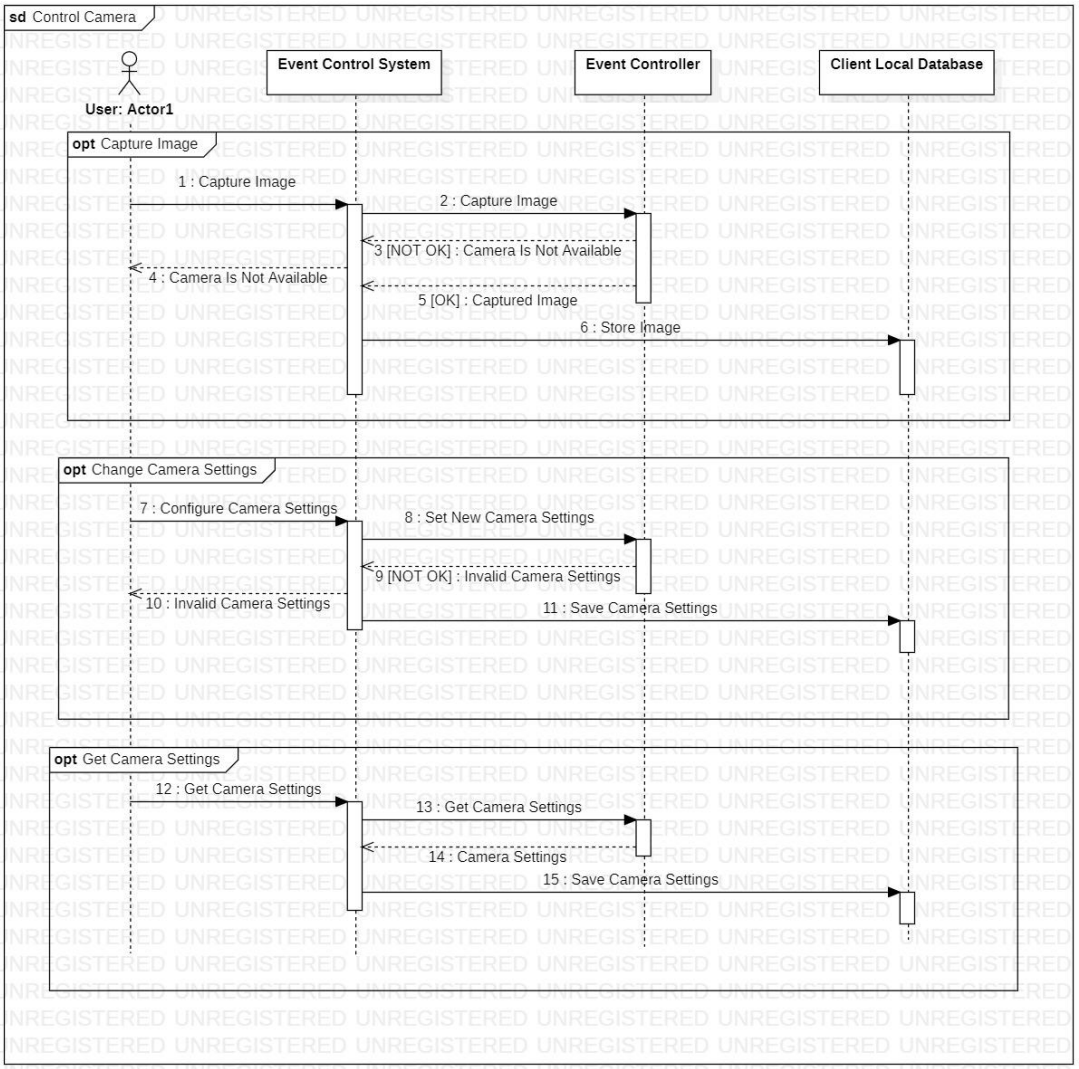

Figure 4.13: Sequence diagram showing the interface between Event Control System and Event Controller (Camera Control)

Design Rationale:

- In case the Event Controller in the server or cameras of the microscope are not accessible, the Event Control System in the client application should warn the user.
- Desired Image format (.png or .jpeg) can also be sent to the server.
- If new camera settings are not possible to realize by the server then the server should send an error to the client.

Interface Between Event Control System and Event Controller (Motor Control)
The Event Control System also allows the user to control the motor by communicating with the Event Controller on the server. Users can move the motor as desired or configure its step size. To move the motor, the Event Control system sends the move motor request with the x,y,z coordinate parameter to the Event Controller in the server. Event Controller makes the motor move in accordance with the delivered parameters, after motion completed, it sends an acknowledgement message to the client. The Client receives the acknowledgement message and updates the new motor options in the local database.

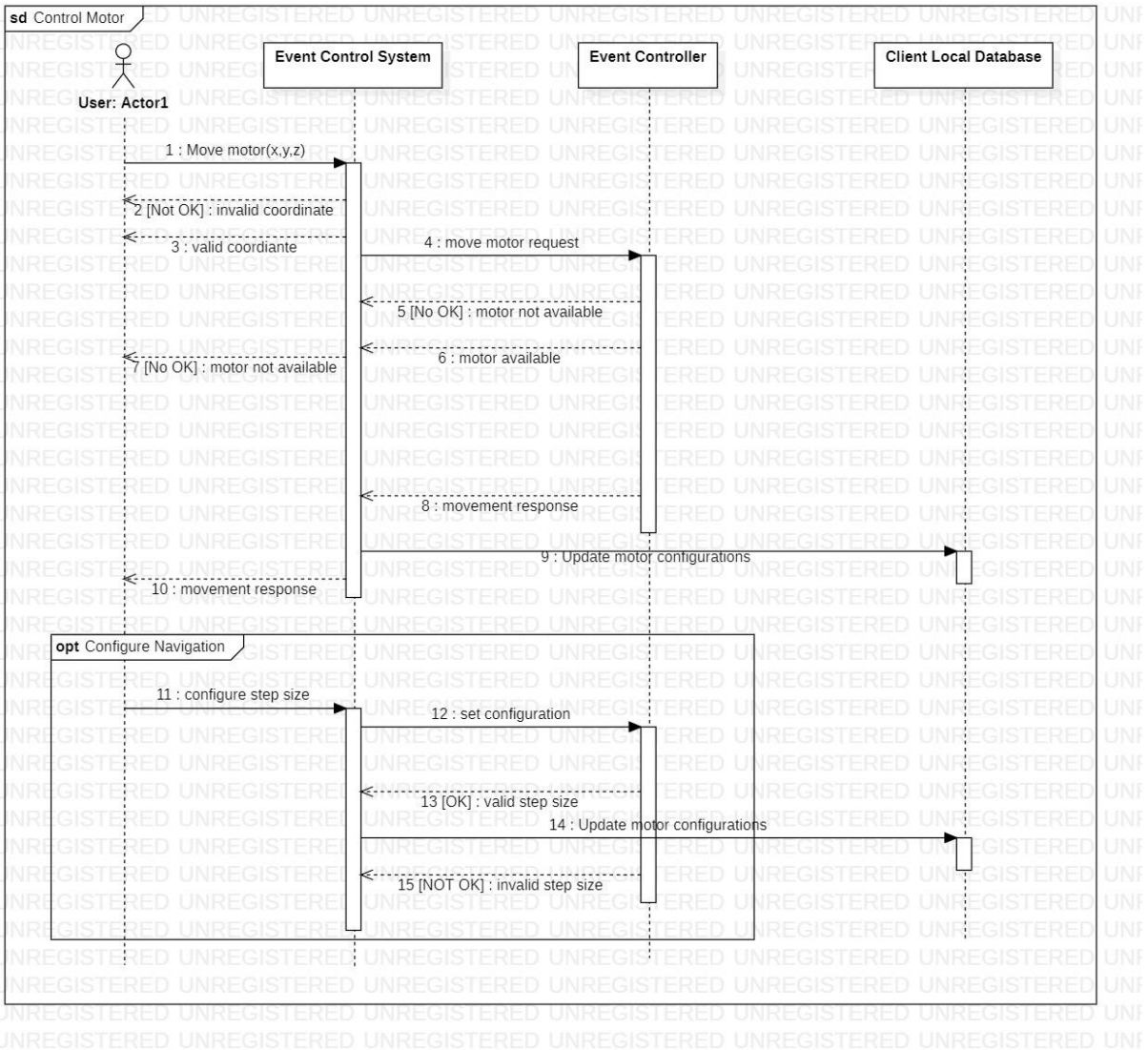

Figure 4.14: Sequence diagram showing the interface between Event Control System and Event Controller (Motor Control)

Design Rationale:

- In case the motor is not available or desired step size is not valid, the Event Controller on the server side sends an error to the client.

- Before the Event Controller System sends the move motor request to the server, it first checks whether it is valid or not. If it is not valid, It doesn't send the request and then it warns the user.