TED UNIVERSITY

TED University

CMPE 491

Virtuanance

Low-Level Design Report

Members

Hakan Ahmet Tekin, Cem Tırpanlı, Ali Can Keskin, Doğukan Terzi

Supervisor

Tolga Kurtuluş Çapın

Jury Members

Aslı Gençtav, Venera Adanova

Table of Contents

1. Introduction

Remote Maintenance is a set of information technology tools that monitors all activity at each workstation and provides information to the professional who is helping the technician. Remote Maintenance typically takes the form of a single interface that provides complete visibility into a client's desktop, AR/VR glasses, and even phones or tablets to maintain hardware components of a machine that is not working. Remote maintenance technology can most of the time tell the difference between critical and non-critical problems and react accordingly. Much of this is undertaken to prevent significant machine failures and provide peace of mind to the organization. [1]

In the project, a Mixed Reality Environment is created and maintained by attaching an external camera to VR glasses, and it is aimed to be user-oriented and user-friendly. Besides, the application is requested not only to maintain a specific object but also to repair any object anywhere. In this way, since this project is not made to maintain a specific product, it can help anyone who wants to maintain an object remotely.

Novelty, and the challenging part is to be able to place a virtual 3D model on an accurate 3D model in the project, and some hand movements on this model are to make a system that shows where the object is to be rotated, or where the place to be maintained. The solution will be reached by showing hand movements with Leap Motion Controller and maintaining tools by creating UI with hand gestures when using Leap Motion Controller.

The rest of this section delves into our decisions about the project's ultimate goals and the concepts we used to document and improve it. The sections that follow define the software's overall architecture and components.

1.1. Object Design Trade-offs

1.1.1. Functionality vs Usability

There are two different UI designs in the project application named technician perspective and professional perspective.

For Technician UI, as a simple principle, a UI design can be made and the functionality can only be provided with their own hands (Leap Motion Controller) At the same time, it is ensured that the hand movements are seen by the professional, and the professional is provided to tell the instructions to the technician in the best way. This increases usability for the technician.

For Professional UI, it is more difficult than the technician's UI. To give the Technician the necessary instructions and get a better 3D view of the object, the professional's UI functionality is higher than the UI design of the technician.

1.1.2. Complexity vs Cost

Expenditures made in a project may be to make that project user-friendly. For our project, it is aimed that those who use the application maintain an object in the simplest ways. Therefore, several technologies are included in the project to use an application with less complexity, such as Leap Motion Controller and Zed Mini external VR Camera.

The versions in which these technologies are not used will also be offered to the clients within the versions. However, in this case, users may have difficulty using the program and may not even fully understand the parts they will maintain.

1.1.3. Performance vs Security

In defense industry projects, security has priority over other requirements. In order for defence industries to integrate technicians and professionals into their systems, P2P connections or their own server implementations are required. In the provision of these network connections, the people who make the project cannot know the types of the users' servers and some performance losses may be experienced. However, the loss of performance can be ignored, as the project team must do it in a way that can scale the network security.

1.2. Interface Documentation Guidelines

In this document, classes are documented with the following components:

1. Package Name: Name of the package the class belongs to.

2. Class Name: Name of the class.

3. Description: Summary of the purpose of the class.

4. Attributes: Public ("+") and private ("-") attributes of the class with their types.

5. Methods: Public ("+"), protected ("#") and private ("-") methods of the class with their signatures.

As evident from above, the guidelines follow the UML principles [2].

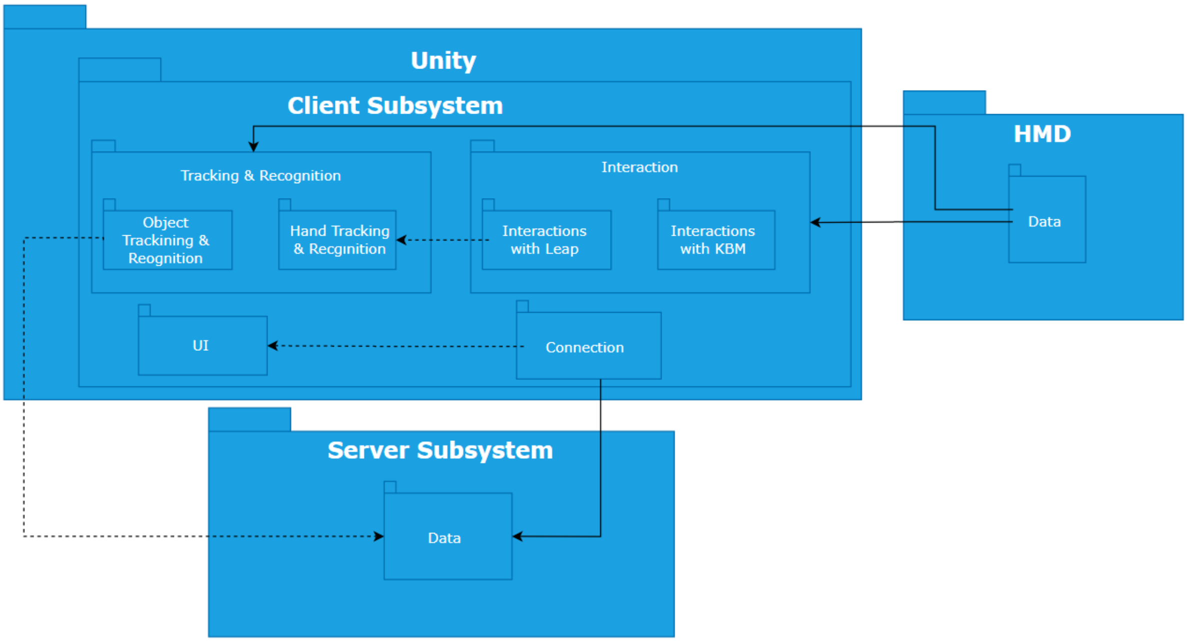
1.3. Engineering Standards

In this document there are two principles adopted. The first of these is the use of UML notation in our diagrams. The second one is the use of IEEE citation guidelines for our referencing and citations [3]

1.4. Definitions, Acronyms and Abbreviations

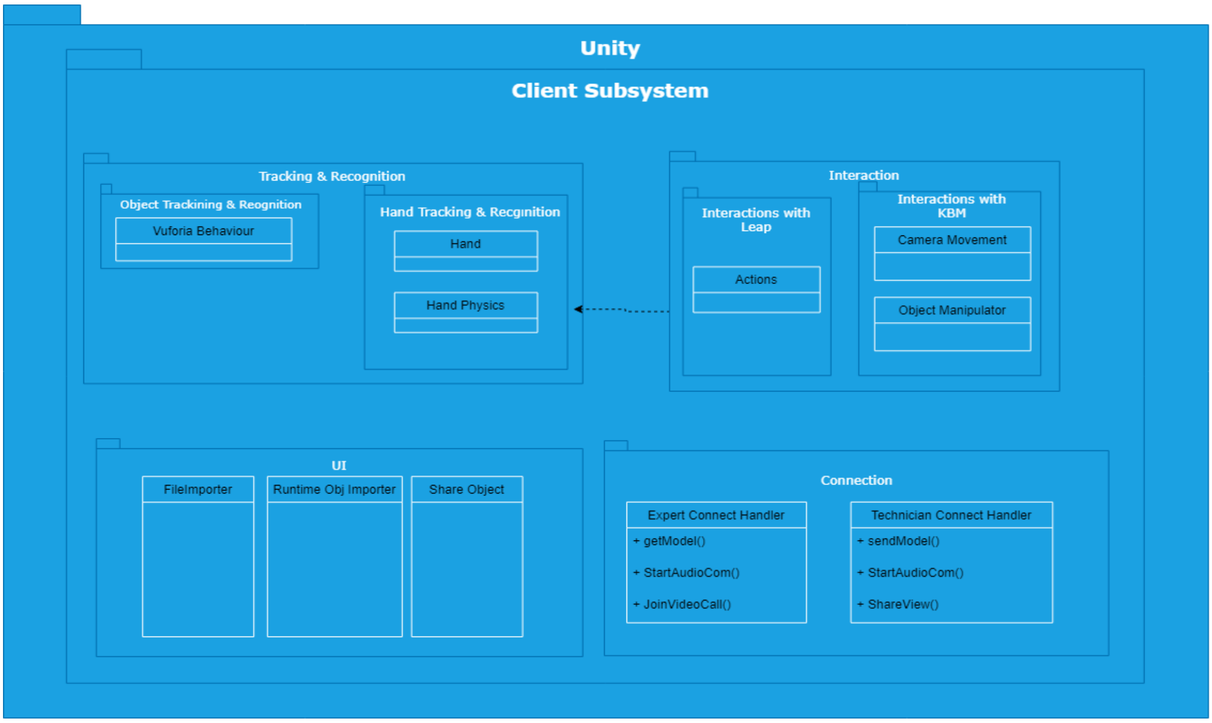
KBM	Keyboard and Mouse: It is an abbreviation indicating the use of Keyboard and Mouse.
MR	Mixed Reality: It is the merging of real and virtual worlds to produce new environments and visualizations.
AR	Augmented Reality: It is an interactive experience of a real-world environment.
VR	Virtual Reality: It is a simulated experience that can be similar to or completely different from the real world.
3D	3 Dimension: Representing a model with his width, height and depth.
UI	User Interface: The interface that user interacts with while using our application.
IEEE	The Institute of Electrical and Electronics Engineers: The engineering and reporting standards used.
UML	Unified Modelling Language: This is the standard diagramming and modeling technique.

2. Packages



Our project can be divided into two packages: Client Layer and Server Layer. Client Layer contains functionalities that handles both visible and background operations for both technician’s and expert's view and experiences. Server Layer is handling all things related between two clients including voice and visual communication and data and object sharing.

In the project different libraries and tools are fused together to produce a helpful and widely applicable solution. This makes it difficult to understand the internal structure of the application and its utilities. This separation will make understanding the application easier. We will explain these packages in more detail in the following sections.



2.1. Unity

The Unity package covers the entire client side of Virtuanance as it was written using Unity libraries and plug-ins. The connection subsystem uses the 'Agora' plug-in for communication between expert and technician and Tracking & Recognition subsystem uses 'Vuforia'. HMD's are connected via Valve's 'SteamVR' plug-in. Most of Virtuanance code takes advantage of Unity's component design pattern to add functionality to objects.

2.1.1. Client Subsystem

Client Layer includes sub-packages that have utilized different libraries in order to provide better experiences for both the technician and the expert. The functionalities and the codes that integrate, utilize, and enhance the libraries in the project that is useful for either the technician or the expert are in these packages. Augmentation of the reality, technician's interaction with augmented objects and world, expert's interaction and annotations on the shared model are examples of the functionalities that is handled in the Client Layer.

2.1.1.1. Tracking & Recognition

This package contains two sub-packages and handles locating and tracking of the real-world objects and elements that will be modeled for the users, namely technician and expert. Both technician and expert will use those tracked models for understanding both the problem and the solution processes together.

2.1.1.1.1. Object Tracking & Recognition

In this package object tracking and recognition is handled with the help of Vuforia library. When technician launches the main screen, the marker that the user uploaded to the system will be found and tracked while it is inside the field of view of the technician. After that, a copy of the model that the user uploaded will be created on the object so that the technician can investigate further or interact and annotate on the object in order to enhance the communication and understanding between him/her and the expert.

2.1.1.1.2. Hand Tracking & Recognition

Hand Tracking layer is responsible from Leap Motion input. The movement of hands and their translation into the digital space is handled. Custom gestures such as pinch or grasp are also detected in this layer and appropriate responses are created. When a hand tries to interact with world objects, this layer notifies the interaction layer.

2.1.1.2. Interaction

The interaction package is one of the most important packages for the ease of use and efficient work targeted by the project. The contents of this package will enable both users to interact with models in many various ways such as rotating, scaling etc.

2.1.1.2.1. Interactions with Leap

Leap Motion Controller translates a user's hand into the digital world of Virtuanance. This layer allows users to touch and control any interactable object within the world (buttons, meshes etc.). This layer provides roughly the same functionalities that are mentioned in 2.1.2.2, but instead of getting input from keyboard/mouse, Leap Motion is used to detect a hands location and its gestures for simulating input.

2.1.1.2.2. Interactions with KBM

This package provides utilities for experts. Expert will have a copy of the object's model so that he can inspect the model clearly and freely while guiding the technician. This package provides functionalities such as scaling, rotating the object as well as moving and rotating the camera that allows expert to see the copy of the model.

2.1.1.3. UI

2.1.1.3.1. Menu Interfaces

This is the entry point of Virtuanance. Each user is presented with a menu to show them options to choose. This layer connects the entire project and directs the user to different components of the project. The UI is designed to work on different screen scales and built with design principles in mind.

2.1.1.3.2. Connection

This is the connection point of Virtuanance. Technician users are given a client ID, which then can be used by the expert user to establish a connection. The UI is designed to be simple, understandable by all the users.

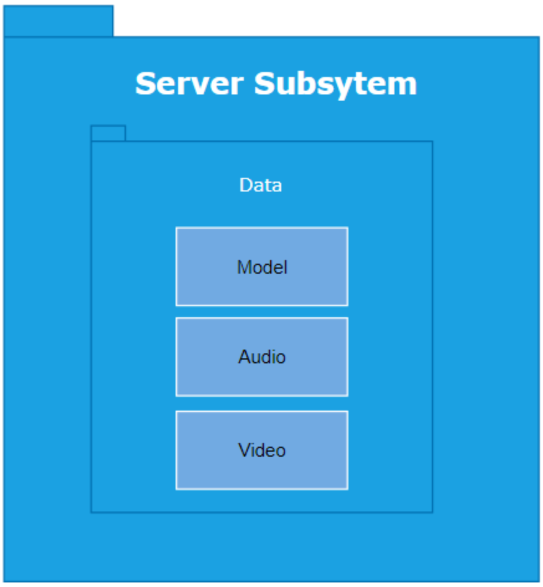
2.1.1.3.3. File Importer

This is where the users can interact with their hard drives and use it to import files to the system. This layer handles all the local storage operations. UI of this layer consists of two scenes, in which one consists of handling the file chooser menu, and the other is helping users to browse their local imported files.

2.1.1.4. Connection

This package handles the connection between expert and technician. Classes inside these packages are responsible for synchronization of models, audio communication between clients and sharing technician's views with the expert.

2.2. Server Subsystem

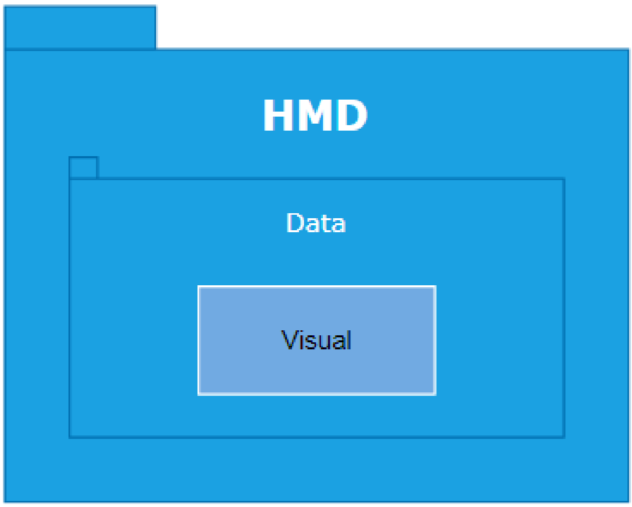


This package contains data of the model uploaded to the system and communication data between two clients.

2.2.1. Data

This package contains information about the virtual models that are going to be displayed on both clients' views, the audio of the clients and visuals that will be shared between both clients.

2.3. HMD



The HMD package directly connects the user (technician side) to the application and it should be able to support various VR glass brands(we used "HTC Vive" while we were doing the project.). In order for HMD data to interact with Unity, a plugin named "SteamVR" will be used. In this way, the application can be used not only for HTC Vive but also with many different VR glasses. The data subsystem is responsible for displaying the digital world of Virtuanance to the user from their perspective. Camera movements are handled in this package.

3. Class Interfaces

3.1. UI

Class – MenuManager
Responsible for handling the menu interactions.
Attributes

-PreviouslySelected: GameObject +OpenPanelAnimator: Animator	
Methods	
+OpenPanel(Animator): void	Opens a panel
-CloseCurrent(void): void	Closes whatever panel was open and selects the previous selected
-SetSelected(gameObject): void	Selects the pressed object’s panel and makes it the opened one
+OnButtonPressed(Button):void	Events for pressed buttons

Class - RuntimeImageTarget	
Class is called on main screen initialization to load markers and corresponding objects from local storage.	
Attributes	
-scriptEnabled: boolean	
Methods	
+loadTargets(): void	Provides a loop for loading targets from storage
-GetRealSize(parent: GameObject): Bounds	Calculates the size of the gameObject, which is currently loading (size = x,y and z size of object)
-CreateImageTarget(): void	Creating the markers and the objects and giving them appropriate vuforia behaviours

Class - ImportedFileManager

The class responsible for creating the screen for viewing the previously imported files.	
Attributes	
+upload, preview, delete: Sprite +buttonPrefab, panelToAttachButtonsTo: gameObject	
Methods	
+Start() : void	While initialization, this function creates all the screen by creating the gameObjects using logs.txt file located in the assets folder.
+deleteClickListener(b: gameObject): void	Basic listener to handle all the clicks on the delete button.
+updateClickListener(b: GameObject, typeOfFile: string): void	Basic listener to handle all the clicks on the update button.
-ShowLoadDialogCoroutine(typeOfFile: string, j: int): IEnumerator	Called by update function, to create new file browsers while updating the markers and objects.
+startPreview(b: GameObject): void	Basic listener to handle all the clicks on the preview button.

Class - FileChooser
This class is used for creating file chooser subscreens when “Upload” button is clicked in the “Upload Files” scene.
Attributes
+uploadButton: Button +markerLocation, objectLocation: String

Methods	
+clickSelect(): void	Basic listener to handle all the clicks on the select button.
-ShowLoadDialogCoroutine(typeOfFile: string): IEnumerator	Called by clickSelect function, to create new file browsers while uploading the markers and objects.
+clickUpload(): void	Basic listener to handle all the clicks on the upload button.
-checkDirectories(toCheck: String)	Used to check if the directory exists, before uploading the file on that directory.

3.2. Tracking & Recognition

3.2.1. Object Tracking & Recognition

Class - VuforiaBehaviour	
This class is the main class of the tracking algorithm that we will use in the project. Using this Class, you can determine the location of objects in the real world.	
Attributes	
#Path : String +StorageType: Enum	
Methods	
+Load(string path, StorageType st): bool	Loads images and makes them Trackable for the Vuforia Engine
#CreateTracakable(TrackableSource trackableSource, String gameObjectName): DataSetTrackableBehaviour	Creates a new trackable behavior attached to a new GameObject with the given name and adds it to this dataset

3.2.2. Hand Tracking & Recognition

Class – Settings Sub Menu	
This class is the controlling hand menu that will appear next to the hands when the required hand signal is made.	
Attributes	
-hand: InteractionHand +MenuPrefab: GameObject	
Methods	
+ToggleMenu(bool): void	Opens or closes the hand menu
-CheckHandPosition(InteractionHand)	Check if hand is in correct position for opening the menu

Class - InteractionButton	
It is a class that allows hands in the Leap Motion Controller to detect and touch 3-dimensional buttons by following the hand.	
Attributes	
-isPressed : bool	
Methods	
+OnPress(void)	Triggered event when a button is ‘pressed’ (collided with an interaction hand object)
+OnDepress(void)	Triggered event when a button is ‘unpressed’ (exited collision with an interaction hand object)

Class – Interaction Hand

Enables interaction with objects that are using the interaction engine of LeapMotion SDK. The class has access to the data output of LeapMotion controller device.

Attributes

-leapProvider: LeapProvider

-handData: Hand

-isTracked: bool

-contactBones: ContactBone[]

Methods

-initContact(void): void	Handles contact bones and other required data when a contact begins
-getGraspPoint(void): Vector3	Returns the point of contact of a grasped object

Class - HandPointer

Enables the utility feature ‘hand pointer’. With it, the user can select objects that are far away from leap hands. Renders a thin red line resembling a laser pointer.

Attributes

-attachedAttachmentHand: AttachmentHand

-handModel: HandModel

-line: LineRenderer

Methods

+ActivateLineTrace(): void	Activates/Deactivates the pointer
-CalculateLinePositionAndRotation() : void	Calculates the transform of the pointer line relative to the hand it is attached to.

+ClickAction: Action	Action event used when a predefined gesture is done by the user (e.g. pressing two fingertips)

3.3. Interaction

3.3.1. Interactions with KBM

Class - Camera Movement	
Expert can control the camera which shows him/her the model of the object next to the technician's view on the expert's screen.	
Attributes	
+mainSpeed: float +shiftAdd: float +maxShift: float +degrees: float +angularSpeed: float #targetCam: Camera	
Methods	
#GetBaseInput(): Vector3	Returns basic directions for camera to move
#RotateCamera() : void	Rotates Camera according to the mouse movement when right mouse button is down
#PivotObject	Checks whether the mouse is on the object when the right button is first clicked. If it is then uses it as a pivot for camera movement.

Class - ObjectManipulator

Enables expert to manipulate the model of the object so that he/she can explain/guide the technician better.	
Attributes	
+rotSpeed: float	
Methods	
#OnMouseDown(): void	Calls GetMouseAsWorldPoint() so that that information can be used when manipulating the object. When the wheel is rotated, it scales the object according to the input.
#OnMouseDrag(): void	Rotates the object according to mouse drag when left mouse button is down.
#GetMouseAsWorldPoint(): Vector3	When mouse is clicked gets mouse's location on virtual world so that it can be used when scaling, rotating and moving the object.

3.4. Connection

Class - ExpertConnectHandler	
This class is used for handling all the connection interactions on an expert's client.	
Attributes	
#AppID: string	
Methods	
+Start(): void	During the initialization, creates the input field.
-CheckAppId(): void	Checks if the current app ID is valid.

-LoadLastChannel(): void	Gets the channel name from the input field.
+HandleSceneButtonClick(): void	After the click on the connect button, moves to the main scene where expert can observe the technicians interactions with the world.

Class - TechnicianConnectHandler	
This class is used for handling all the connection interactions on expert’s client.	
Attributes	
#AppID: String	
Methods	
-CheckAppId() : void	Checks if the current app ID is valid.
+HandleSceneButtonClick(): void	After the click on the connect button, moves to the main scene where technician can interact with the world.

Class - SendToServer	
Method is called whenever the technician interacts with an object, to send the corresponding object’s data to the server.	
Attributes	
-hostName: string	
Methods	
+sendObjectToWebService(g: GameObject) : void	After the technician touches an object, its data is transferred to the live web service with POST request.

Class - RetrieveFromServer	
Method is called whenever the expert clicks “Sync” button, to retrieve any object data current on the server.	
Attributes	
-hostName: string	
Methods	
+retrieveObject(): void	Calls the getGameObject, and spawns the returned object from the server.
-getGameObject(): GameObject	Client sends a GET request to the web service and gets a byte array of the object in the server, then returns in to the caller function.

3.5. Server Subsystem

3.5.1. Data

Data - Model
Description
Model data is the byte array, consisting of the data of the object interacted by the technician, which is then transferred to a web service with POST request (Sending a POST request to www.virtuanance.com/uploadFile with the byte array on body). Then the data can be obtained by sending a GET request with the current channel name. (Sending a GET request to www.virtuanance.com/downloadFile/fgalws)

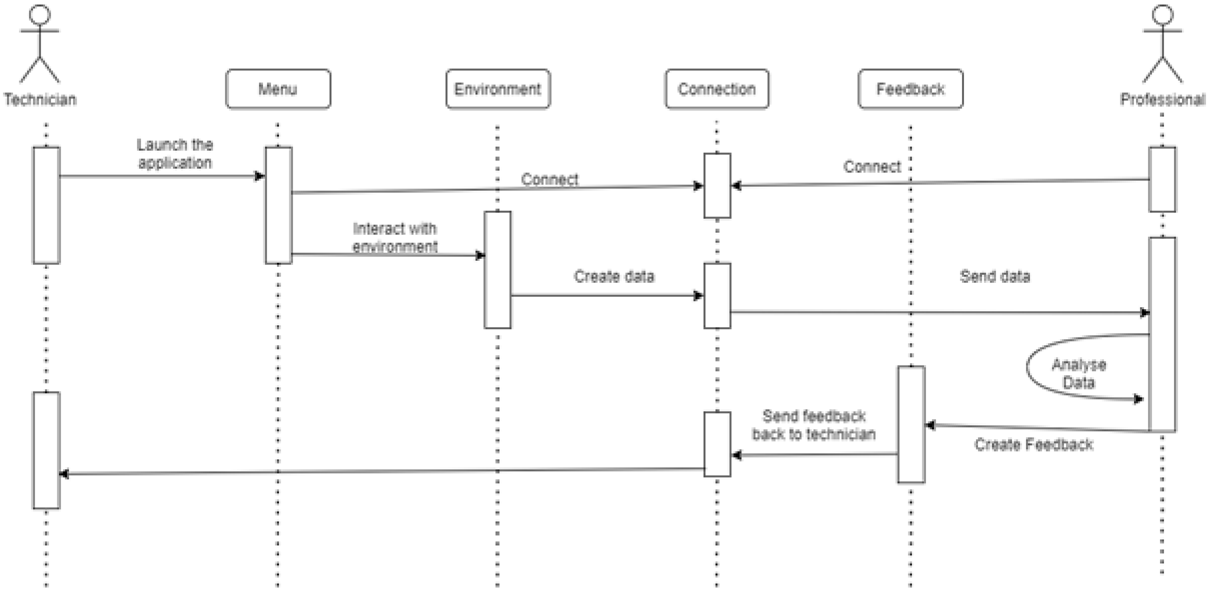
Data - Audio
Description
Audio data is transferred between the users in the same channel. Audio operations are carried by the library “Agora”.

Data - Video
Description
Technicians screen is live broadcasted to the expert in the same channel. All the screen data is transferred, the augmented environment is included. The video broadcasting is handled by the library “Agora”.

4. Sequence Diagrams

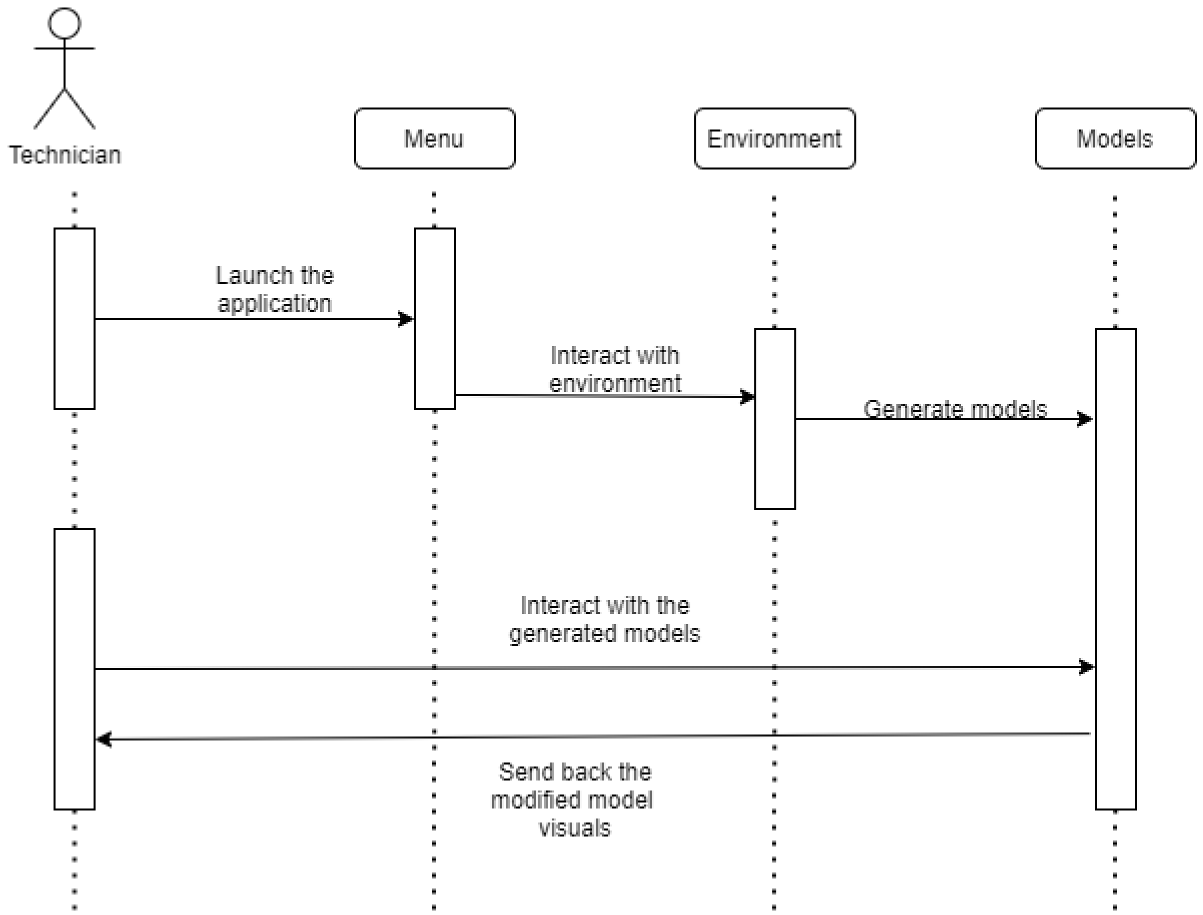
4.1. Scenario #1

The technician and the expert launch the application. They connect to the same room. System establishes their connection. The technician interacts with the environment. Interacted data is sent to the professional. The expert analyses the data, and gives the required feedback by creating animations, or by guiding the technician verbally. After the feedback loop is completed, both users terminate their client applications.



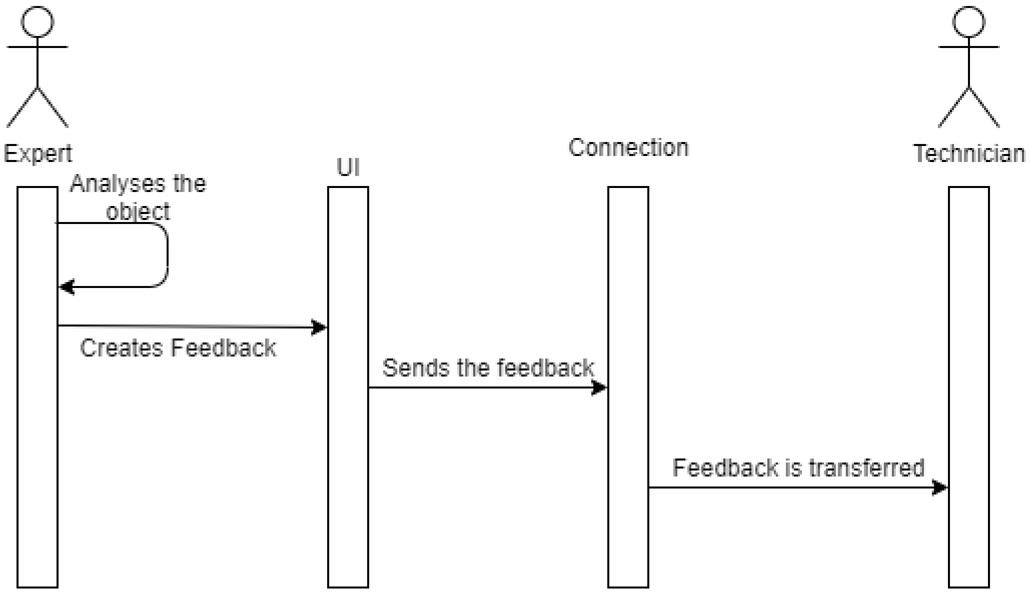
4.2. Scenario #2

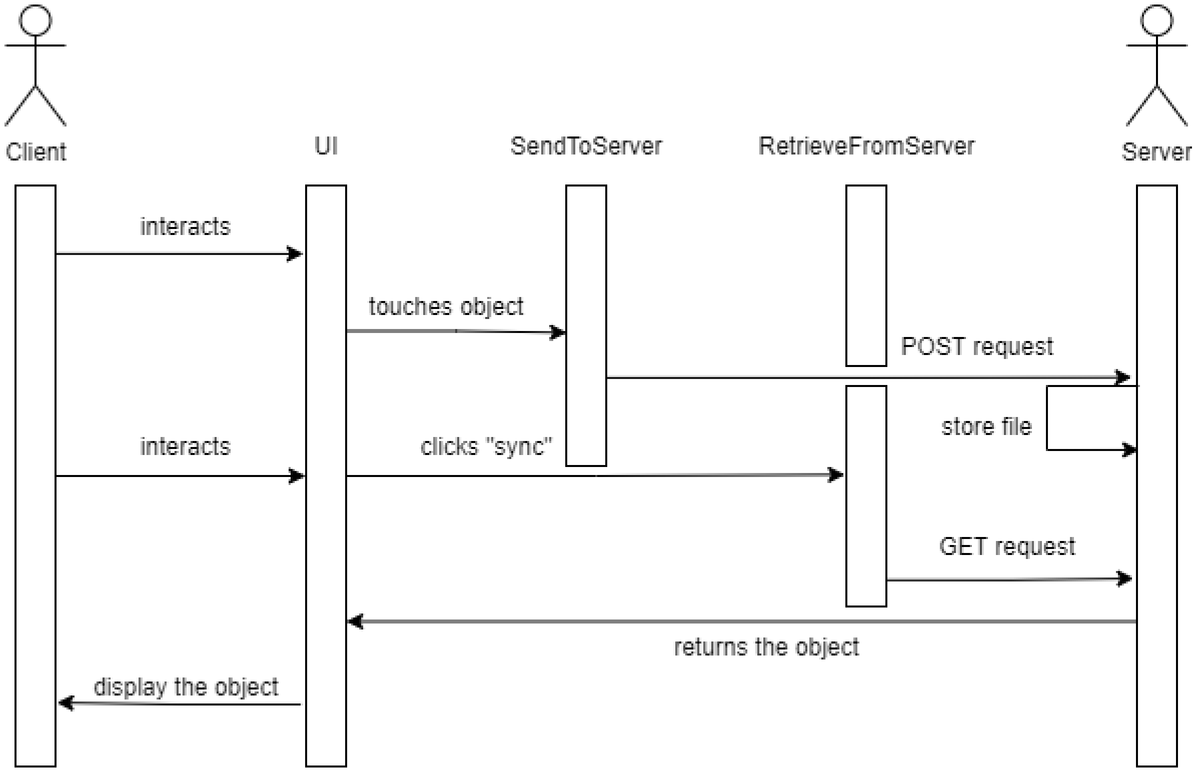
The technician launches the application. Expert is not available to connect right away. The technician starts interacting with the environment. When they select a model, they rotate it, move it, or rescale it in their visual screen, but not in the real environment. They use the program in offline mode to observe the object without interacting with them in real life.


Technician
Menu
Environment
Models
Launch the application
Interact with environment
Generate models
Interact with the generated models
Send back the modified model visuals

4.3. Scenario #3 (From expert view)

Expert is presented with an object technician has just interacted with. Then, after analyzing the object, expert creates feedback with the UI, and the feedback is transferred back to the technician to aid them.





5. Glossary

AR: (Augmented Reality) It is an interactive experience of a real-world environment where the objects in the real world are enhanced by computer-generated perceptual information, sometimes across multiple sensory modalities, including visual, auditory, haptic, somatosensory, and olfactory.

HMD: It is a display device, worn on the head or as part of a helmet (See Helmet-mounted display for aviation applications), that has a small display optic in front of one (monocular HMD) or each eye (binocular HMD).

HTC Vive: The brand of VR glasses that we use for this project.

Leap Motion Controller: it is an optical hand tracking module that captures the movements of your hands with unparalleled accuracy.

Vuforia: It is an augmented reality software development kit for mobile devices that enables augmented reality applications. It uses computer vision technology to recognize and track planar images and 3D objects in real-time.

VR: (Virtual Reality) It is a simulated experience similar to or completely different from the real world. Applications of virtual reality can include entertainment (i.e. video games) and educational purposes (i.e. medical or military training).

6. References

[1]- “What is Remote Monitoring and Maintenance (RMM)?,” *CGS*, 16-Sep-2018. [Online]. Available: https://www.cgsinc.com/en/resources/what-remote-monitor-and-maintenance-rmm. [Accessed: 14-Mar-2021].

[2]- *Modeling Style Guidelines*. [Online]. Available: http://agilemodeling.com/style/. [Accessed: 14-Mar-2021].

[3] - Ieeeauthorcenter.ieee.org. 2021. [online] Available at: <https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf> [Accessed 14 March 2021].