

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
THE UNIVERSITY OF TEXAS AT ARLINGTON**

**ARCHITECTURAL DESIGN SPECIFICATION
CSE 4316: SENIOR DESIGN I
SUMMER 2021**



**VR NURSING TEAM
CAREVR**

**MAX D. CHAN
KYLE DEWEESE
NIKOLAS MURGUIA
ADAM ALBAWAB**

REVISION HISTORY

Revision	Date	Author(s)	Description
0.1	08.16.2021	MC, KD, NM, AA	Initial draft

CONTENTS

1	Introduction	5
2	System Overview	6
2.1	User Interface Layer	6
2.2	Software Layer	6
3	Subsystem Definitions & Data Flow	7
4	User Interface Layer Subsystems	8
4.1	Controller Subsystem	8
4.2	Headset	9
4.3	Base Station	10
4.4	PC	11
5	Software Layer Subsystems	13
5.1	OpenXR Plugin	13
5.2	Game Engine	14
5.3	Scenes	15
5.4	Entity Handler	16
5.5	Component Manager	17

LIST OF FIGURES

1	A simple architectural layer diagram	6
2	A simple data flow diagram	7
3	Controller subsystem description diagram	8
4	Headset subsystem description diagram	9
5	Base Station subsystem description diagram	10
6	PC subsystem description diagram	11
7	OpenXR subsystem description diagram	13
8	Game Engine subsystem description diagram	14
9	Scenes subsystem description diagram	15
10	Entity Handler subsystem description diagram	16
11	Component Manager subsystem description diagram	18

LIST OF TABLES

2	Controller subsystem interfaces	9
3	Headset subsystem interfaces	10
4	Base station subsystem interfaces	11
5	Subsystem interfaces	12
6	OpenXR subsystem interfaces	14
7	Game engine subsystem interfaces	15
8	Scenes subsystem interfaces	16
9	Entity handler subsystem interfaces	17
10	Component Manager subsystem interfaces	18

1 INTRODUCTION

The nursing department is in need of a tool to prepare students to provide care for patients in a hospice setting. This project is designed to accomplish that by simulating experiences that commonly occur when treating a patient in hospice care. These simulations will be virtual reality scenes designed within Unity game engine which will be running on a PC. The targeted scenarios for development include meeting the patient for the first time in the hospital, visiting the patient in their home and conducting safety checks, providing in-home palliative care for the patient, and finally providing postmortem care for the body of the patient after the patient is deceased. These scenes will be populated by objects which will be referred to as entities, and these entities will be organized by components. Components are the data associated with each entity. The user will be able to experience each scene using a VR headset and will also be able to interact with the environment using VR controllers. For the intended device of this project, which is the HTC Vive Pro 2, the movements and inputs of the controller and headset are tracked by two HTC Base station which transmits this data to the PC.

This project has been in development since 2019 and has been worked on by several teams already. Each of these teams was assigned one of the previously mentioned scenarios to develop. In addition to this progress has seen significant disruption due to the situation caused by the Covid-19 pandemic, leading some teams to not be able to develop their scenario for VR. Therefore the current state of this project is a combination of incomplete scenarios which are all separate and operate differently.

In order to get this project back on track our team will be performing systems integration, so our key requirements are to merge previous scenarios into one project, introduce consistent functionality across all scenarios, and in an effort to expand the usability of the project we will be integrating OpenXR which will allow the project to run on more VR devices. To accomplish these task we first must bring each of the previous projects into the same version of Unity. For this our team has chosen one of the current stable builds of Unity which is Unity Version 2020.3. This version of Unity will allow us to utilize OpenXR. Next we must create a menu which will allow the user to select one of the four scenarios and then connect the corresponding scenario to its menu button. Finally, once each scenario has been successfully introduced to our teams project we will then perform consistency improvement across all scenarios including the menu. These improvements include making sure all scenes are VR capable, that player movement/interaction and environment look/feel are the same, and that the underlying systems and objectives are consistent.

2 SYSTEM OVERVIEW

The system overview of our project consists of two high level layers. These layers include the user interface layer as well as the software layer. The user interface layer is made up of the systems which enable the user to input data to affect the virtual reality environment and receive feedback on what those effects are. The software layer handles what occurs in the virtual reality environment, including how user input affects it, and processes how it should be updated. It then sends the updated environment back to the user through the PC.

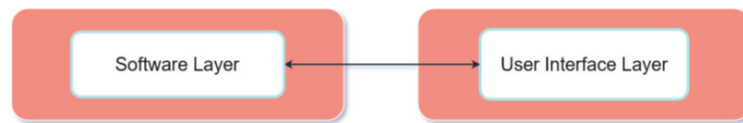


Figure 1: A simple architectural layer diagram

2.1 USER INTERFACE LAYER

The User Interface layer is comprised of all of the physical devices that the user will need to interact with the VR environment. You can think of this layer as the input/output layer. The user will input data through the VR controllers and headset. These inputs consist of user movement and button presses. For the HTC Vive Pro 2, these inputs are tracked by the HTC base station which will then in turn transfer that data to the PC. The PC then sends this data to the Unity engine for processing which will return back updates. The information returned from the Unity engine includes data for the visuals of the environment, objects in the environment, and effects of the player interactions. This information is then output to the user through the VR headset allowing the user see what is happening in the VR environment.

2.2 SOFTWARE LAYER

The software layer consists of all of the software tools needed to generate the virtual reality environment and process updates as the user interacts with the system. The OpenXR plugin maps inputs and outputs between various VR headsets and controllers and the Unity engine allowing the project to be used on a wider variety of VR devices. The Unity engine gathers the mapped data from OpenXR and asset data stored locally and inputs that into the current scene for processing while in return it takes updates from the scenes and sends it back through OpenXR for mapping to eventually update the user on how the VR environment has changed. The engine also provides critical functionality for things like physics, collision, and rendering. Scenes are comprised of all of the entities that populate the 3D environment. Scenes take the input data from the game engine and transfers that to the affected entities. The entity handler processes all of the communication between the different entities and sends that data to the component manager. The component manager processes what events should occur like movement, physics, and collision by using the associated scripts. This data is then sent back to the entity handler which will then update the state of the affected entities, and finally this will be updated in the scene.

3 SUBSYSTEM DEFINITIONS & DATA FLOW

The PC acts as an intermediary between the software and the I/O devices. It receives input from the controllers, base stations, and headset about the user's actions and location. In the user interface layer the PC is the hub between reality and software. Using that information the PC connects the I/O devices to the game engine in the software layer through the OpenXr plugin. The software layer uses the input and the PC's computing power to produce output through carefully crafted scenes and code. The scenes contain entity and component managers that interact with objects and the user within the scene to produce the user experience. This output is then relayed back to the user through the haptic feedback on the controller and the audiovisual feedback from the headset.

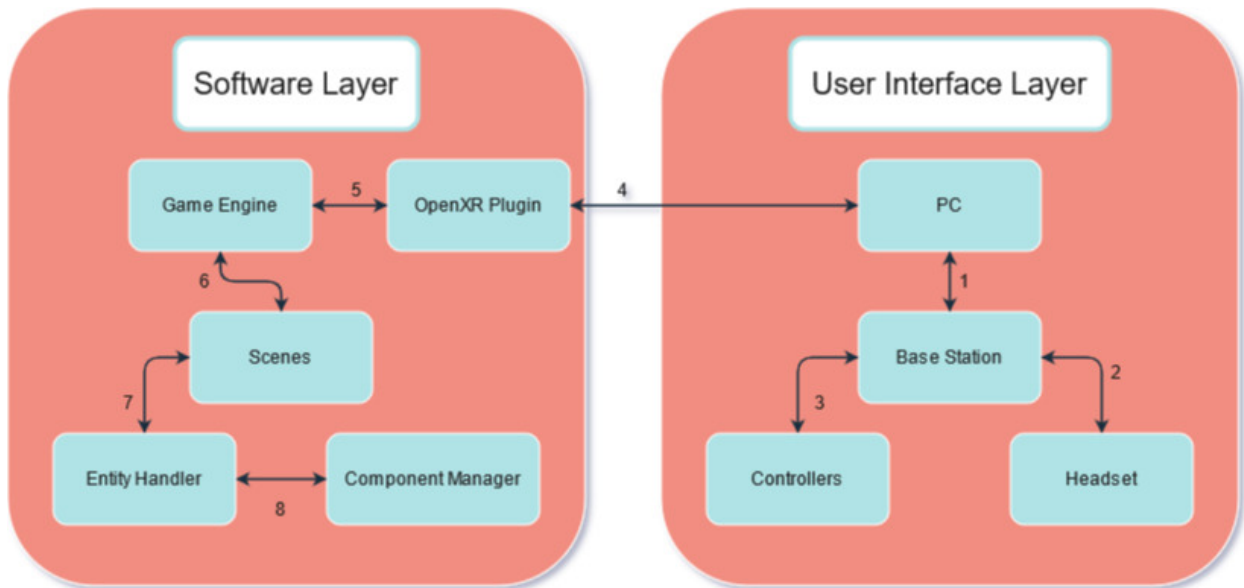


Figure 2: A simple data flow diagram

4 USER INTERFACE LAYER SUBSYSTEMS

The User Interface layer accounts for all of the devices that the user will interact with. This includes the PC, the controllers, the base station, and the headset. This layer takes input from the user and forwards the data to the software layer. The layer also takes in game updates from the software layer and displays that information to the user.

4.1 CONTROLLER SUBSYSTEM

As this is a VR project, input must include both button presses and user hand gestures. The VR controllers will allow the user to interface with the VR scenarios using both of these methods. The project is being designed for use with the HTC Vive Pro II controllers in mind, but because we are integrating the OpenXR plugin for Unity other controllers like the Oculus Quest II should be supported as well.

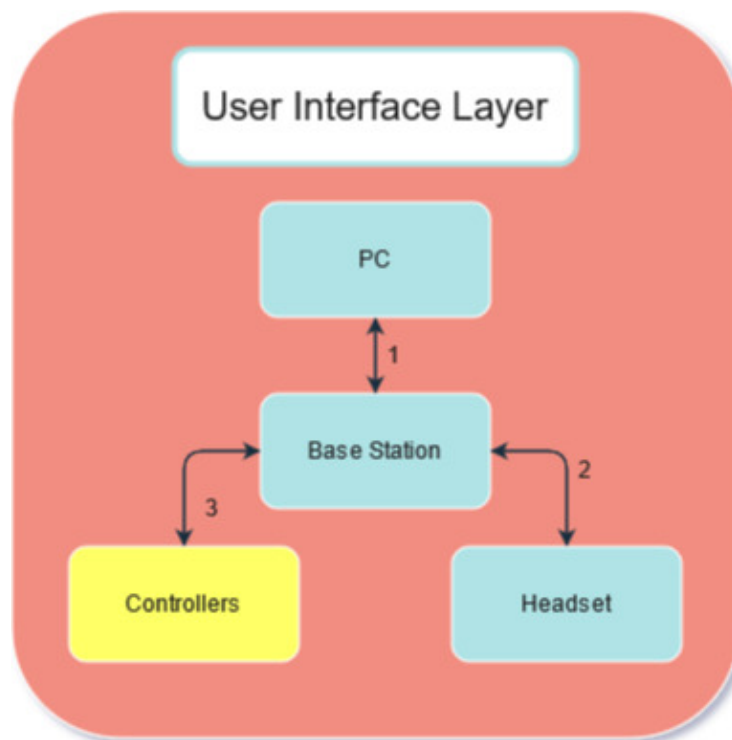


Figure 3: Controller subsystem description diagram

4.1.1 ASSUMPTIONS

We assume that the controllers being used are charged or plugged in, that there are no software/hardware issues with the controllers, and that the base stations are properly setup to allow for proper tracking of the controllers. It is also assumed that the user is capable of utilizing both of the VR controllers to interact with the VR environments.

4.1.2 RESPONSIBILITIES

The responsibility of the controllers is to allow the user to interact with objects within the VR environment through movements and button presses. This includes selecting destinations for user movement, interacting with objects, and selecting menu options.

4.1.3 SUBSYSTEM INTERFACES

Table 2: Controller subsystem interfaces

ID	Description	Inputs	Outputs
#3	Interface with base station subsystem	N/A	User actions Hand movements Button presses

4.2 HEADSET

Raw input from VR headset will be collected by the PC to update the game state for each frame. Headset and PC has bidirectional relationship since PC will also send frame updates to the headset. Headset input will also be collected by the Base Station. Base Station ensures the accuracy of headset direction and location. Headset also displays input from Controllers.

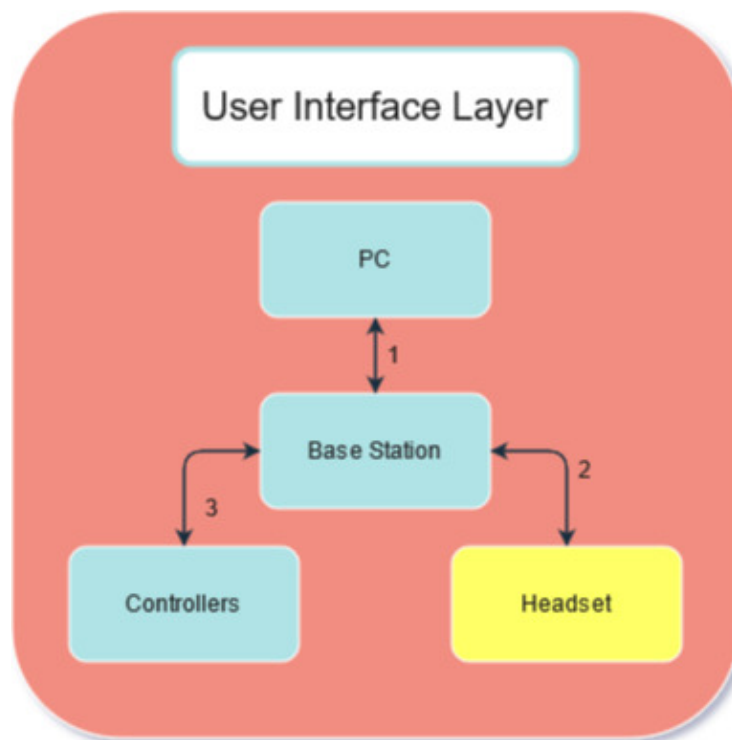


Figure 4: Headset subsystem description diagram

4.2.1 ASSUMPTIONS

We made an assumption that PC and Base Station will process input from headset correctly as well as headset will receive input from PC correctly. We also assume that user has access to headset. We assume that there are no mechanical issues from VR equipment.

4.2.2 RESPONSIBILITIES

Inputs and outputs are used to allow user to interact with simulation environment. Accurate headset direction, location and low latency are keys for immersive simulation experience and user satisfaction.

4.2.3 SUBSYSTEM INTERFACES

Table 3: Headset subsystem interfaces

ID	Description	Inputs	Outputs
#2	Data Flow with Base Station	N/A	Updated frame Headset direction Headset location

4.3 BASE STATION

Powers the presence and immersion of room-scale virtual reality by helping the headset and controllers track their exact locations. This is done through wireless syncing and Bluetooth as well as sensors.

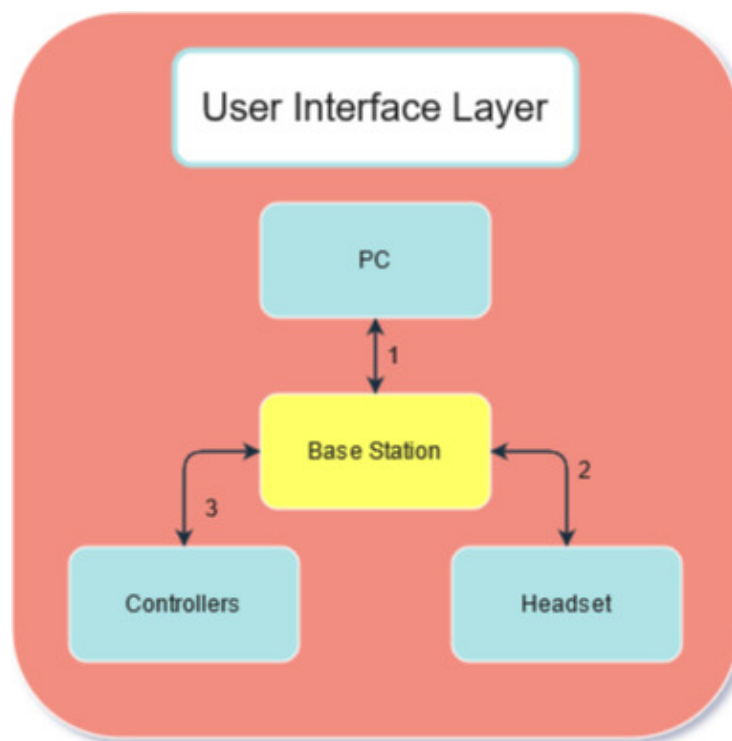


Figure 5: Base Station subsystem description diagram

4.3.1 ASSUMPTIONS

The base stations are placed and calibrated correctly. The base stations communicate with the controllers and headset via Bluetooth technology.

4.3.2 RESPONSIBILITIES

The base stations are responsible for triangulating the user's position through sensors. This information is used to determine the location of the user and used in simulations and games.

Table 4: Base station subsystem interfaces

ID	Description	Inputs	Outputs
#1	PC interface	Setup application	User location Hand movements
#2	Headset interface	Player location	N/A
#3	Controller interface	Player location Player actions	N/A

4.4 PC

The PC receives information from the user via the headset and the base station. It also hosts the VR application and manages all the necessary information for the VR equipment to operate.

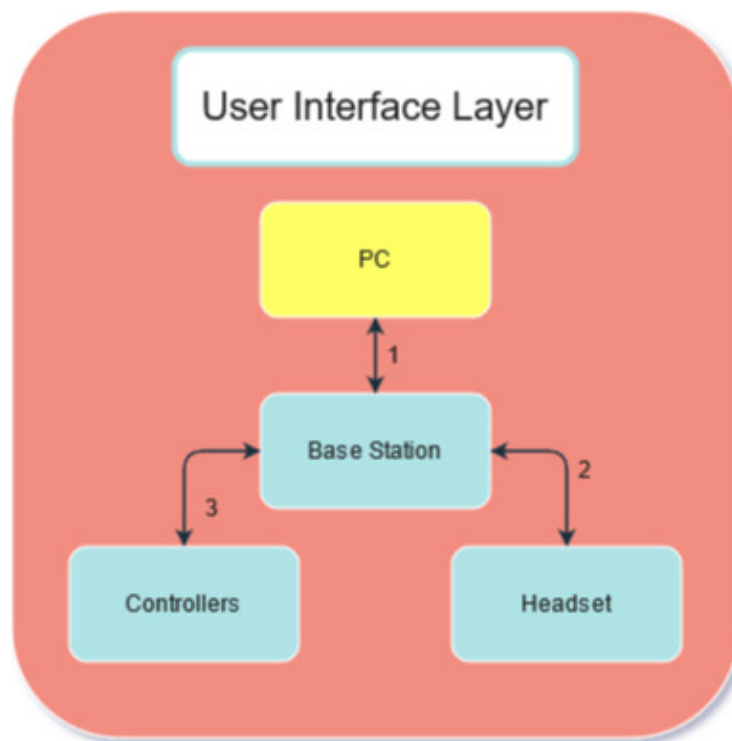


Figure 6: PC subsystem description diagram

4.4.1 ASSUMPTIONS

The PC will contain hardware and software capable of running and hosting the VR application and VR equipment.

4.4.2 RESPONSIBILITIES

The PC is responsible for hosting the VR application and ensuring it has the necessary resources to operate. The PC is also responsible for storing user information such as player movements and actions and sending this information to the Unity game engine.

4.4.3 PC SUBSYSTEM INTERFACES

Table 5: Subsystem interfaces

ID	Description	Inputs	Outputs
#1	Base Station Interface	User location Hand movements	Setup application
#4	OpenXR Plugin Interface	Application graphics Game update	Player movement Player actions User movement

5 SOFTWARE LAYER SUBSYSTEMS

The Software Layer is the application layer of the project. It manages scenes, assets, and game updates as well as providing an interface for interaction with the hardware and user interface layer. This layer takes in user's actions and location and outputs game and graphics updates.

5.1 OPENXR PLUGIN

The OpenXR Plugin serves as the interface between the Unity application and the VR system. The plugin handles various functionality including frame composition, peripheral management, and raw tracking information. OpenXR also allows cross platform VR functionality for application use.

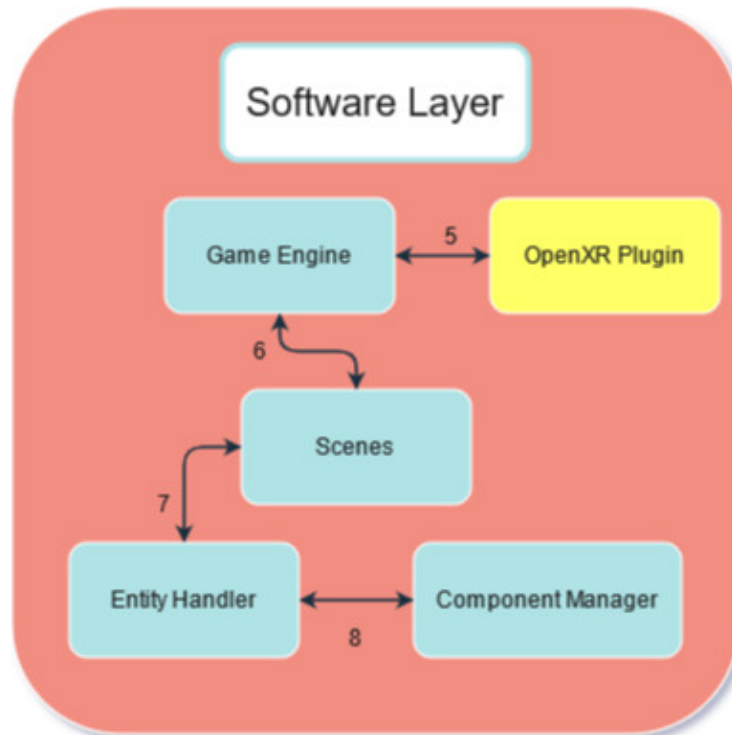


Figure 7: OpenXR subsystem description diagram

5.1.1 ASSUMPTIONS

Both the Oculus Quest II and the HTC Vive can be used with OpenXR. The plugin is available for the 2020.3 version of Unity.

5.1.2 RESPONSIBILITIES

OpenXR manages input from the VR system such as the controller location, various tracking positions. The plugin is also responsible for ensuring the PC receives the rendered frames to be used for the user's graphics.

5.1.3 SUBSYSTEM INTERFACES

Table 6: OpenXR subsystem interfaces

ID	Description	Inputs	Outputs
#05	Game Engine	Application graphics Game update In-Game movement	Player movement Player actions
#04	PC	Player movement Player actions User movement	Application graphics Game update

5.2 GAME ENGINE

The Game Engine manages all scenes and assets as well as being the location for all necessary plugins and extensions. It enables the VR application to have physics, collision, VR implementation, and general game aspects.

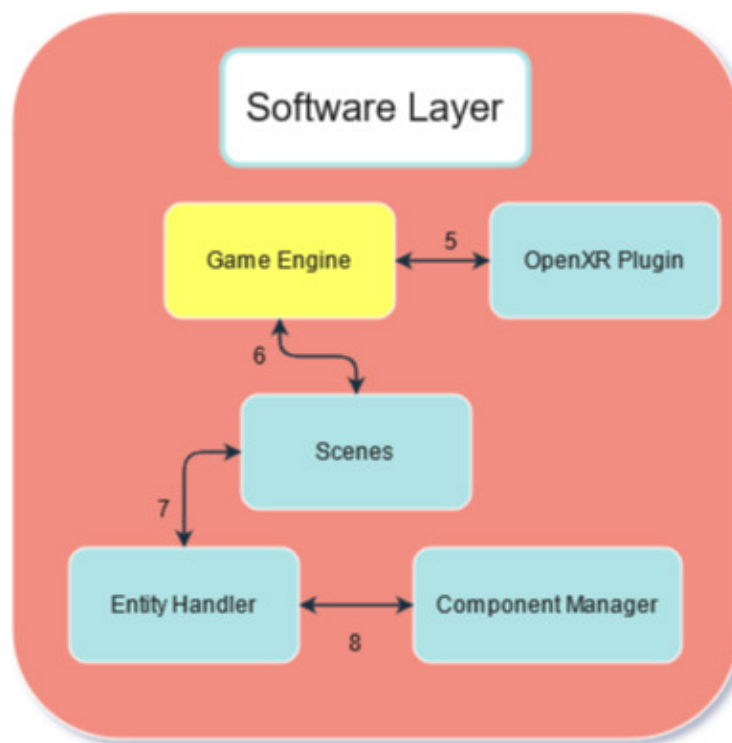


Figure 8: Game Engine subsystem description diagram

5.2.1 ASSUMPTIONS

The Unity Game engine will support OpenXR as well as be able to provide a functional end product for academic use.

5.2.2 RESPONSIBILITIES

The Unity Game Engine is responsible for managing all assets that the application uses including materials and game objects. The game engine also provides necessary features such as physics, collision, and 3-D rendering.

5.2.3 SUBSYSTEM INTERFACES

Table 7: Game engine subsystem interfaces

ID	Description	Inputs	Outputs
#05	OpenXR Plugin	Player movement Player actions	Application graphics Game update In-Game movement
#06	Scenes	Game update In-Game movement	Player movement Player actions

5.3 SCENES

Updates the states for all the entities in the game, including the player. It also calculates physics and interactions. Update happens one time for each frame. [1]

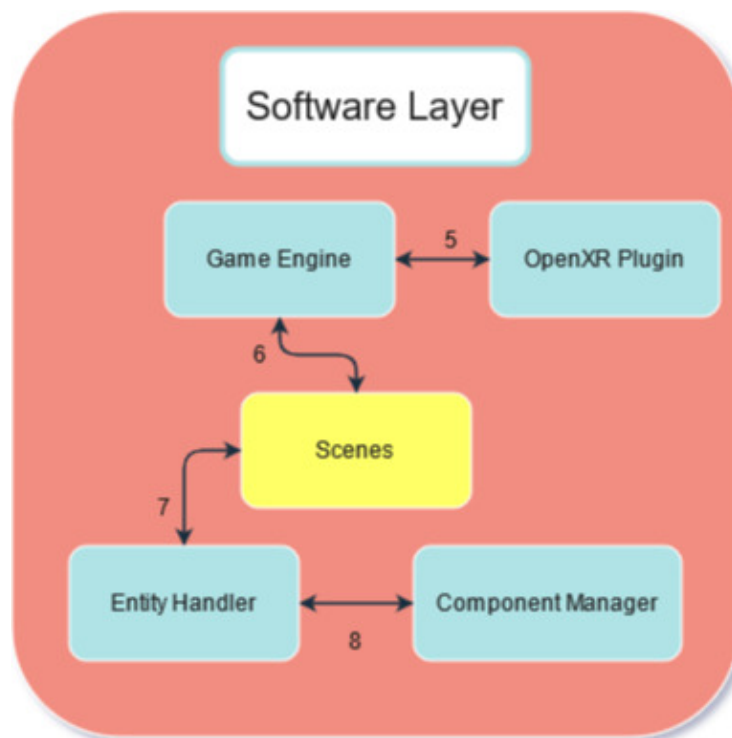


Figure 9: Scenes subsystem description diagram

5.3.1 ASSUMPTIONS

The Scenes receives all the information it needs from the information it needs from the Entity Handler. To reach a frame rate of 60 frames per second, updating all the entities and drawing them to the screen

should all happen within 16.6 milliseconds for each frame. [1]

5.3.2 RESPONSIBILITIES

All entities, physics, and interactions must be updated for each frame. The Scenes will not need to store any of this information, only pass it down to the Entity Handler. [1]

5.3.3 SUBSYSTEM INTERFACES

Table 8: Scenes subsystem interfaces

ID	Description	Inputs	Outputs
#6	Game Engine	User actions User movement	Game Update In-game movement
#7	Entity Handler	Entity information User input affecting entities [1]	Entities to be updated [1]

5.4 ENTITY HANDLER

The Entity Handler subsystem is responsible for every entity communication in the game. Most of the entities must interface with the Entity Handler in order for the subsystem to effectively manage all data pass between each entity. This includes: Player, NPC, Patient, Listener, General Objects, and Camera. The Entity Handler will also correspond with the Component Manager, which will update new information about entities and then pass these updates to the Scenes. [1]

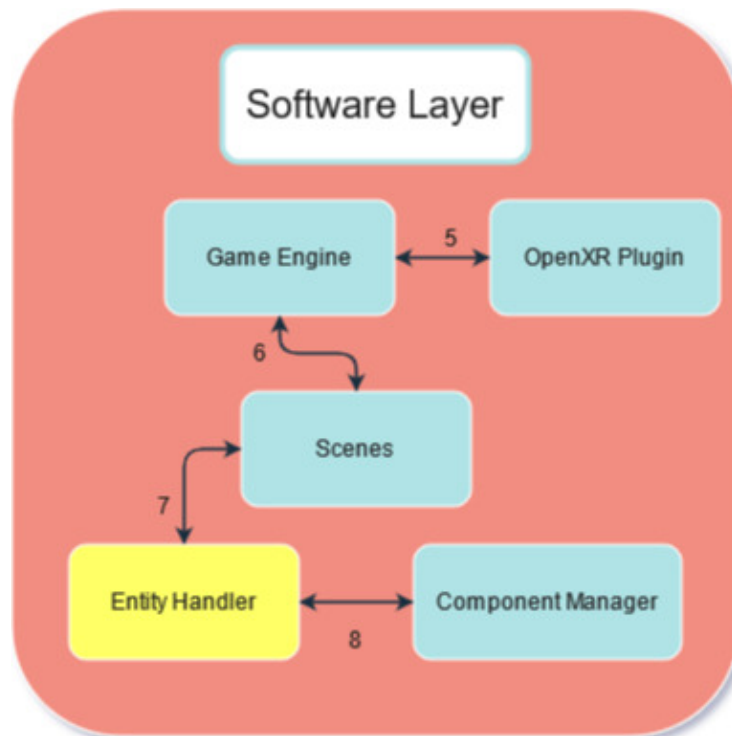


Figure 10: Entity Handler subsystem description diagram

5.4.1 ASSUMPTIONS

It is assumed that Entity Handler handles all information correctly.

5.4.2 RESPONSIBILITIES

The Entity Handler must manage all entities in the simulation. Every change made to entity data must be handled by this subsection. It is also used for sending and retrieving entity data to and from the Component Manager and Scenes. [1]

5.4.3 SUBSYSTEM INTERFACES

Table 9: Entity handler subsystem interfaces

ID	Description	Inputs	Outputs
#7	Scenes	Receive data on entities to be updated [1]	Send updated entity information. User input affecting all entities, including Player and NPCs [1]
#8	Component Manager	Updated Player data, Patient data, Camera data, Listener data, General Objects data [1]	Current Player data, Patient data, Camera data, Listener data, General Objects data [1]

5.5 COMPONENT MANAGER

The Component Manager deals with the attributes of the subsystems handled by the Entity Handler and their behaviors in the world. The smaller subsystems of the Component Manager are the interactivity scripts, movement scripts, physics, and collision. [1]

5.5.1 ASSUMPTIONS

The component managers reads the data from other subsystems, but cannot alter any of them. [1]

5.5.2 RESPONSIBILITIES

The component manager is responsible for gathering the data from other smaller subsystems that are requested by the Entity Handler [1]

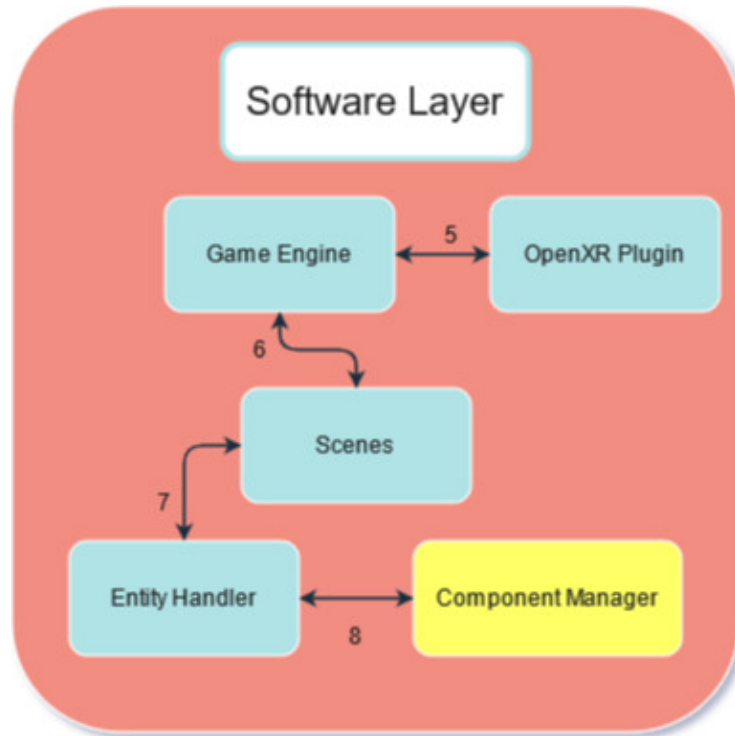


Figure 11: Component Manager subsystem description diagram

5.5.3 SUBSYSTEM INTERFACES

Table 10: Component Manager subsystem interfaces

ID	Description	Inputs	Outputs
#8	Entity Handler	input 1: Request from the Entity Component for a component's information [1] input 2: Data from smaller subsystems within Component Manager [1]	The requested by Entity Handler data [1]

REFERENCES

- [1] VRx - CSE Senior Design. <https://blog.uta.edu/cseseniordesign/2021/05/04/vrx-3/>. Accessed: 2021-08-12.