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

ARCHITECTURAL DESIGN SPECIFICATION I
CSE 4317: SENIOR DESIGN II
FALL 2022

VR NURSING
VR PALLIATIVE CARE

MUSTAFA ABOUL-JIBIN
KENNETH GARZA
MAHDY JOUDEH
BEVAN PHILIP

Revision History

Revision	Date	Author(s)	Description
0.1	07.15.2022	MA, KG, MJ, BP	document creation
0.2	07.17.2022	MA, KG, MJ, BP	initial draft
0.3	07.22.2022	MA, KG, MJ, BP	modified introduction
1.0	07.27.2022	MA, KG, MJ, BP	finalized version 1
2.0	08.14.2022	MA, KG, MJ, BP	Start of revising of version 2
2.1	08.14.2022	MA, KG, MJ, BP	modified introduction
2.2	08.14.2022	MA, KG, MJ, BP	finalized version 2

CONTENTS

List of Figures

List of Tables

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 the Unity game engine and will be able to run in one of two ways: run directly via Unity or packaged into an APK then uploaded and run on the headset. The targeted scenarios for development include meeting the patient for the first time in the hospital, visiting the patient's 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 they have passed. These scenes will be populated by objects which will be referred to as entities, and these entities will be further organized into 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. This project is intended to be used with any VR equipment compatible with OpenXR. Development and testing will be accomplished via the Oculus Quest 2.

The end goal is for the simulation to be used across many universities in order to contribute to the knowledge and experience of palliative care among nursing students as a whole. As a short term goal, this VR simulation will be used by the UTA undergraduate nursing students who will perform one of the four scenarios per semester while completing the undergraduate nursing program. The simulation may be completed with a solo user or with a trainer. 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, progress has seen significant disruption due to the situation caused by the COVID-19 pandemic, meaning some teams did not develop their scenarios for VR.

The previous Senior Design 2 Fall 2021 team has performed systems integration by combining functioning aspects of the project into a cohesive product. This includes modifying and fixing scenarios 1 and 2 in order to meet all requirements needed for each scenario and creating a fourth scenario in accordance with project requirements. This team also planned to or have already added quality-of-life (QoL) improvements to enhance simulation experience such as a time system with which the user can see the time and record the amount of time spent completing a task, adjusting the communication interface in order to fit the most amount of text as possible, and implementing voice acting for all non-playable characters (NPCs). Other QoL improvements include implementing a fast travel system to decrease time spent on travelling across the play space and to minimize dizziness and other symptoms typical of playing VR, as well as including a list of high-level objectives for the player in case they are unsure of what needs to be done next. The latter QoL improvement will not give hints to tasks that are worth points.

The primary objective of the Senior Design 1 Summer 2022 team is to perform a final requirement, performance check for each scenario, and testing each scenario to find bugs, as well as modifying scenario entities to be more consistent across scenarios in order to increase simulation immersion. Any QoL improvements mentioned previously that have not yet been implemented will also be done by this team.

2 System Overview

The system overview of our project consists of two high level layers: the User Interface Layer and the VR Simulation 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. Depending on the method used to run the simulation, the VR Simulation Software Layer either contains or creates the environment the user will interact with, dictates what happens when components of the environment are interacted with, and outlines the events that the user will experience. Users will be able to run the program in one of two ways: directly via the Unity game engine or packaged into an APK that is uploaded and run on the VR headset.

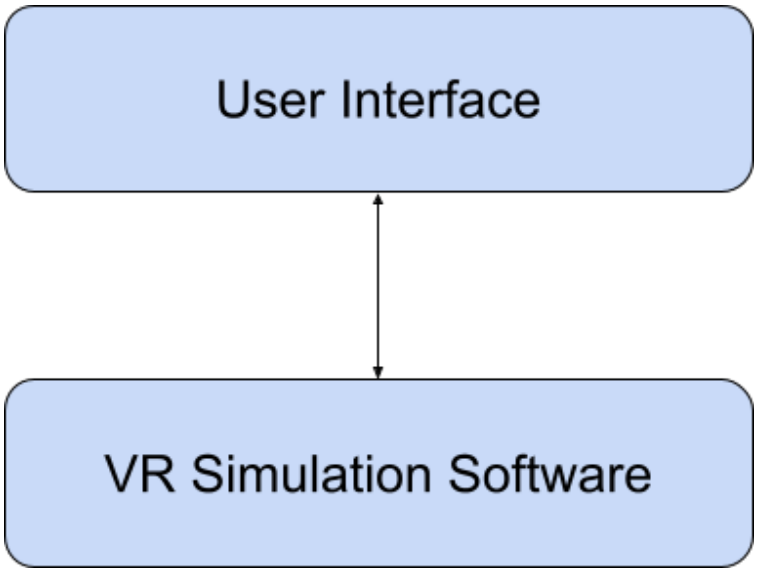

Figure 1: A simple architectural layer diagram

2.1 User Interface Layer

The User Interface Layer consists of all actions that are made by there user while using the VR hardware. This layer specifies any and all forms of potential input the user can give to the software. This includes the accelerometer, gyroscope, and cameras in the headset that help it determine the user's location and the direction the user's head is facing, the buttons on the hand controllers along with the controller's position, as well as many other sensors and controls. When running the simulation via Unity, this is also where the game engine will send the visual and audio output that is generated to the user so that the user can then respond. Otherwise, these will be generated by the software uploaded to the headset. Visual text boxes, images, HUD information, amongst many other aspects of this project are all communicated to the user through this layer. There will be two types of users, the simulation proctor and the student partaking in the simulation. The interface for each type will remain mostly the same, however some minor differences include point system and log of actions taken.

2.2 VR Simulation Software Layer

The VR Simulation 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 allows 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 update the scene. When packaged, these things will be done by the APK uploaded to the headset.

3 Subsystem Definitions & Data Flow

Each high-level layer of the system overview is composed of a group of subsystems that detail how data flows within and between each layer. Within the User Interface Layer are the controllers, headset, and PC, while the VR simulation software layer is composed of the OpenXR plugin, game engine, scenes, entity handler, and component manager. As displayed in Figure 2, data flows unidirectionally from the controllers to the headset and bidirectionally between the headset and PC. The PC is where data flows between the User Interface Layer and VR Simulation Software Layer via the OpenXR plugin. There, data has bidirectional flow between the OpenXR plugin and game engine, game engine and scenes, scenes and entity handler, and finally the entity handler and component manager.

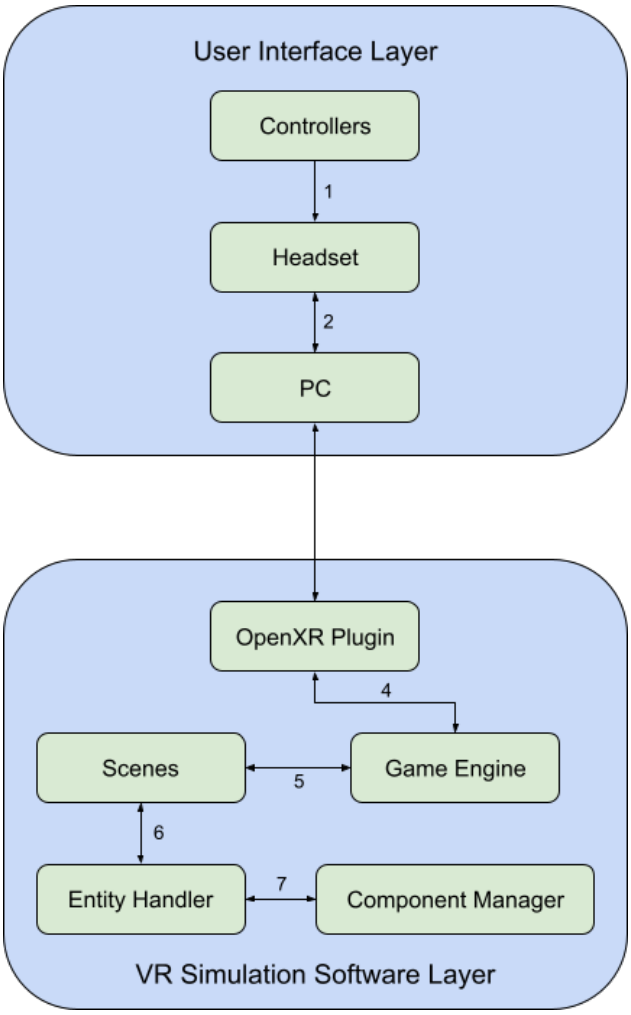

Figure 2: A simple data flow diagram

4 User Interface Layer Subsystems

The User Interface Layer accounts for all of the devices that the user will interact with. This includes the controllers, the headset, and the PC. This layer takes input from the user and forwards the data either directly to the VR Simulation Software Layer or to the APK built by that layer. This layer can also take in simulation updates and display that information to the user.

4.1 Controllers

As this is a VR project, input must include both button presses and user hand gestures. The Controllers will allow the user to interface with the VR scenarios using both of these methods. Additionally, the Controllers will also aid the Headset in inside-out tracking by producing IR signals that are accepted by the Headset to discern the direction the body of the user is facing. The project is currently being tested using Oculus Quest 2 controllers, but because the OpenXR plugin for Unity is being used in development, other VR setups should be supported.

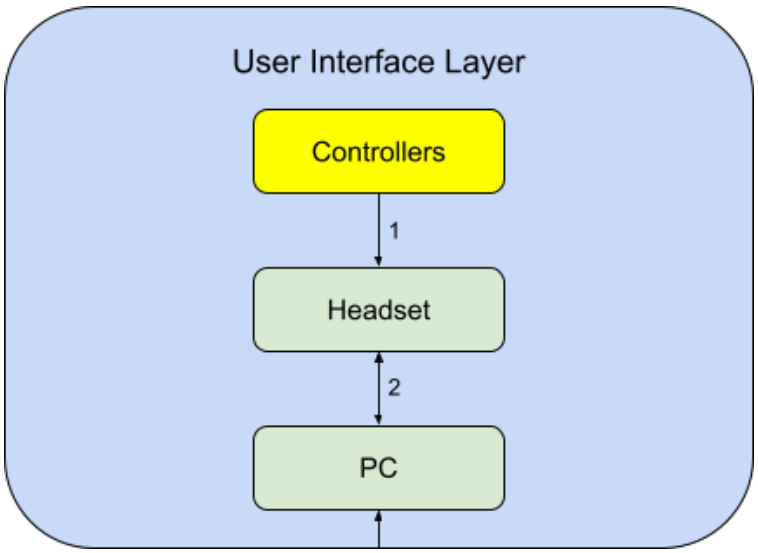

Figure 3: Controllers subsystem description diagram

4.1.1 Assumptions

We assume that the controllers' batteries are working or charged, that there are no software or hardware issues with the controllers, and that they can be detected by the headset. It is also assumed that two controllers, left and right, are used in order to operate the system.

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. The Controllers are also responsible for aiding the Headset in inside-out tracking using its IR tracking dots.

4.1.3 Subsystem Interfaces

Table 1: Controllers subsystem interfaces

ID	Description	Inputs	Outputs
#02	Headset Interface	IR Signals User Actions Hand Movements Button Presses	Visual Data

4.2 Headset

The headset is capable of operating as an independent console that can run programs without the use of additional external hardware, with the only exception being the controllers that come with it. As such, it is also capable of independently monitoring its own location and direction, with only minimal input from the user, using inside-out tracking. When running via the APK, user input from the controllers will be collected by the headset, which will then process, update, and display the effects of those actions on the game state to the user. The program software that will dictate these effects will be uploaded to the headset via the PC. When run via Unity, these things will be done directly by the VR Simulation Software Layer through the PC.

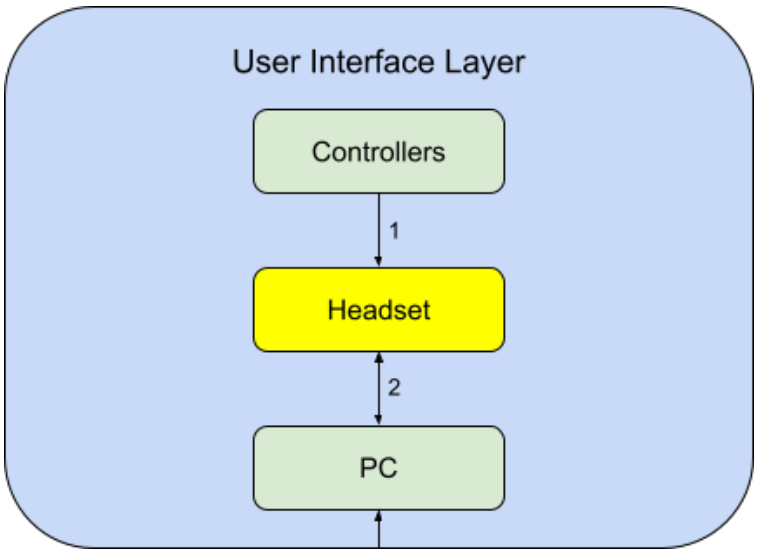

Figure 4: Headset subsystem description diagram

4.2.1 Assumptions

We assume that the headset will process input from the controllers correctly, as well as the effects of that input in the game state, and that there are no issues uploading the program software onto the headset via the PC if this method is used. We will also assume that the user has access to the the headset and that there are no mechanical or software issues with the headset.

4.2.2 Responsibilities

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

4.2.3 Subsystem Interfaces

Table 2: Headset subsystem interfaces

ID	Description	Inputs	Outputs
#01	Controller Interface	N/A	IR Signals User Actions Hand Movements Button Presses
#03	PC Interface (Unity)	User Actions Hand Movements Button Presses	Game Updates
#03	PC Interface (APK)	N/A	Program Software

4.3 PC

The PC can either send information between the headset and VR Simulation Software Layer or upload the program software onto the headset. It is also the hardware where we will create, modify, test, and build the software package.

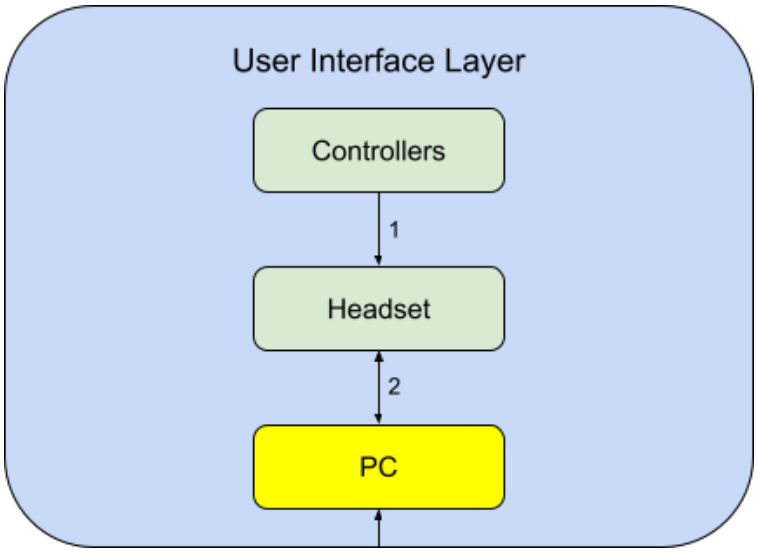

Figure 5: PC subsystem description diagram

4.3.1 Assumptions

The PC will contain hardware and software capable of running and hosting the VR application and VR equipment. If it is desired to use the simulation wirelessly, the PC is assumed to have access to a wireless

adapter.

4.3.2 Responsibilities

The PC is responsible for hosting the VR application and ensuring it has the necessary resources to operate during testing. The PC is also responsible for creating and building the program software on the Unity game engine. During the testing phase, the simulation should be able to run on both a wired and wireless connection. At present, there is only support for a wired connection since the PC is connected to the internet via Ethernet.

4.3.3 Subsystem Interfaces

Table 3: PC subsystem interfaces

ID	Description	Inputs	Outputs
#02	Headset Interface (Unity)	Game Updates	User Actions Hand Movements Button Presses
#02	Headset Interface (APK)	Program Software	Visual Data

5 VR Simulation Software Layer Subsystems

The VR Simulation Software layer is the application layer of the project. It manages scenes, assets, and game updates and builds the program software for upload onto the User Interface layer. This layer is also capable of taking in the user's actions and location, and outputting 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.

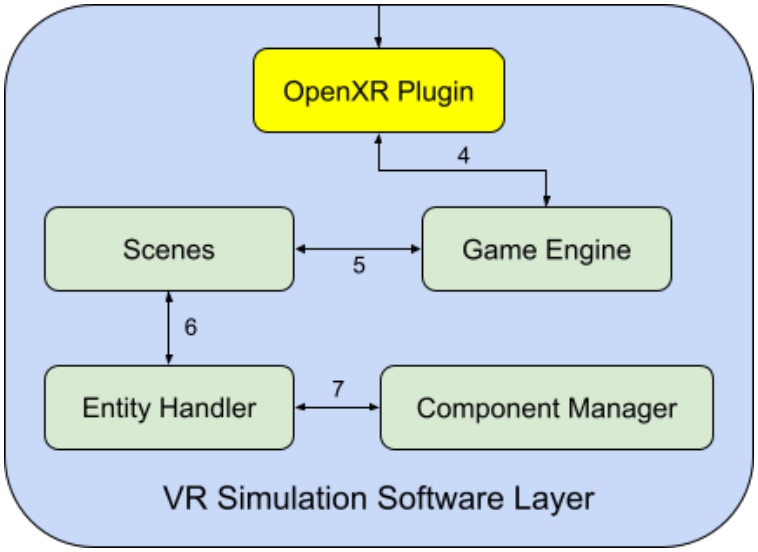

Figure 6: OpenXR subsystem description diagram

5.1.1 Assumptions

Both the Oculus Quest 2 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 4: OpenXR subsystem interfaces

ID	Description	Inputs	Outputs
#03	PC Interface	Player Movement Player Actions User Movement	Application Graphics Game Updates
#05	Game Engine Interface	Application Graphics Game Updates In-Game Movement	Player Movement Player Actions

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.

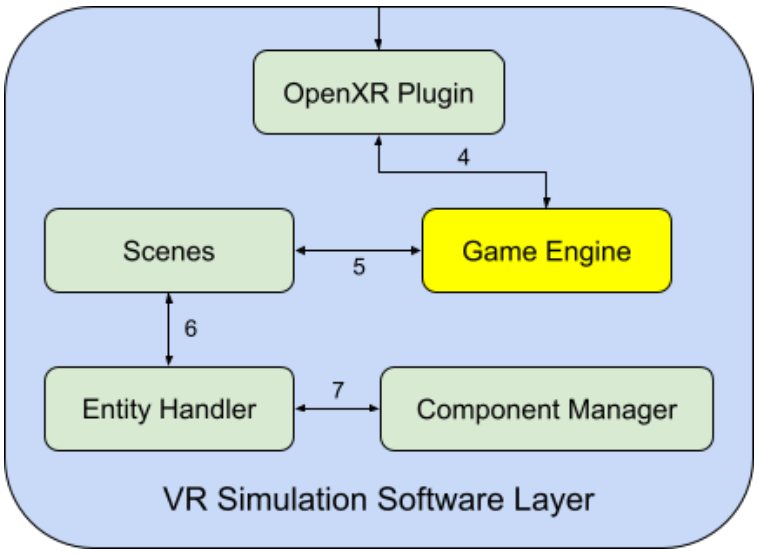

Figure 7: Game Engine subsystem description diagram

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 5: Game Engine subsystem interfaces

ID	Description	Inputs	Outputs
#04	OpenXR Plugin Interface	Player Movement Player Actions	Graphics Game Updates In-Game Movement
#06	Scenes Interface	In-Game Movement Game Updates	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].

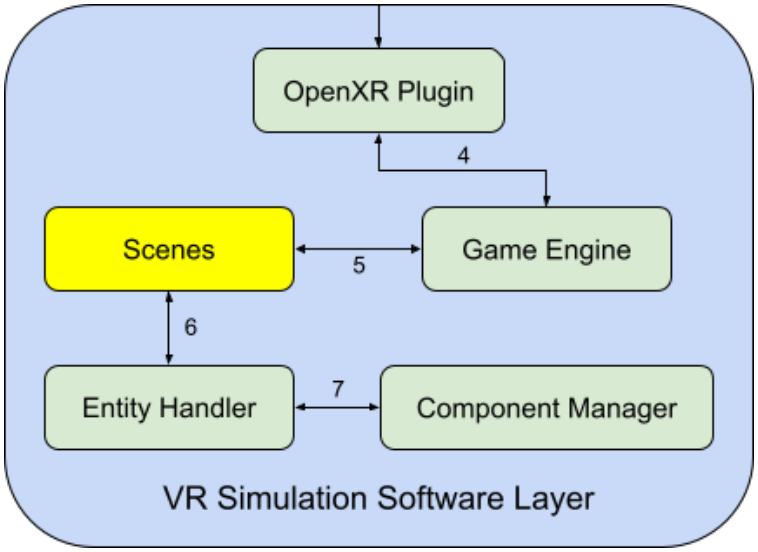

Figure 8: Scenes subsystem description diagram

5.3.1 Assumptions

The Scenes receive all 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 happen within 16.6 milliseconds of 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 6: Scenes subsystem interfaces

ID	Description	Inputs	Outputs
#05	Game Engine Interface	User Actions User Movement	Game Updates In-Game Movement
#07	Entity Handler Interface	Entity Information User Input Affecting Entities	Entities to be Updated

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 the Player, Patient, NPC, Camera, Listener, and General Objects. 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].

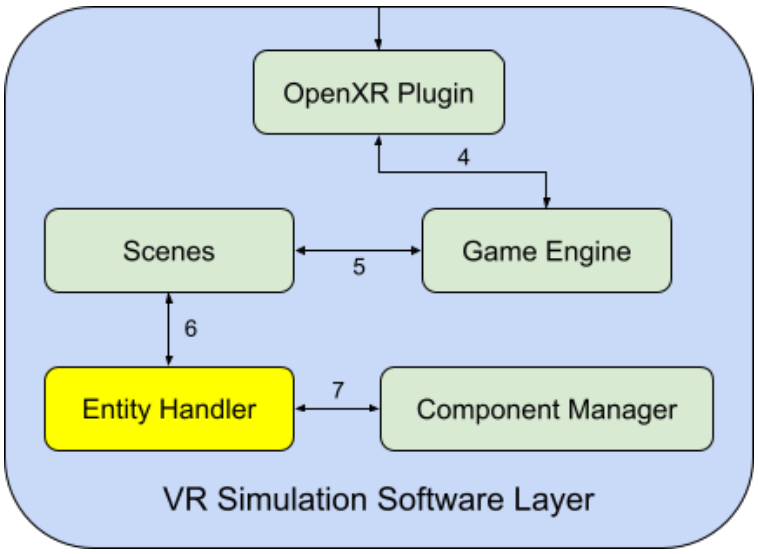

Figure 9: Entity Handler subsystem description diagram

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 7: Entity Handler subsystem interfaces

ID	Description	Inputs	Outputs
#06	Scenes Interface	Data on entities to be updated	Updated Entity Information User Input Affecting All Entities
#08	Component Manager Interface	Updated Player Data Updated Patient Data Updated Camera Data Updated Listener Data Updated General Objects Data	Current Player Data Current Patient Data Current Camera Data Current Listener Data Current General Objects Data

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].

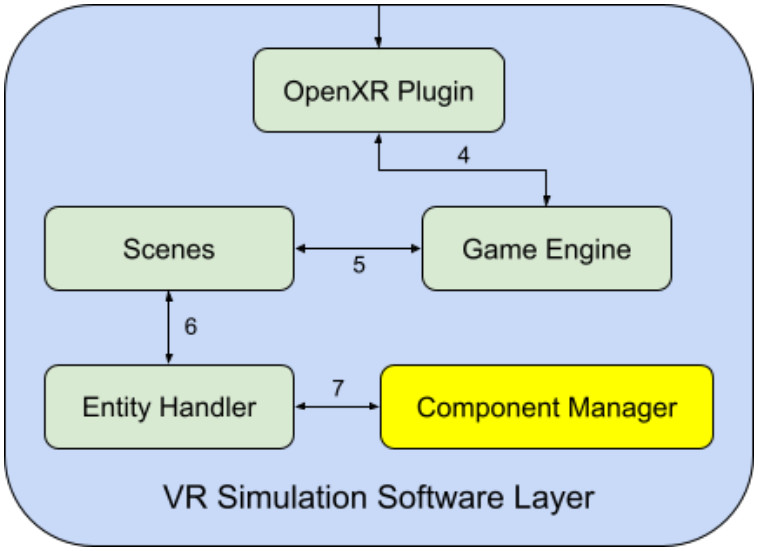

Figure 10: Component Manager subsystem description diagram

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].

5.5.3 SUBSYSTEM INTERFACES

Table 8: Component Manager subsystem interfaces

ID	Description	Inputs	Outputs
#07	Entity Handler Interface	Request from the Entity Component for a Component's Information Data from Smaller Subsystems within Component Manager	Requested Data

REFERENCES

[1] VRx. VR Nursing CSE Senior Design. https://blog.uta.edu/cseseniordesign/2021/05/04/vrx-3/, 2021. Accessed: 2022-04-10.