Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Summer 2022

NurSim
VR Palliative Care

Emelyne Hoang
Jackson Park
Jorge Rodriguez
Rafel Tsige

Revision History

Revision	Date	Author(s)	Description
0.1	06.23.2022	EH	document creation
0.2	06.23.2022	EH, JP, JR, RT	initial draft
1.0	06.27.2022	EH, JP, JR, RT	finalized version 1
1.1	07.13.2022	EH	updated text to reflect current testing equipment
1.2	07.15.2022	EH	updated figures to reflect current testing equipment
1.3	08.10.2022	EH	corrected inconsistent parts of text
2.0	08.14.2022	EH, JP, JR, RT	finalized version 2

CONTENTS

List of Figures

List of Tables

1 INTRODUCTION

The palliative care VR simulation that will be produced will allow the user to simulate one of four scenarios that are not taught during the regular course load for nursing students. These scenarios are delivering a terminal diagnosis to a patient and their family, accessing a patient's home for hospice release, informing the patient and their caregiver of the proper way to sterilize equipment and maintain good hygiene, and finally how to deal with the patient's body following their death. The Palliative Care VR system will use Virtual Reality to allow the user to better interact with their surroundings and gain more from the learning process than the traditional computer application. The application is intended to be cross-platform through the use of the OpenXR plugin, and will currently be tested on the Oculus Quest 2.

The specific software requirements for each scenario were defined by the sponsor Jennifer Roye and former sponsor RaeAnna Jeffers. The current software requirements are described more extensively in the Software Requirements Specification.

The system overview of our project consists of two high level layers. These layers include 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. The VR Simulation 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 either packages the program into an APK to be uploaded onto the headset or runs directly via Unity and sends the updated environment back to the user through the PC.

These main two components and their subsystem layers are described more extensively in the Architectural Design Specification.

The purpose of this Detailed Design Specification is to describe in depth how the previously defined architecture and software requirements will be implemented. Any programming languages, extra packages or resources will be listed and will be used as blueprint to follow continuously as the product is developed.

2 SYSTEM OVERVIEW

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.

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.

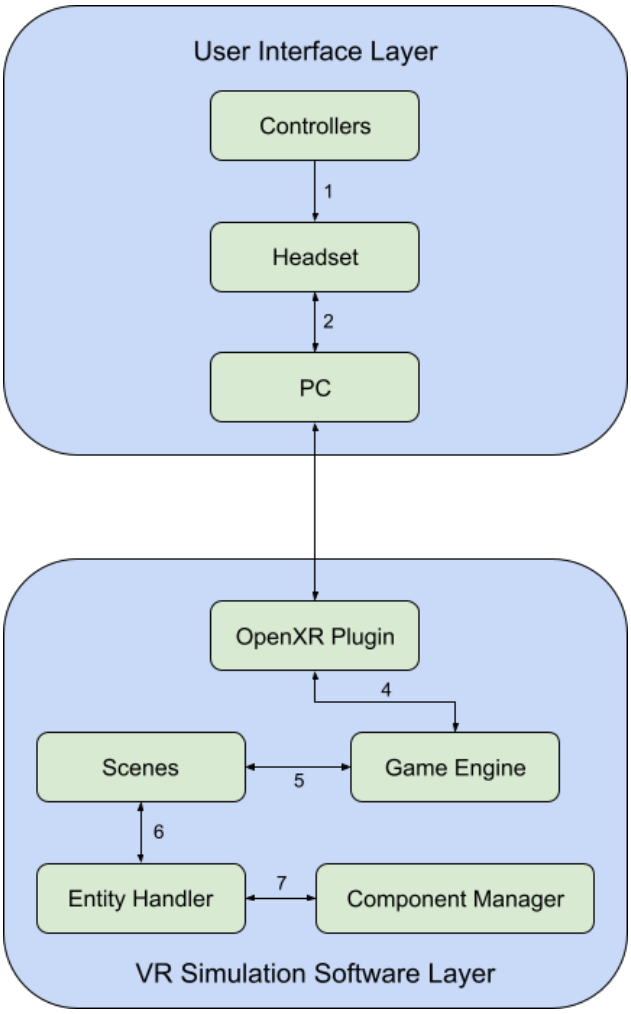

Figure 1: System Architecture

3 User Interface Layer Subsystems

3.1 Layer Hardware

The User Interface Layer is specifically designed to encompass all hardware aspects that the user uses to interface with the software. As a basic overview, on the Oculus Quest 2, the Headset uses internal cameras and sensors, as well as sensors on the Controllers, to perform markerless tracking using simultaneous localization and mapping (SLAM). This creates a real-time 3D map of the user's environment without the use external markers that can simultaneously detect the user's position, the direction the user is facing, and any potential obstacles in the environment. This data is then either processed directly by the simulation APK uploaded to the Headset or sent from the Headset to the PC for processing by Unity along with other input data such as any Controller button interaction, Headset direction, etc. The PC will, in response, provide visual and audio stimulus that are determined by the VR Simulation Software layer.

3.2 Layer Operating System

The Controllers and Headset all run on proprietary firmware and can be run independently or interface to the PC using OpenXR. Fundamentally, OpenXR functions much like an operating system for Unity to interface between the hardware components without customizing code for each specific setup.

3.3 Layer Software Dependencies

In order for our software to be hardware independent, as far as VR equipment goes, the hardware needs to be able to interface with the program without custom code. For this reason we use OpenXR that acts as an interface between the VR hardware and our software. For this reason, this entire layer requires the use of OpenXR to interface correctly.

3.4 Subsystem 1: Controllers

The Controllers are two pieces of hardware provided by the VR Headset manufacturer. They are responsible for tracking the user's hand movements and location and allows the user to interact with the VR system using physical buttons, triggers, and thumbsticks.

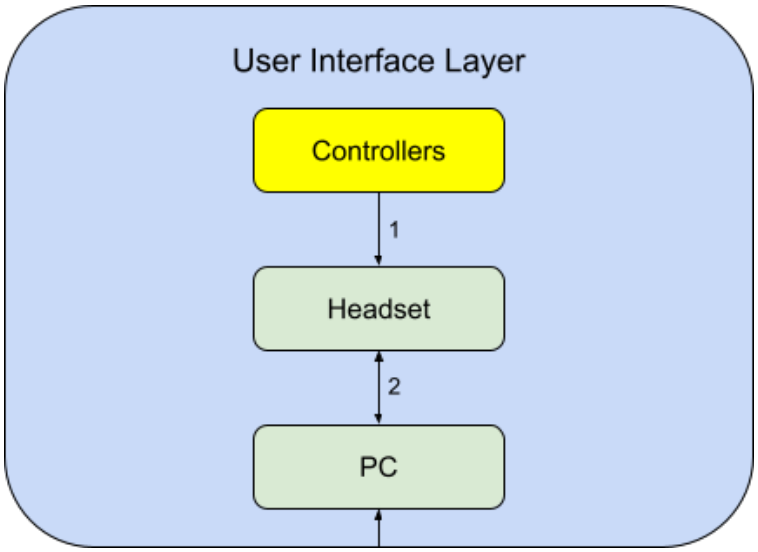

Figure 2: User Interface Layer: Controllers

3.4.1 Subsystem Hardware

The Controllers feature multiple hardware components. A menu button is provided below the thumbstick and main buttons on the left controller. The Oculus button is located in the same position on the right controller and returns the user to Oculus home. The thumbsticks are located on the outer edge of the top surface of both controllers. It may be used to move the user in VR. Two buttons are preset next to the thumbstick on each controller. A status light is located on the user-facing part of the ring on the top of the controller. This ring is also where multiple IR tracking dots are located. On the handle of the Controllers are an index finger trigger and a hand trigger. The functionality of the buttons can be set by software, that is, the functionality of these buttons depends on what software is currently being used.

The Controllers are paired with the Headset, and data is transmitted from the Controllers to the Headset. The Controllers are battery powered and it is up to the user to replace the batteries should they run out of power.

3.4.2 Subsystem Software Dependencies

The Controllers require the Oculus app as well as a functioning Headset to run. In order to interface with the Controllers in this simulation, the OpenXR plugin for Unity is used.

3.4.3 Subsystem Data Structures

Interaction is done using the Unity input system and OpenXR interaction toolkit. Interactable objects will inherit properties from OpenXR Interactable classes. For example, the XRGrabInteractable class, which is an interface for grabbing interactable objects, will allow the state of the object to be updated based on input. The XR Ray Interactor will be used heavily, for objects that are interacted with using a raycaster. It is important to note that interaction and input through OpenXR is done through an action-based approach, not a device-based approach

3.4.4 Subsystem Data Processing

The Controller will allow interaction with objects in the VR Palliative Care scenarios. Hovering over certain objects will allow a more detailed and large view. For example, hovering over the vitals in Scenario 1 will enlarge the vitals so they can be properly read. Using the grip button, objects can be interacted with. The type of interaction depends on the purpose of the object. Interacting with the sink will toggle the water flow. Interacting with the paper towels will give the user a paper towel. Interacting with hazardous items will remove or change them. Hover and select mechanics will be used for dialogue, so that the user can select a dialogue choice during conversations. Most object interactions will change the state of the object interacted on and will progress the scenario, as well as be recorded by the reporting system. For example, if the user interacts with objects in the proper order to wash and dry their hands, they will get points for hand washing and it will be recorded on the report.

3.5 Subsystem 2: Headset

The Headset is a the primary piece of hardware for a VR system. The Headset is responsible for positional and directional tracking, and provides the user with a VR view of the scene they are in. The VR Headset has two screens for each eye, simulating how a human views a scene in real life, and two speakers on either side of the headset to provide appropriate audio queues.

3.5.1 Subsystem Hardware

The Headset contains multiple cameras, an accelerometer, gyroscope, IMU (Inertial Measurement Unit), laser position sensor, magnetometer, microphones, lenses, speakers, and volume and power button controls. The Headset uses the cameras in conjunction with the accelerometer, gyroscope, IMU, magnetometer, and IR signals from the Controllers to perform markerless tracking using simultaneous lo-

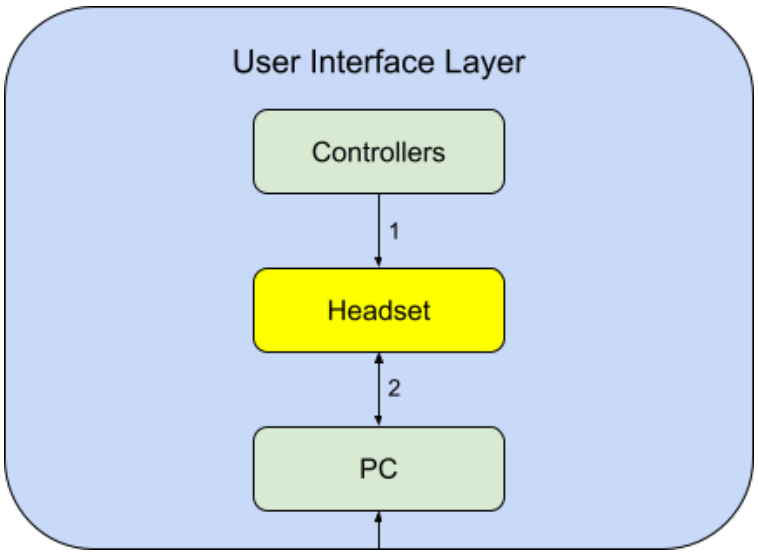

Figure 3: User Interface Layer: Headset

calization and mapping (SLAM). The lenses can be manually pushed inward and outward to adjust positioning. The Headset can be connect to the PC via an Oculus Link cable.

3.5.2 Subsystem Software Dependencies

The Headset will require the Oculus app to be recognized by the PC and used for VR simulations. Unity and the OpenXR plugin will be required to interface with the Headset. All systems in the simulation will need to be developed specifically for VR. This includes placing dialogue in world-space, designing objects so that they are interactable in VR, and designing scenes so that they are easy to traverse in VR.

3.6 Subsystem 3: PC

The PC is a the combination of hardware necessary to operate the VR system. The PC is the "central hub" for the VR system, and is responsible for computation necessary to execute the VR software.

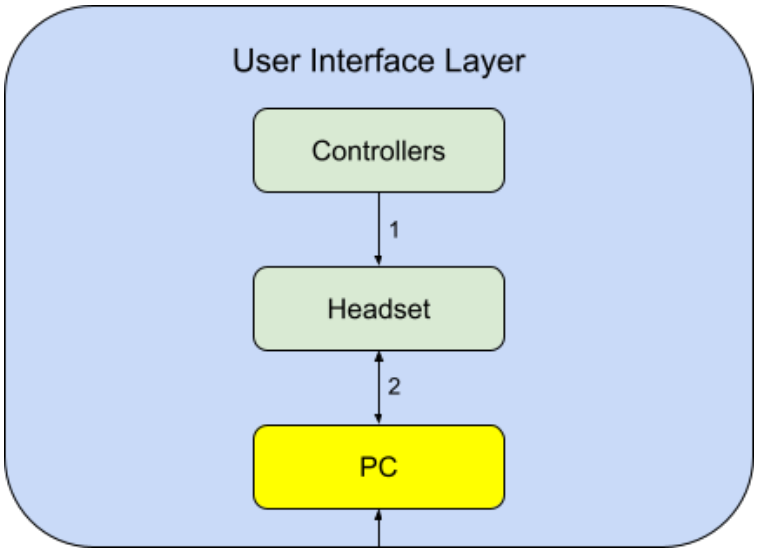

Figure 4: User Interface Layer: PC

3.6.1 Subsystem Hardware

The PC includes several components, including a graphics card, CPU, motherboard, RAM, storage medium, and connectivity through USB. The specifications of the PC can be variable, but should meet the Oculus Quest 2 minimum requirements:

Processor	Intel i5-4590 / AMD Ryzen 5 1500X or better
Memory	8 GB RAM
Graphics	NVIDIA GeForce GTX 1070 or AMD 500 Series or higher
Additional Notes	1x USB port

Table 2: Oculus Quest 2 minimum requirements

In addition, if the user desires to connect to the Headset via Oculus Air Link, both equipment must be on the same network. Lastly, proper storage space will be necessary for the Palliative Care VR system. 50 GB of free space (space not used by the OS or applications) is required.

3.6.2 Subsystem Operating System

Windows 10 is the required OS for the system. While Oculus does support operating systems as old as Windows 7, those will not be tested with this system and thus Windows 10 will be required.

3.6.3 Subsystem Software Dependencies

The Oculus app, Oculus Developer Hub, Unity, and the OpenXR plugin are all required. DirectX libraries will need to be installed, and will be installed automatically by Windows on the first run if not already installed.

4 VR Simulation Software Layer Subsystems

4.1 Layer Hardware

Because the project requires the use of VR technology our software must be used in conjunction with VR hardware. This hardware includes a headset meant for conveying visual and audio stimulus to the user along with two hand controllers for user input into the simulation. For OpenXR to know the position of the user, the Headset works in conjunction with the Controllers to perform inside-out tracking, described in Section 3.5.1. Outside of this, the VR headset also needs a compatible and powerful GPU that is capable of rendering and building the simulation.

4.2 Layer Operating System

Because the software we are making will be built in Unity, we are capable of developing this software for many different operating systems, however the use of VR limits our build platforms to PC platforms such as Linux, Mac OS, and Windows. The build we create will make use of each operating system's libraries and tools.

4.3 Layer Software Dependencies

Our software is dependent on the Unity game libraries and engine along with the OpenXR VR libraries.

4.4 Subsystem 1: OpenXR Plugin

The OpenXR plugin is a subsystem that creates an interfacing layer between the VR hardware and the actual code implemented in Unity. This layer provides the same functionality between different headsets without having to customize the code for each set. In this way, we can abstract from the hardware and simply communicate with OpenXR.

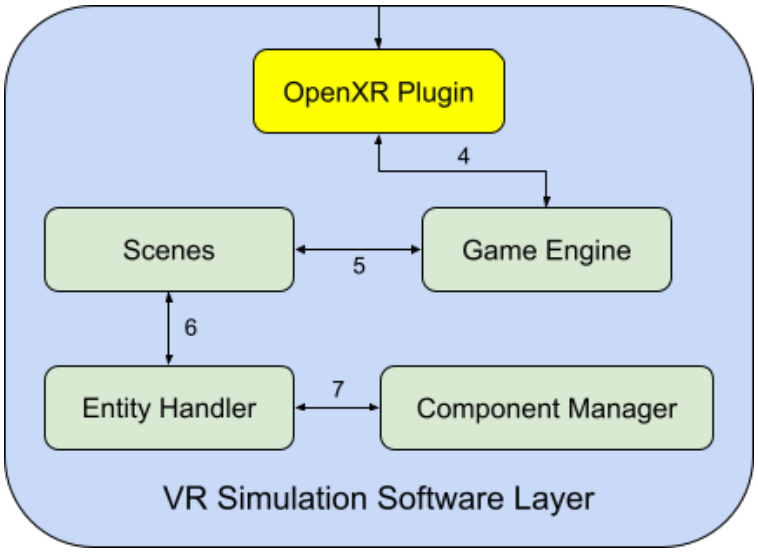

Figure 5: VR Simulation Software Layer: OpenXR Plugin

4.4.1 Subsystem Hardware

OpenXR is designed to interface between multiple types of hardware including the Oculus. As it is meant to interface between hardware components it requires VR hardware to be useful.

4.4.2 Subsystem Software Dependencies

OpenXR requires that the VR hardware have the proper updated firmware and drivers to interface properly.

4.4.3 Subsystem Programming Languages

Though the goal of this project is not to create code for OpenXR, OpenXR is written in C.

4.4.4 Subsystem Data Structures

OpenXR has many different structures that are used to control and customize the software's experience for the programmer. One set of structures that are particularly interesting are CreateAction, CreateActionSet, and CreateActionSpace. These three structures are used to tell OpenXR what to do when in a VR session. Each of these help create a set of actions that the OpenXR plugin must execute and are important to adding features to each scene.

4.4.5 Subsystem Data Processing

When using OpenXR the programming team must first start the Application, then create the OpenXR instance. Once these two are setup, programmers can implement VR sessions where different actions can be implemented by OpenXR. When a session is over, the instance then needs to be freed and the application can end. This is the basic implementation of the OpenXR plugin.

4.5 Subsystem 2: Game Engine

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

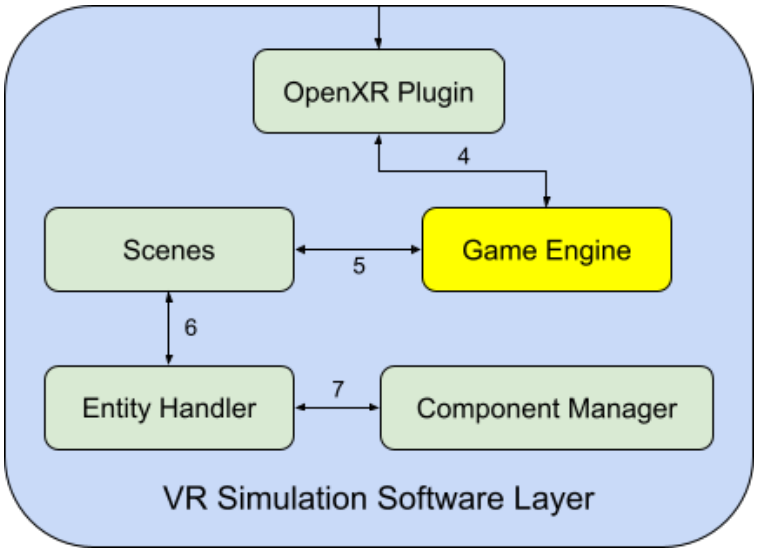

Figure 6: VR Simulation Software Layer: Game Engine

4.5.1 Subsystem Software Dependencies

Unity Game Engine requires the full set of libraries included with each version of Unity to run properly.

4.5.2 Subsystem Programming Languages

The Unity Game Engine uses C# for its programming language.

4.6 Subsystem 3: Scenes

Scenes are a mechanism in Unity that helps programmers separate levels or spaces within a game universe. Each scene records what assets and objects exist within the scene so that the same level, in the same begin state, can be loaded on demand at a later time.

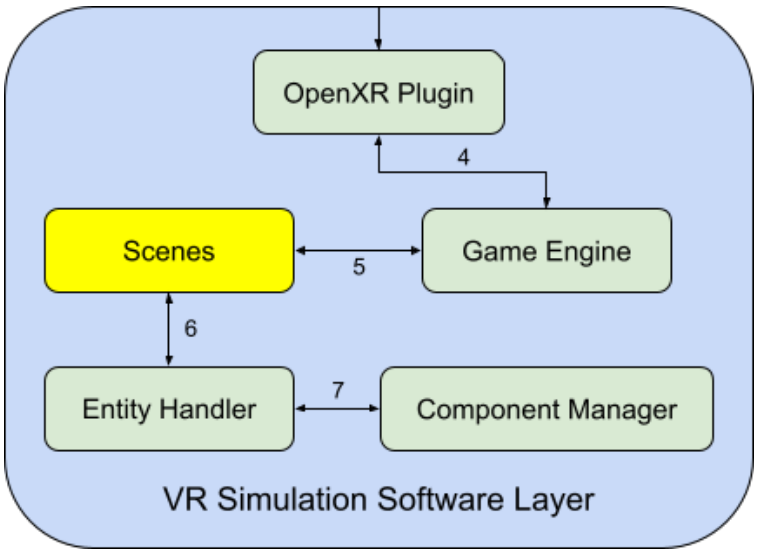

Figure 7: VR Simulation Software Layer: Scenes

4.6.1 Subsystem Software Dependencies

Assets are a heavy dependency for all scenes. If any asset is missing or in an incorrect location, the scene will not be able to render correctly. Assets may be the 3D models of game objects, certain textures, and other prefabs from the Unity store.

Scene 3 will need the same assets as Scene 2 which will be the house setting and the family members. In addition to Scene 2 assets, Scene 3 will need the at-home nurse equipment such as the suction mechanism and blood pressure reader. Scene 4 will need all assets from Scene 2.

4.6.2 Subsystem Data Structures

In a high level sense, scenes hold the data for all assets that are used in a 'Scene'. Scenes help game developers separate between each 'set piece' of play area and give the game engine the set of assets that will need to be loaded and rendered.

4.7 Subsystem 4: Entity Handler

The Entity Handler is a mechanism inside Unity whose main task is determining how entities interact with one another in the game space. This include determining when they collide, when certain objects set off triggers, etc. The Entity Handler receives all objects that need to be loaded from scenes.

4.7.1 Subsystem Software Dependencies

Outside of the necessary Unity API, the Entity Handler requires the presence of entities and objects within the game scene to provide any real useful functionality.

4.7.2 Subsystem Data Processing

The Entity Handler keeps track of each object within the game world to help the game engine know how and when different objects interact and what the significance of that interaction is.

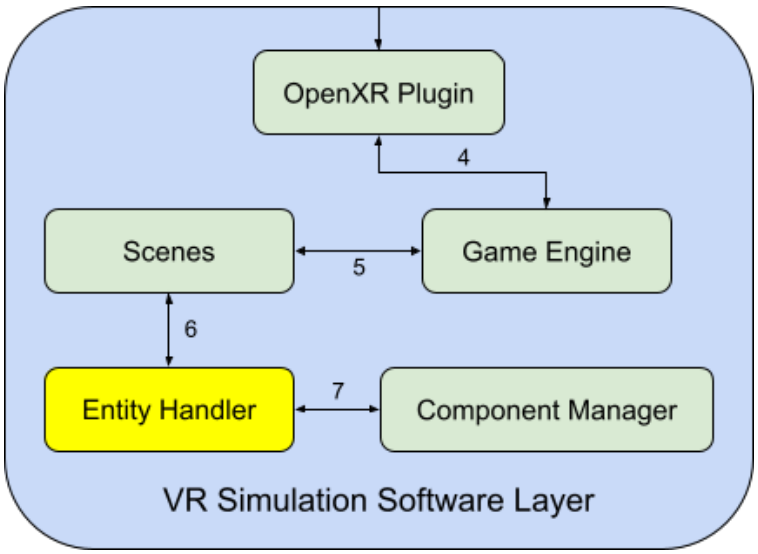

Figure 8: VR Simulation Software Layer: Entity Handler

4.8 Subsystem 5: Component Manager

The Component Manager is a subsystem within Unity that manages the usage of components for each object within the game scene. Each object within the game scene has at least one component, that being the transform component, and are able to add more functionality and define collision behavior using the component manager.

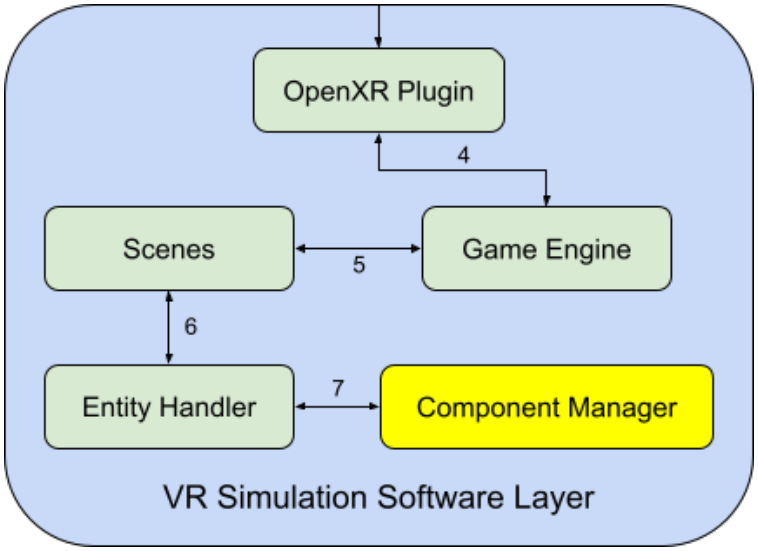

Figure 9: VR Simulation Software Layer: Component Manager

4.8.1 Subsystem Software Dependencies

The Component Manager requires Unity Libraries and objects to exist as each object has at least one component to manage.

4.8.2 Subsystem Data Processing

The Component Manager manages the components that are added to an Object or Prefab that help to define object behavior. By using the Component Manager we can add functionality such playing audio when a certain trigger occurs or defining a Rigidbody for an object.