Emergent Architecture Design

Computer Games Contextproject 2015-2016
Course TI2806, Delft University of Technology

Group PixelPerfect

April 28, 2016

Supervisor
Dr. ir. Rafael Bidarra

Teaching Assistents
Sander van den Oever
Jurgen van Schagen

Team

David Alderliesten	4368703	*J.W.D.Alderliesten@student.tudelft.nl*
Floris Doolaard	4362748	*F.P.Doolaard@student.tudelft.nl*
Dmitry Malarev	4345274	*D.R.Malarev@student.tudelft.nl*
Jesse Tilro	4368142	*J.Tilro@student.tudelft.nl*
Wouter Zirkzee	4398858	*W.J.Zirkzee@student.tudelft.nl*

Table of Contents

Introduction

Pixelperfect was assembled during the fourth quarter of the academic year of 2015-2016 for the Context Project course of the Computer Science bachelor program at the Delft University of Technology. The team consists of five members and was assigned to design, develop, and release a virtual reality game for the Oculus Rift platform.

The project requires an array of specialized hardware devices in order to be played, including a virtual reality headset called the "Oculus Rift" which is at the heart of the software product. The connectivity between the other participants of the game on their phones or devices and the dependability of the game on the motion tracking hardware makes for a tough challenge which will be solved using [TECHNOLOGY]. The continuous interdependability is outlined in this document and was at the core focus of the design for the game.

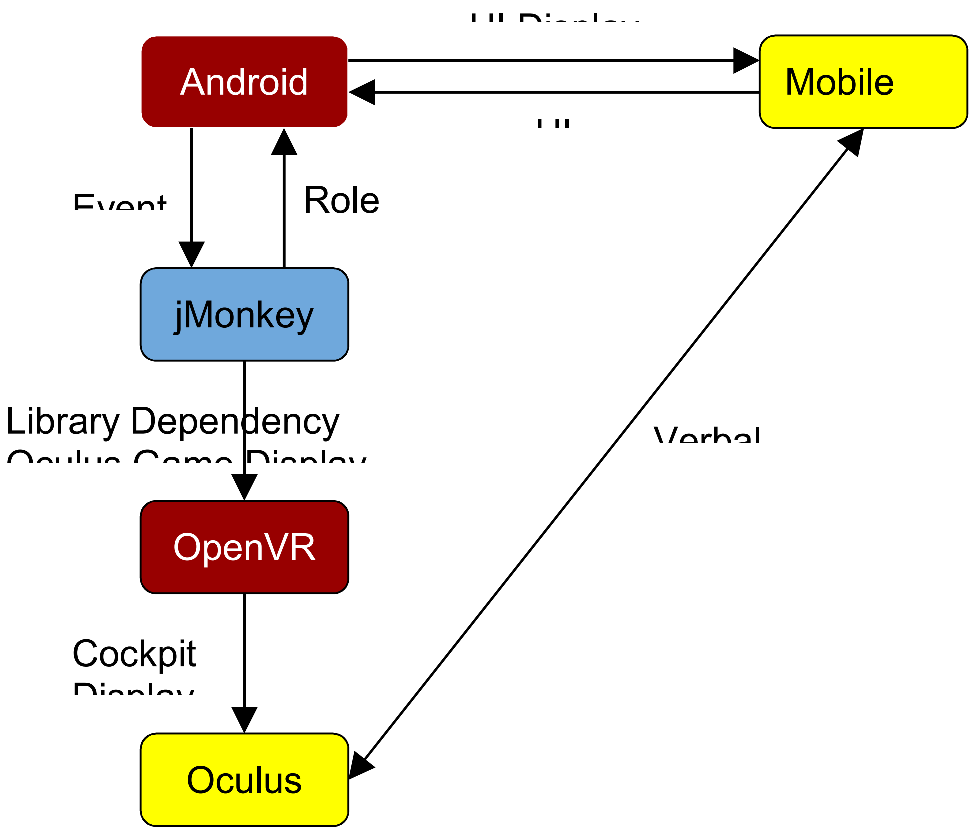
The focus of the design of the system architecture was to find a balance between performance, reliability, and manageability. The performance of the system was crucial due to the interdependability between the Oculus Rift player and all the mobile players, who must be able to reliably communicate during time-restricted moments of the game. The reliability refers to the reliability of the connections and interactions between the players, which must be stable to allow for a simple implementation and to prevent frustration during the act of connecting of the players. The maintainability is of the essence for the maintenance of the project, which must be advanced enough to offer a rich experience to the player, yet maintainable for the programmers to prevent bloat and clutter.

This document outlines the decision made for the software project with regards to the subsystem decomposition, the mapping of hardware and software, the method utilized for the storage of data, and how concurrency in systems is handled. A glossary can be found at the end of the document outlining certain important terms for the project.

Architectural Views

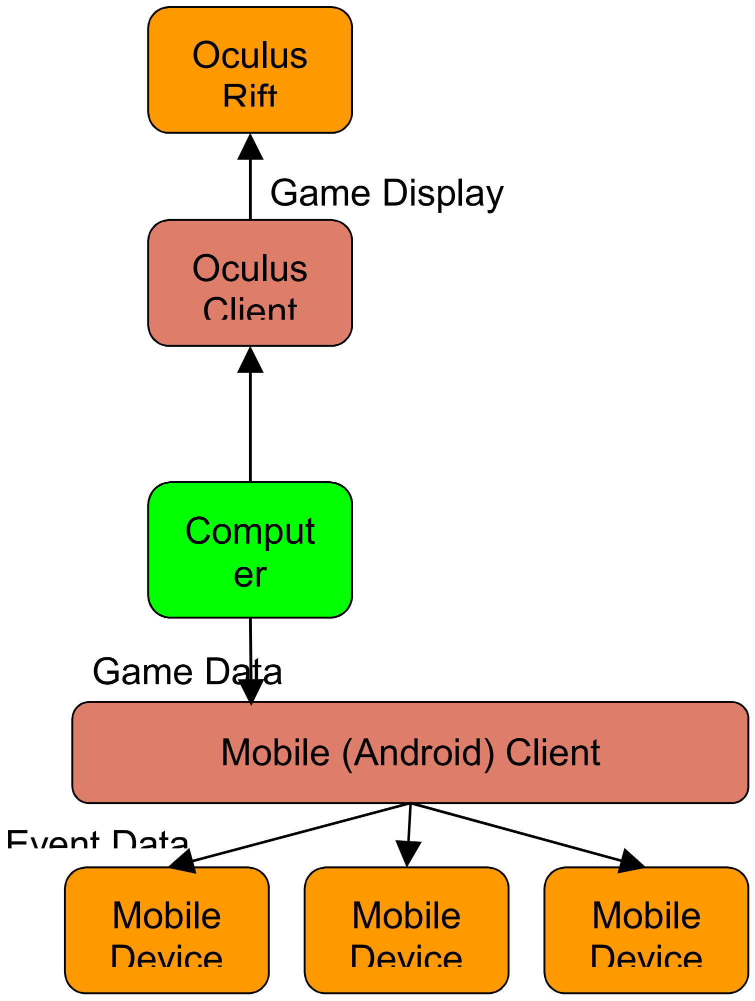
Subsystem Decomposition

The game runs on JMonkeyEngine, and relies on the Android mobile OS libraries and the OpenVR integration library to display the relevant game data and interactions on the phone and Oculus Rift, respectively. Verbal communication occurs between the players of the game, regardless of their hardware. In the subsystem decomposition diagram, integration systems are shown in red, the engine is shown in blue, and players and their hardware is shown as yellow.



Hardware & Software Mapping

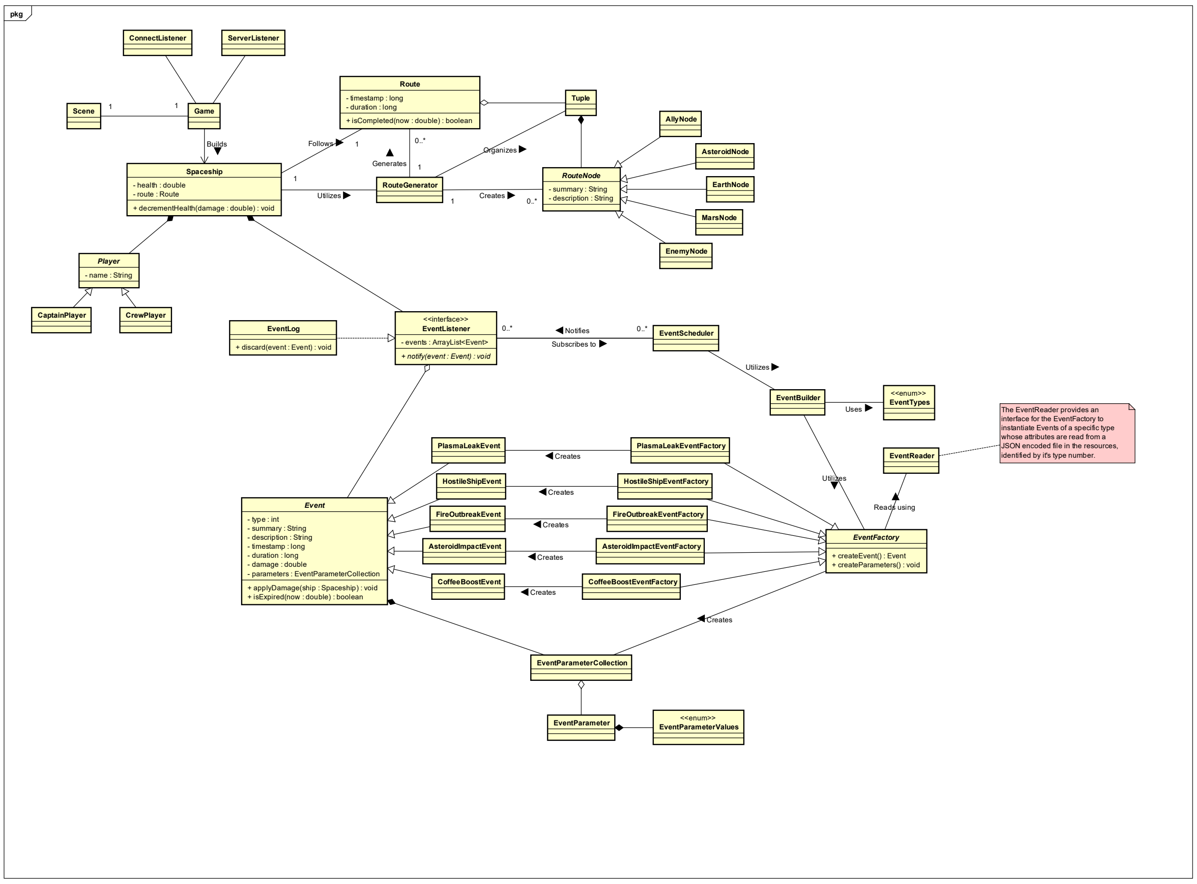
The peripherals required for the project are the Oculus Rift virtual reality system, the mobile devices used by the other players, and a central computer that acts as the processor for the Oculus Rift and the central access point for the other players. The Oculus client handles the game data and displays the 3D cockpit interface on the Oculus Rift. The Mobile Client takes selected game data (active events, game score, ship health) and displays the score and health on the UI, and allows the players to perform actions associated to their chosen role in order to complete the events. The Android OS is responsible for the engine to phone integration, while the OpenVR library is responsible for the engine to VR integration.



As can be seen, the Oculus Rift player's device acts as a type of server for the game session, and the mobile devices act as clients which connect to the server through the use of a LAN. Connection is done by having the mobile clients connect to the server using the local IP address of the server.

UML System Design

The project contains the following interactions between the classes and their subsystems.



Persistent & Session Data Management

The game includes a file system to track local high scores that will be updated and synchronized whenever the game client is active. This will be done by using a file that stores a finite amount of scores, aiming for a maximum of ten. The file will be then read every time the game is launched and updated whenever a game completes if the score attained in the most recent game is higher than any of the ten stored scores.

If a score has to be written in the file, the program will check at what place does it have to be in the file and write it, shifting the lower scores one spot and removing the score of the last place.

All the session data is handled by the client, which is defined as the system that is run on the central computer and to which the Oculus Rift player is connected. The system acts as a central access point which sends data to the phones at regular intervals, and checks for updates from the mobile players during these intervals.

Concurrency

Most of the mini games will be solved on an individual basis, requiring communication between all the players but having only a single player completing the specific event. Certain mini games will require some team effort to complete, such as the fire event, which can be performed by any player in any role, and in which coordination will bring the event to a more timely (successful) conclusion. Concurrency control ensures that all players are able to perform any action related to any mini-game at any time, but where multiple completions to incompatible mini-games results in ship damage and perhaps even game loss.

Concurrency control also exists within the game engine and handling devices to ensure that the multiplayer/networking, graphical updates, game logic, and game audio can all be updated and displayed/shown/function as required without interruptions occurring to any of them due to a lack of suitable concurrency.

A final location in which concurrency control exists is within the synchronization of the oculus client and the mobile clients. Sockets within the operating system are triggered by interrupts from the mobile client, which allow the game to both send and receive relevant game data in order to remain synchronized with the master client.