Architecture Design - HiVR

Adriaan de Vos - addevos - 4422643
Boris Schrijver - brschrijver - 4315332
Carlos Brunal - cbrunal - 4002725
Leon Hoek - ljhoek - 4021606
Wim de With - wdewith - 4295277

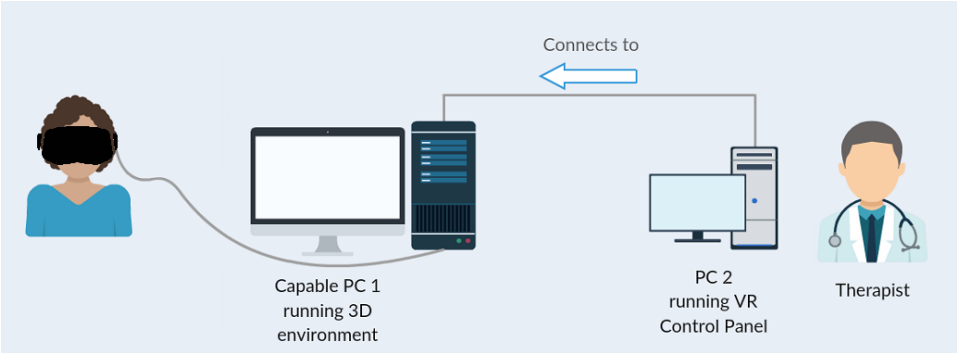
June 17, 2016

Contents

1 Introduction

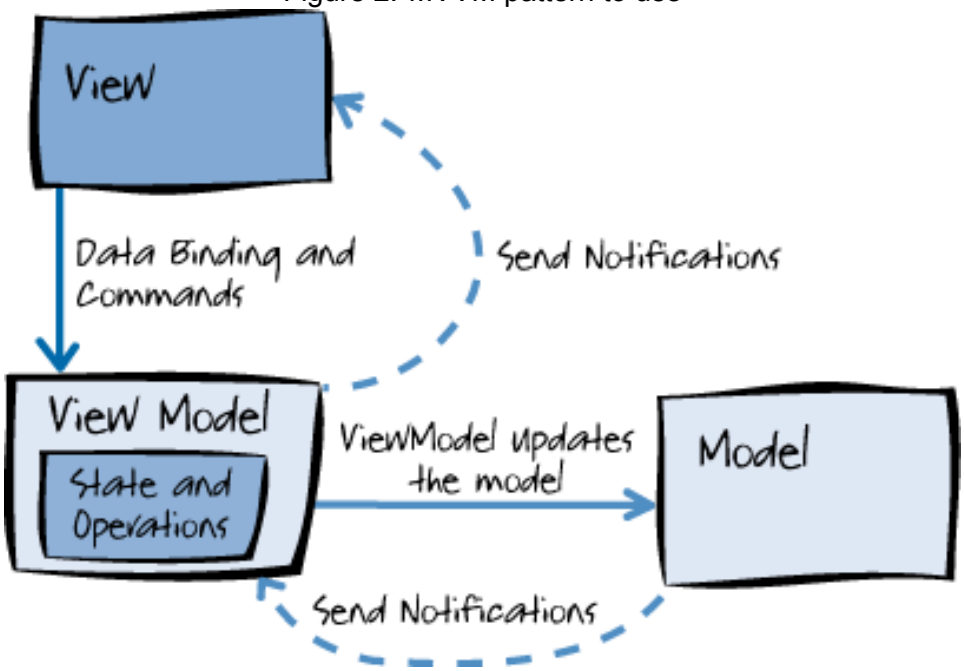
This document provides an overview of the system which will be built during the context project Health Informatics for CleVR.
The software is targeted towards medical specialists that wish to use the CleVR therapy software suite. The main focus is to make an interface as user friendly as possible without limiting treatment options. The interface will provide an overview of where the patient is situated in the VR world and also provide many tools that can aid the therapy sessions to the specialist's liking. figure 1 and figure 2 give a better overview of the architecture of our software.

Figure 1: Interface setup



The specialist will be able to both see exactly what the patient is seeing via computer number 1 as well as have an overview of the VR world drawn on a map on computer number 2, in order to see what falls outside of the patient´s field of view.

Figure 2: MVVM pattern to use



Representation of the MVVM pattern as planned for the project.

The architecture of the system will be explained in the form of high level components, sub-components and sub-systems. The structure of the implementation will be explained afterwards.

1.1 Design goals

We maintained three design goals throughout the project: ”Performance”, ”Ease of use” and ”Extensibility”. They’ll be explained in each subsection respectively.

1.1.1 Performance

The existing technology runs at an frame rate of 90 frames per second. Which means each frame has just over 11ms to render, of which we have at most 1 ms available for our code. We, as the development team, should limit the time needed for each frame to update the map/world as much as possible to keep the program running smoothly. A hard requirement is that we shouldn’t use more then 1ms of processing time each frame. This is of

vital importance to the treatment of the patients because any delay in performance may break immersion and cause nausea.

To meet the requirement set above we implemented a tracking system to keep references to all relevant models. By doing this we don't have to search through the whole Unity scene each time we want an update. We also don't extract information each frame, instead we use the fixedUpdate() call to fetch information 30 times a second. That way we are not bound by the frame rate of the VR-device. The information is afterwards handled by an asynchronous network processor. Another preference stated by CleVR is the use of a Canvas class in WPF, since without this the layout manager may not be able handle drawing many objects.

1.1.2 Ease of use

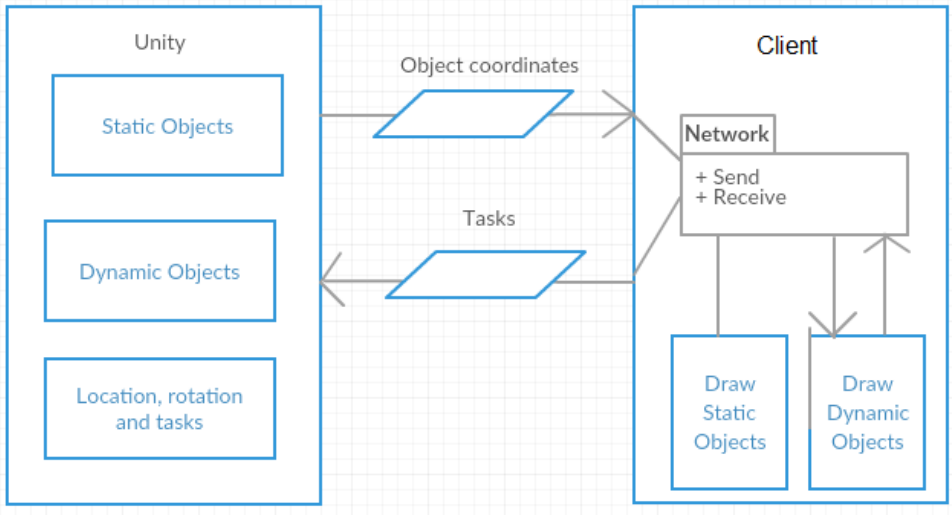
The interactive map should require little maintenance from the therapist during treatment. The focus of the therapist while giving treatment should be on the patient, not troubleshooting with the tools. It is important for the therapist to quickly see up-to-date information on the location of objects and actors in the Unity world, as well as the moods of actors, the field of vision of the patient and more, to quickly determine what triggers the patient. It should also be quick and simple for the therapist to manipulate the environment according to trigger the patient.

1.1.3 Extensibility

The software is to be implemented with future growth in consideration. The representatives of CleVR expressed the need for the software to be free of licensed libraries and also that it may be upgrade-able in the future. For example: the use of static bitmap images is not preferred, we are to use vector images instead.

In order to handle objects from Unity we implemented a control system in Visual Studio that reads both the coordinates and the scale of the objects in Unity. This system uses a list of data-templates for each of the relevant objects in Unity. This means for each object in Unity there exists a Model class and a viewModel class in the client. A class to be able to draw it on the canvas, and a control class to implement its behavior or procedures. Please look at the following figure 3.

Figure 3: How the canvas draws objects

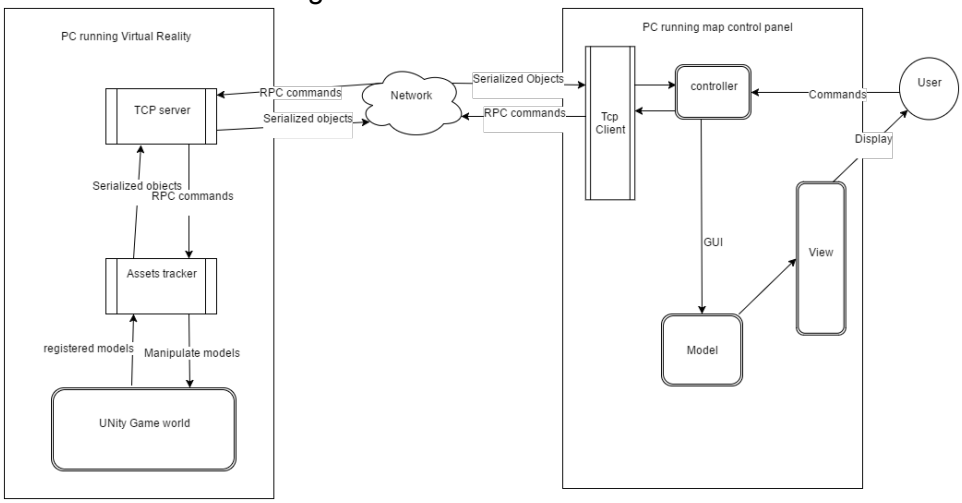


The clients sets map-coordinates and scaling of the received objects initially, then for each dynamic object keeps updating their location on retrieval. The client contains the same database of objects as the Unity server and it scales everything to a convenient size for displaying. Tasks are sent to Unity from the client, which in return sends a reaction containing coordinates and other important meta-data from unity.

2 Architecture

This section discusses the architecture of the system. We will briefly describe what each subsystem does, what the inter-dependencies are and how they work together.

Figure 4: Planned Architecture



A more in-dept view of the design we have implemented.

2.1 Subsystem decomposition

We are planning to build 3 components:

- The graphical interface itself including, layout of the map and inside the static objects like cars, buildings, benches and dynamic objects like actors and the patient. Two examples:

Figure 5: GUI map

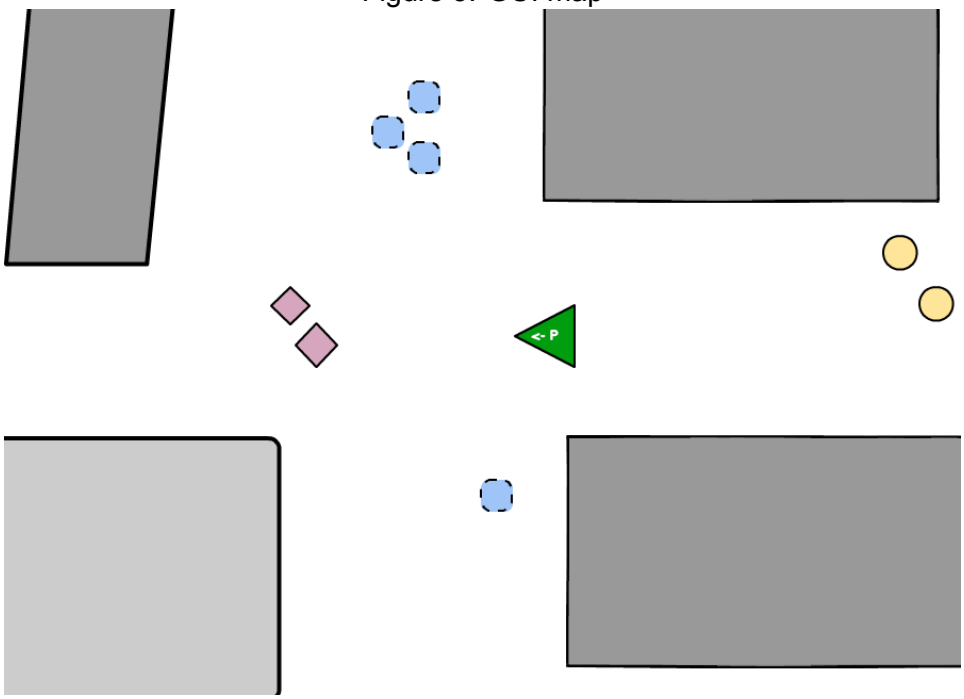
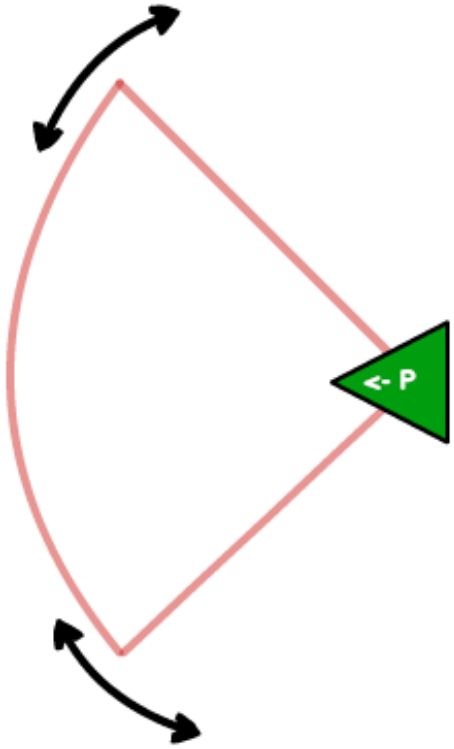


Figure 6: Proposed patient orientation design



The Patient's icon; the longest edges point to the direction its facing.

- The needed plugin/protocol with Unity to let the GUI communicate with the Unity world.
- The needed network protocol to connect host and client computers

2.2 Hardware/Software mapping

The tools we used are the standard input devices for any computer, namely the keyboard and mouse. An optional hardware that we thought we might implement is touchscreen integration. In the end we did not follow up on this plan. Furthermore, the system runs on 2 computers, one for the VR simulation and one to function as control panel for the therapist. These are connected via a local network.

2.3 Persistent data management (file/ database, database design)

The interact-able map is a viewer for the world that runs in Unity. Everything the viewer processes will be given to it at run-time. The specifics of the work being done by Unity are out-of-scope for this context project, and will not further be discussed in this document. This project will therefore not include persistent data management.

2.4 Concurrency

Each frame has just over 1 ms to render. We should limit the time needed for each frame to update the map/world as much as possible to keep the program running smoothly. We agreed on a time window of 1 ms to send and receive all our commands. We still have to determine our lag limit, but ideally something low enough that the user itself wont notice.

3 Final Implementation

We have managed to successfully implement the first 2 of the 3 phases of the project. The phases being:

- Build a static window using WPF and draw a static map onto it.
- Build a server/network layer that connects to unity to draw dynamic (movable) objects onto the map.
- Implement a way to send tasks to the unity world via the map.

Some shortcomings: the presentation of the map could have gotten more work. E.G vector graphics clearer orientation indicator. The network layer took us a considerate amount of time to get right, together with getting accustomed to the WPF framework. This and our miscalculated planned targets (such as more focus on code quality and targeting advanced tasks too soon) costs us valuable time. Implementing phase 3 was not above our capabilities. We have developed the necessary skills over the project and would have only needed more time to make it possible.

4 Glossary

CleVR VR development team at Yes!Delft, focused on virtual reality therapy solutions. `http://clevr.net/`. 3

Unity Unity is a game development platform, can be used to make 3D environments. `http://unity3d.com/unity` In this document this will sometimes be referred to as the VR world.. 5