Architecture Design

Context: Health Informatics

Group: HI1 a.k.a. Geen Naam (Group Number 4)

13-5-2016

Group Members:

Name:	NetID:	Studentnumber:
Jonas Dieterich	jdieterich	4317424
Maiko Goudriaan	maikogoudriaan	4302125
Ruben Meeuwissen	rmeeuwissen	4287150
Louis Sikkes	lsikkes	4298977
Thijmen Jaspers Focks	thijmendewilde	4158393

Table of Content:

Introduction

This document provides an approximate overview of the system that will be built during the context project in cooperation with CleVR. The application will be presented in a component based manner, including the division into sub-components if applicable.

Design goals:

The following design goals are essential for our project (Design Goals, n.d.):

Availability
At each end of the week's sprint we build a product that is ready to be used by the user. In this way we always have a working product to show to the user. The product has a minimised amount of possible failures. If there is a failure this is easy to fix, so the program will never have a long downtime.

Manageability
The product is easily manageable. The interface for the user is as straightforward as possible. You can click on an item and the possible actions of that item will appear. For more advanced options there is a manual, which explains how to change certain user preferences. The code of the product is well commented so developers who didn't work on the product, can read through the documentation and will understand the code.

Performance
The virtual reality world runs at 90 fps and should be fluent. The map that we're building should give the current overview of what is happening in this world and should also be fluent. Since we're giving commands to the world, and showing what is happening in there our map should be fluent. Since the plan of CleVR is to use our map as a basis to build other extensions on, like a notepad, if the performance is not good now than extending it is only going to make it worse. All actions should be short and easily processed to maintain high performance.

Reliability
Reliability is one of the most essential design goals due to the kind of medical simulation being run. The patient should at all times during the simulation be present in the virtual world and exposed to the virtual environment. Unexpected application termination could have serious implications for the mental state of the patient, as he might start panicing or reach a state of mental unease. Therefore, reliability will be prioritized at all times to meet the constraints of this health application.

Scalability
The product will at all times be scalable in the sense of being able to add new characters and interactive objects at will. Furthermore, new actions can be implemented and easily added in a modular approach. Moreover, additional features such as patient monitoring (e.g. heartbeat sensor) should be easily implementable, which will be supported by thorough documentation and strict code styling. This will allow the customer to expand his product from the point of handing it over.

Securability
Due to the secluded environment that this software is used in, there will be no major steps taken to secure the application. Furthermore, there is no remote possibility of disturbing the simulation due to the constraints of the set up (two computers interconnected by cable). Additionally, since no data is collected in the process, there is no need to be anticipate data theft. The only possibility of intervening in the procedure is by physically entering the building where the therapy session is taking place and tapping the cable between two computers. Securing the building is outside of our scope of making the application secure.

Software Architecture Views

The following chapter will discuss the architectural aspects of the application, by laying out the subcomponents and their interaction between them. The mapping between software and hardware as well as data management and concurrency will be discussed.

Subsystem decomposition

- Virtual World: The VR world is the environment that the patient will find itself in when using the VR goggles. It should be able to initialize the VR world and show it to the patient through the goggles. When given commands, like move person A to position X, it should execute these properly and update the world accordingly. It should also be able to send its current state to the therapist UI and should be able to respond to commands given by the therapist UI.

- VR Goggles: This is what the patient will use to see the virtual world. It should be attached to the virtual world and provide the patient with everything he or she needs to experience the situation.

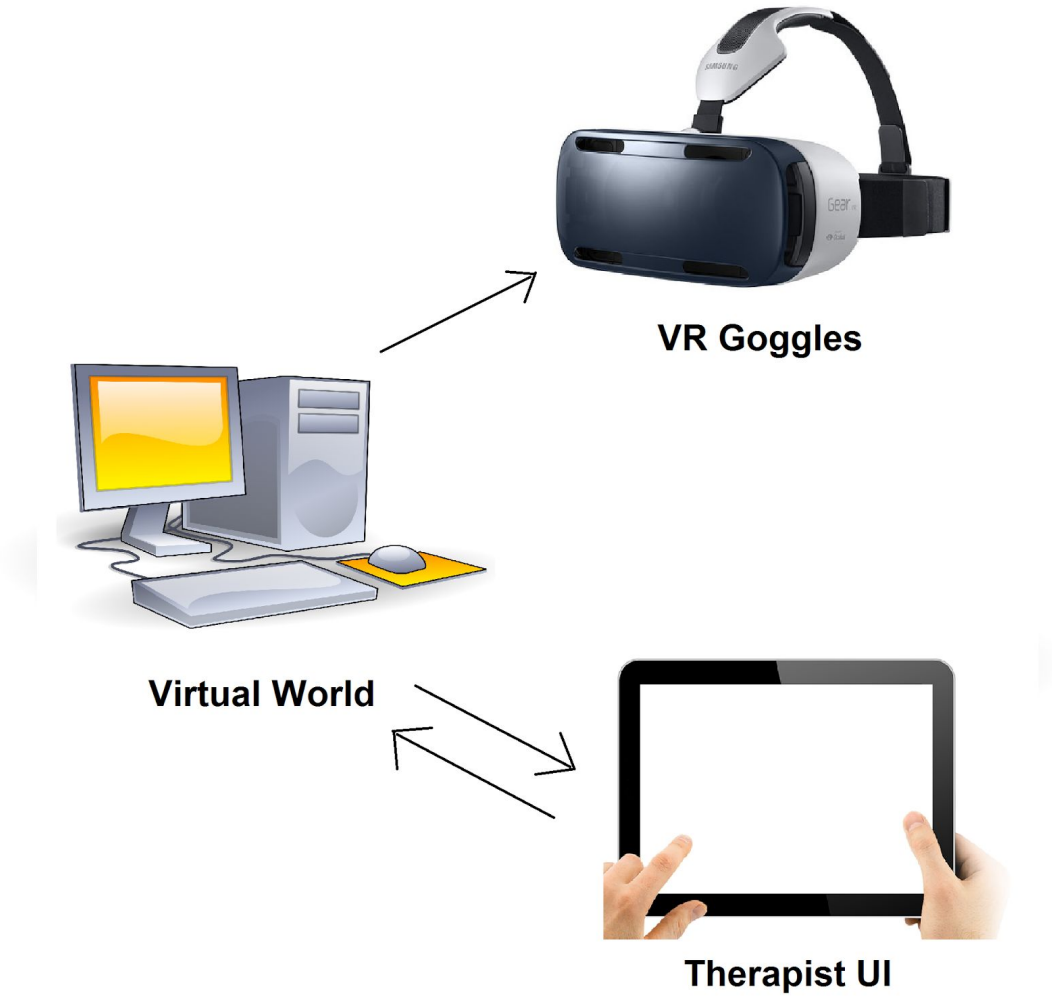
- Therapist UI (2D map): The therapist UI is a 2D map and is what the therapist sees and uses to modify the current VR world to their needs. This UI will run on a separate tablet / computer than the VR world. The map should display the current state of the

environment. The UI will receive the current state of the environment from the VR world and should make a clear view of it. The UI should also be able to give commands to the environment and display the changes that this commands makes.

Hardware/software mapping

- The communication between the VR-World based computer and the GUI based computer representing the 2D map will take place via the network. This will be implemented via a cable (Ethernet) connection.

- The communication between the VR goggles and the virtual world will take place through a USB cable.



Protocol for Data Exchange:

The protocol chosen to implement the data exchange between the two computers is UDP. The main reasons for this choice are motivated by the real-time constraint the application faces as well as the transmission medium used, the ethernet cable. TCP connections are required to set up a connection prior to exchanging data, which requires time and is an unnecessary overhead considering there are only two computers involved in the LAN. Furthermore, UDP allows for quick and independent transmission of packets, which in this specific environment are almost guaranteed to arrive, due to the safe transmission medium, an ethernet cable. There are only two senders, one of which (the therapist) will not transmit frequently, leading to low likelihood of collisions.

Persistent Data Management:

We receive information from the virtual world, which must be bound to the visual map. For example, emotions of the virtual characters in the virtual world are represented accordingly on the 2D map. The data is also sent in the other direction. Hence, if the therapist selects another emotion, this update is sent to the virtual world to update the characters emotion. The status information changes constantly as there are characters walking around at all times, thus data persistency only has to be insured in short intervals. This persistency is achieved by processing the incoming data packets and synchronizing it with the 2D or 3D representation.

Concurrency:

There are two main processes running concurrently during the simulation, one on the VR-world computer (to which the VR goggles are connected to) and one on the UI computer. The graphical representation of the virtual world is constantly being sent to the VR goggles (via cable) to display the simulation for the patient. Furthermore, the current state of the VR-world is sent to the UI computer (via network), hence sharing the resources of the VR-world with the UI computer. Furthermore, there is a data exchange occurring in the other direction, namely the issuing of commands from the UI computer to the VR-world.

Installation/ Setup Overview:

The project uses two main applications, the “Visualizer” (2D map) which is a WPF (XAML and C#) based application (main focus of the project), as well as a Unity based virtual environment, that is run to generate information for processing and display in the 2D map.

Required:

Operating system:

- Windows (recommended due to .NET framework)

Applications:

- Visual Studio (or other .NET IDEs, however due to tooling VS recommended!)
- Unity (Version 5.xx, to run the Virtual World)

Networking:

As soon as the Unity environment starts we create a server that sends out information about the environment. By clicking ctrl + c you can open the console to see which messages are being sent. On our map side we have a client (not yet invoked by application) that receives the messages from the environment and draws the appropriate object at the correct location.

Network Testing:

At this stage the client that receives the status information of the VR environment is not yet integrated into the Visualizer (2D map) application. Therefore we have a standalone application “Client.cs” that receives the UDP packets containing the status information via the ethernet interface. Use the mentioned tools to test the communication via virtual/or cable channel.

Tools for testing:

Wireshark recommended to listen to “virtual local host interface” or “ethernet interface”, prior for testing with one computer, latter for testing the connection using ethernet cable and two computers (simulating the real setup).

Interfaces to listen to:

- Ethernet interface (testing with cable and two computers)
- Ethernet localhost (testing with single computer)

Protocol (transport layer):

UDP used due to real-time constraint and safe transmission medium.

Coding Tools:

The following tools are used during development:

- GhostDoc
- NUnit (version 1.0 or higher)
- StyleCop (version 4.7 or higher)
- Visual Studio Spell Checker (current version)
- Productivity Power Tools 2013 (current version)
- CodeMaid
- GitHub

Organizational Tools

- Slack
- Trello
- Google Calendar

Logger

We have a logger to track the actions that we performed. We make use of an HTML file for this which you can open with any browser. Once an action is performed it is written to the current session logger file. We also have a logger file containing the current logger session file so that you can just refresh your browser to see what is going on now. We have a couple of different messages that are denoted by a different color like info, warning, error etc.

Design Patterns:

Singleton

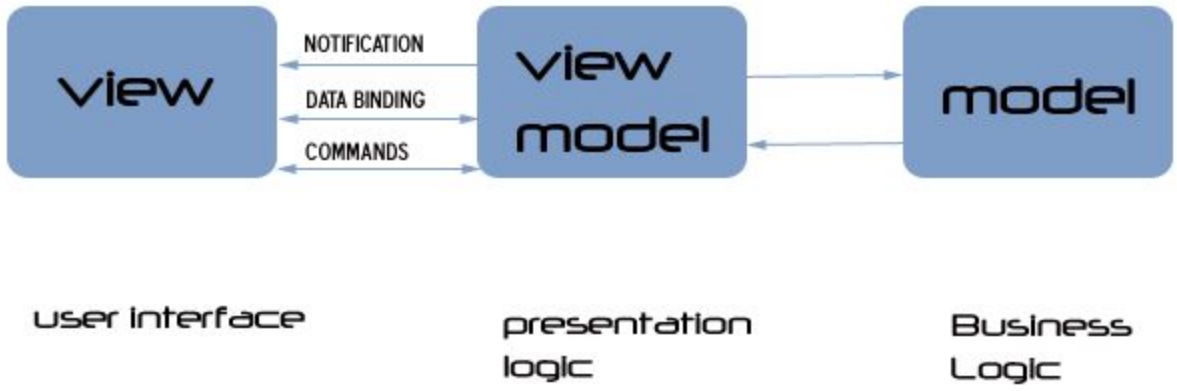
For the logger we use a singleton instance. This can be done since, there is only one instance of the logger active. The instance can easily be getted by all the classes.

MVVM pattern

We use the MVVM (Model View ViewModel) design pattern to keep the view, model and ViewModel separated for each other.
The view, as the name already says, does display the objects. They only draw the objects with the information, received from the ViewModel.
The ViewModel translates the model to the view. It receives data and puts this in the model, also it notifies the view to redraw the changed properties.
The Model stores the information about the object. It performs the most calculations, as we don't make that many calculations, those are mostly data classes.

To efficiently use the MVVM we created in the factories the view and the ViewModel. Those are linked. The view is send to the map, which add them to the canvas. The ViewModels are stored in the data handlers that handle the incoming data from network. Those divide the work the ViewModel. The ViewModel create a model instance. The ViewModel receives the incoming

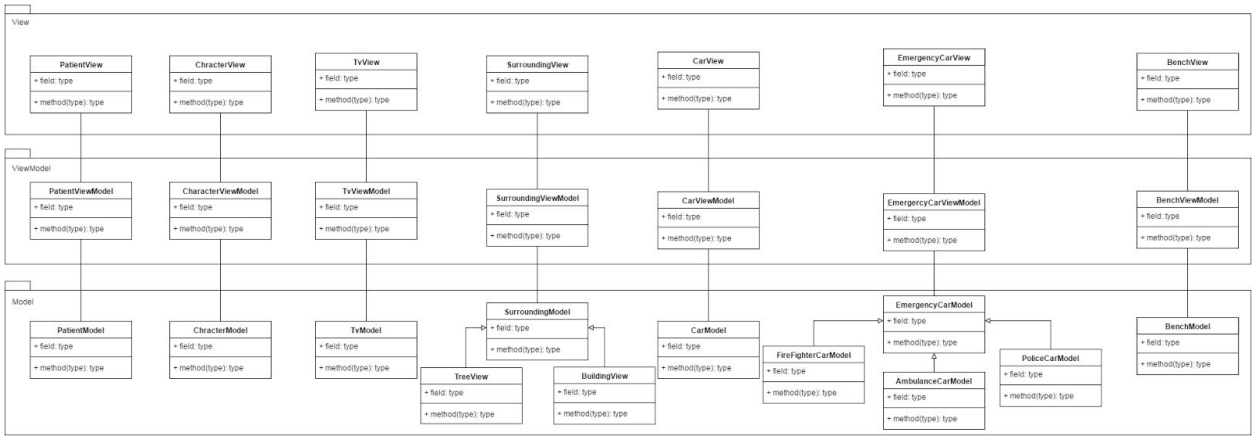
data and translates it to data that can be stored in the model. The model, if needed does his actions on the data. The ViewModel gets the value back from the model. It translates it now to a value that is needed for the View. It sends a property changed notification and the view updates the view.



Setting up the Unity VR World:

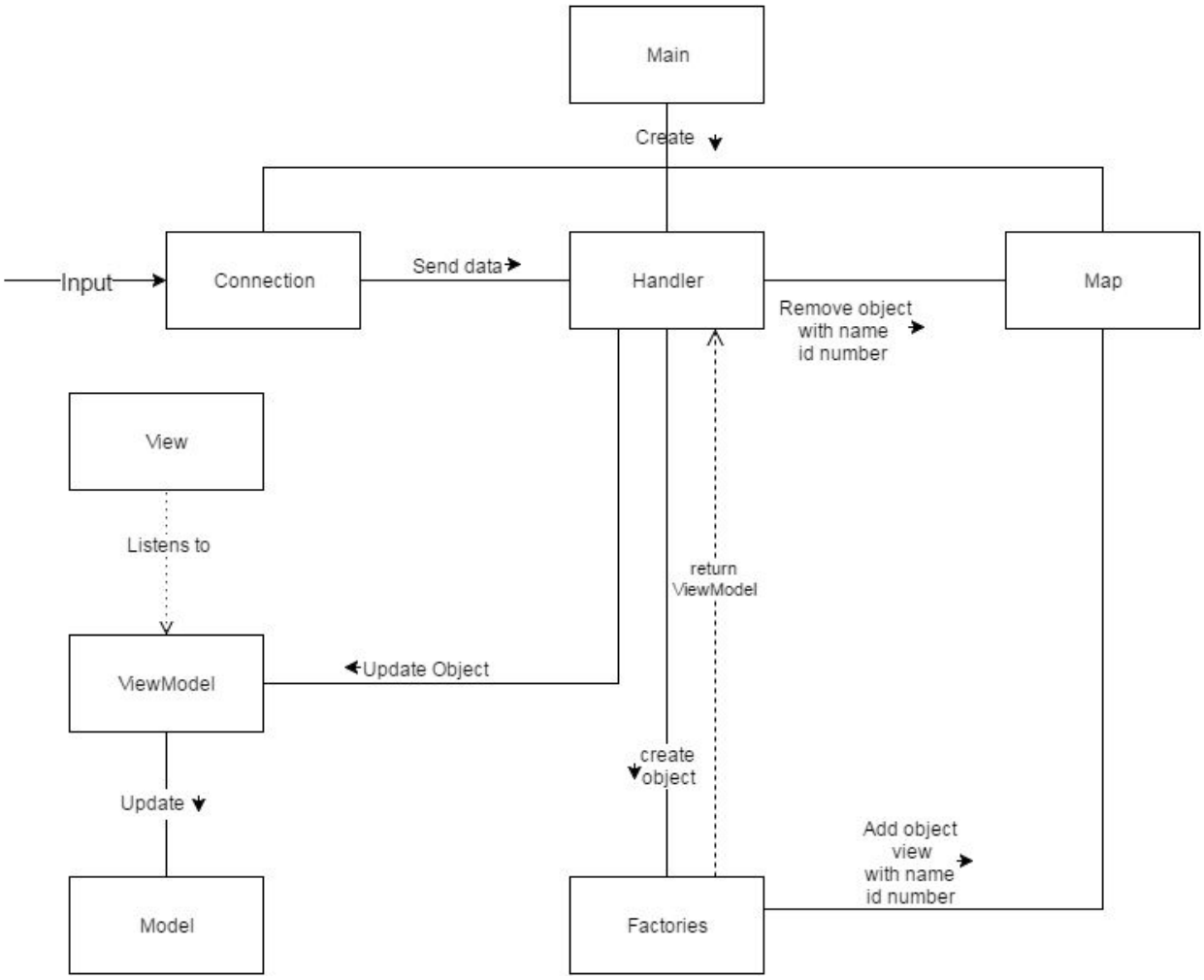
- Download and install Unity (free version suffices)
- Open project using the current VR Map from GitHub
- Create a new empty object
- Go to Assets/Network/ where the networking files are
- Drag and drop the desired file on the empty object that you just created
- By linking the script to the object, the script automatically gets invoked on start of the environment
- We do this to keep objects and semantics separated

MVVM UML



This is the MVVM UML. The image is also as raw recourse available in the GIT repo. The Model classes extends the AbstractModel. The ViewModel classes extends the AbstractViewModels. Those abstract classes have the attributes and methods that each object needs. The surrounding and emergency car needs a second base Model as there are different objects where images needs to be stored.

General Set-Up



This is the general overview of the program, also available in the git. The Main creates the connection, Handler and the map. The connection receives the data from the Environment and sends it to the Handlers. The handlers create the object through the factories or updates the object. The factories create the object, ViewModel and View and links them. It sends the view to the map, which at it on the canvas. The ViewModel is send back to the handler which stores it for further updates.

Glossary:

VR goggles - A virtual reality headset that allows its user to view and look around in the virtual world.
VR - Virtual reality
CIeVR - Company based in Delft working on Virtual Reality Therapy using interactive virtual worlds that clients are placed in and can interact with.
UI - User interface
GUI - Graphical user interface
UDP - User Datagram Protocol
TCP - Transmission Control Protocol

References:

- **Design Goals. (n.d.). Retrieved April 29, 2016, from https://msdn.microsoft.com/en-us/library/aa291862(v=vs.71).aspx**