Software Requirement Specification For A Graph Drawing Framework Prototype

by
Wilhelm Kerle

at
Computational Life Sciences
Department of Computer and Information Science
University of Konstanz

supervised by
Dr. Karsten Klein
Michael Aichem
Sabrina Jäger

April 19, 2020

Contents

1 Introduction

1.1 Overview

In this document, the first section describes the project in general. The second section explains the overall functionality, connected conditions and assumptions, and the particular use cases. The third section lists the requirement for the system towards the framework's prototype. Additionally, it notes the functionalities of that prototype.
In this document, *he* is deemed to include *she*, *it*, *them*, and other prepositions.

1.2 Purpose

The purpose of this document is to give an overview of the general idea of the bachelor's project. It describes the architecture that is behind the prototype itself and the overall requirements of the system.

1.3 Scope

The software shall represent a prototype of a framework. The prototype itself is a trimmed version of the mentioned framework. The prototype shall be able to assist researchers, students, and other groups of users who want to investigate a network in a virtual reality environment. The prototype provides the user with standard possibilities of visualization and interaction towards the loaded network. The prototype shall run in Unity. The user has to load the graph-file manually. A guide about how to use this framework is located in section 4.

1.4 Definitions, Acronyms, Abbreviations

Shortform		Description	meaning
IA		Immersive Analytics	Is the use of engaging, embodied analysis tools to support data understanding and decision making [5]
VR		Virtual Reality	Artificially, computer aided created virtual, interactive environment
HMD		Head Mounted Display	Is an (visual) output device, mounted on the head
Vive		HTC Vive	Virtual Reality Device (HMD)
Oculus		Oculus Rift	Virtual Reality Device (HMD)
-		Graph Drawing	Is a combination of methods from graph theory and information visualization to represent networks
UI		User Interface	Is a virtual element that inherits content

2 Overall Description

2.1 Prototype Purpose

The purpose of the prototype is to provide an outlook on how the future framework will operate. Further, it shall be a foundation for it. Hence, the prototype shall provide reliable codebase and basic usability and functionality.

2.2 User Characteristics

The users are mainly researchers and students. They have at least basic knowledge about graph theory. They don't struggle using Unity or VR environments. They want to visualize a network to attain knowledge about their data.
Potential users shall not use the prototype if they are suffering nausea during VR usage quickly.
There are two types of users:

- **Spectators:** They are using the software to investigate the characteristics of a network and only use the functionality at hand.
- **Contributors:** They are using the software to investigate the characteristics of a network and are able to and do develop new functionality and add them to the framework.

2.3 Prototype Design

The prototypes code structure has a modular scheme. That's due to the purpose of the prototype, as stated in section 2.1. The prototype supports the beforementioned user types (section 2.2), by providing basic interaction methods. Spectators will find that they can already load their graph into the environment, move around it, and apply pre-defined algorithms on that graph. The network file will have to be an adjacency list in .csv format.
Contributors can insert their algorithms, as long as their classes match given constraints (see section 2.7). They can mainly add new parsers, new analysis or visual algorithms, and new geometrical objects for nodes and edges, respectively (see section 4).
This design is supposed to ease the addition of new interaction methods with the graph and detach the addition of new functionality from the need to reiterate over a whole project. This design shortens the amount of time needed to implement immersive environments to inspect networks and therefore facilitates research. Lastly, the prototype shall support a reasonable level of embodiment, as described in the book *Immersive Analytics* in chapter 4 [5].

2.4 Prototype Environment

The Prototype is developed in Unity using version 2019.3.0f3 in c# and won't be provided as an executable standalone version in the scope of this project. It is developed and tested on a system with these specifications:

- **Prozessor:** AMD Ryzen 5 3400G 3.70 GHz
- **RAM:** 16.0 GB
- **System:** x64
- **Graphics:** NVIDIA GeForce GTX 1660 Ti

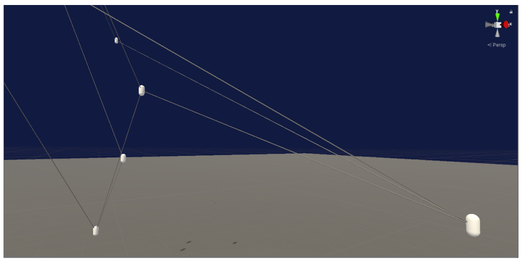
Additionally, the VR device used to test the framework is the Oculus. Also, SteamVR is used to process the inputs. SteamVR provides support for the Oculus and the Vive and is therefore best suited for a framework.

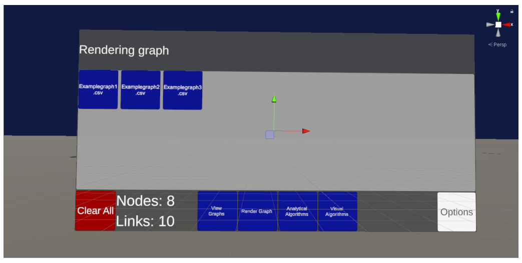
2.5 Prototype Functions

- The prototype shall visualize networks in VR.
- The prototype shall provide network analysis tools. It shall enable the user to view the number of nodes and links.
- The prototype shall enable contributors to add new algorithms, node representations, link representations, and parsers in a modular manner.
- The prototype represents its functionalities in the form of a canvas that can be opened by the user when pressing a pre-defined button
- The prototype shall use the virtual mouse ray-casting paradigm([2]) for interaction with the canvas

2.6 Prototype Appearance

The prototype shall have a clean design, to correspond to the scientific nature of its purpose. The skybox shall have a neutral color that allows distinguishing the graph to the background. There shall also be a floor to stand on. Its purpose is to improve immersion.
Further, there is a UI that contains visual and analytical algorithms, a status that indicates the current situation, and further options.





2.7 Constraints

2.7.1 System

Unity has to be used.
SteamVR is needed.
Users are encouraged to use setups similar to the one described in 2.4.

2.7.2 New file types

To be able to read new file types, a new parser has to be included. It is crucial that a piping function takes the output of the parser and constructs lists of nodes and lists of links. These lists will then be validated (see ch. 4.3.1).

2.7.3 New Algorithms

New visual algorithms, just as new analytical algorithms, can be added in the respective folders inhering the respective interfaces (see ch. 4.3).

2.7.4 New Nodes and Links

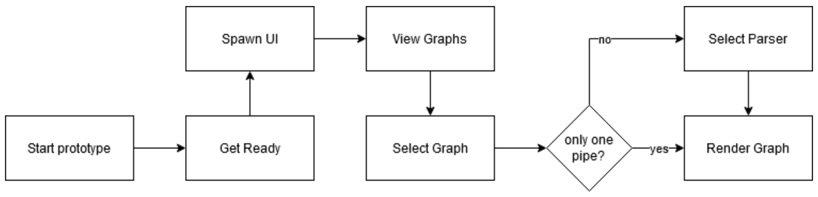
New node and link representations can be added in the respective *Resources* folders (see ch. 4.2).

2.8 Prototype Workflows - Spectator

For every workflow, it is important to start Unity and load up the prototype inside of it.

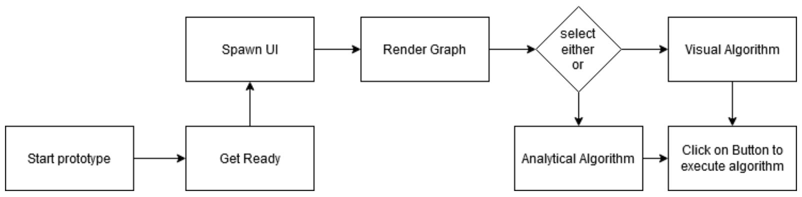
2.8.1 View Graph

Start the prototype. Put on HMD and pick up controllers. Summon UI (see 4.1.2). Click *View Graphs*. Select the graph. Click *Render Graph*.



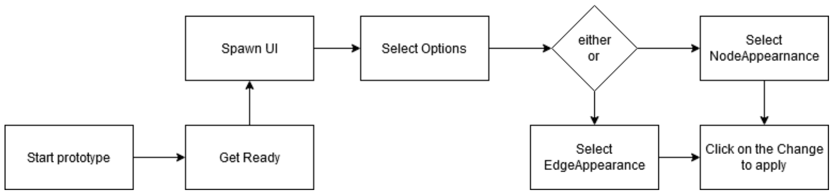
2.8.2 Use algorithm

Start the prototype. Put on HMD and pick up controllers. Summon UI (see 4.1.2). Render a graph (see 2.8.1). Click *Analytical Algorithm* or *Visual Algorithm*. Select routine to execute.



2.8.3 Alter appearance

Start the prototype. Put on HMD and pick up controllers. Summon UI (see 4.1.2). Select *Options*. Select *Node Appearance* or *Edge Appearance*. Select the change to make.



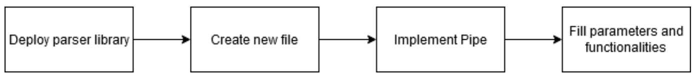
2.9 Prototype Workflows - Contributor

2.9.1 Add Prefab

Create new 3D game object. Navigate to link objects or node objects and insert the new prefab there. See 4.2 for more detail.

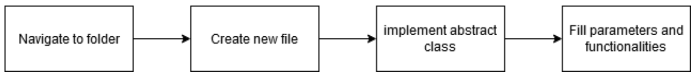
2.9.2 Add Parser

Navigate to the Libraries folder. Add the parser there. In same parent folder, navigate to Pipes. Create new file. Let it implement the abstract class *Pipe*. Fill necessary parameters and functions. For more detail see 4.3.1.



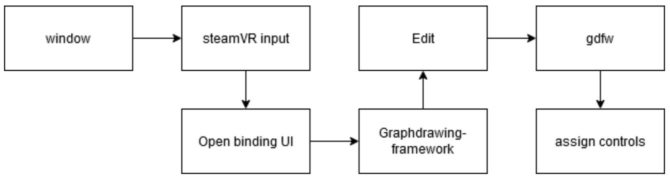
2.9.3 Add Algorithm

Navigate to analytics scripts or visual scripts in project, depending on which kind of algorithm is needed. Add new file there and let it implement the abstact class *AnalyticsScript* or *VisualScript*, respectively. Fill necessary parameters and functions. For more detail see 4.3.



2.9.4 Change controls

For this workflow, it is important to start Unity and load up the prototype inside of it. If the steamVR plugin is not yet installed, get it from the asset store.
In Unity: click window, then steamVR input. Click on *Open binding UI* and search for *Graph-drawingframework*. There click on edit and navigate to the gdfw tab. This is, where the prototype specific controls are located. Bind as required.



3 Specific Requirements

3.1 External interfaces

A VR ready PC with, at least, Unity 2019.3.0f3 installed. A Vive or an Oculus is required, as well. If adding additional code is planned, an IDE (MS Visual studio is strongly recommended!) is required to work with c# properly. Also, SteamVR must be active.

3.2 Functional requirements

- The prototype shall provide the user with a VR interface that contains the implemented algorithms and further settings. This reinforces immersion and provides a surface to dynamically add new content.

- The prototype shall provide a way to easily insert new graphs. This can, for example, be a pre-loaded button.

- The prototype shall accept minimalistic input nodes and links, to ensure that contributors do not have to implement additional complicated algorithms. It shall add missing, crucial information and omit invalid data.
- The prototype uses interfaces of specific algorithm types, to supply the necessary node and link information to the contributors, and identify algorithms.
- The prototype provides the possibility to use new types of custom node and link representation by receiving the geometries from a dedicated resources folder in the project.
- The prototype provides the possibility to choose, if links are rendered with actual geometrical objects or just with a line renderer.
- The prototype provides navigation through the representation in form of teleportation due to its simplicity. It might be replaced however, by the one-handed flying. This physical movement metaphor is the most intuitive one [3].

3.3 Framework requirements

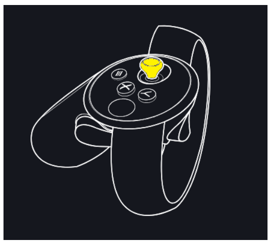
- **Maintainability** Modular expandability of functionality with as few constraints as possible, but as much as necessary.
- **Efficiency** Natively provided algorithms shall be as efficient as reasonably possible.
- **Stability** As long as the prototype is used as specified it shall always provide a response and terminate in a reasonable time.
- **Reliability** The prototype shall always terminate if the given constraints are matched.

4 Guide

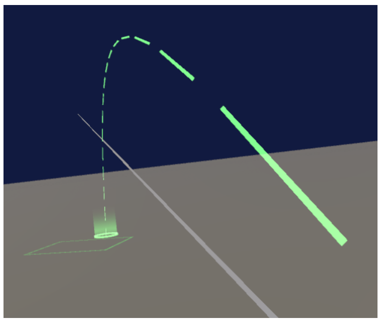
4.1 Navigation and Interaction

4.1.1 Navigation

Navigation happens mainly through teleportation. The default keybinding is the control stick.

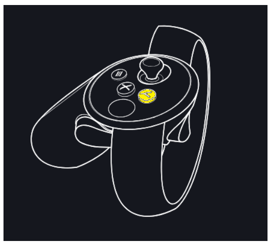


In VR, the user has to push it forward and keep it pushed to aim for where to go. A placement area and an arc indicate the new position. Further, simple, intuitive locomotion is also possible.

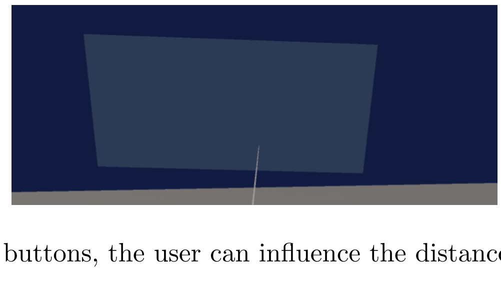


4.1.2 Interaction

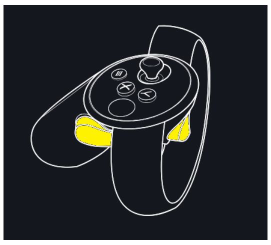
Interaction in the prototype consists of simple UI interaction. The user can summon the UI by pressing a pre-defined button. The standard configuration for that (on the Oculus) is the Y-Button or the B-Button, respectively.



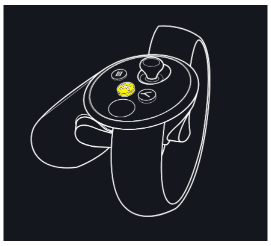
By keeping the button pressed, the user can first adjust the initial position of the UI.

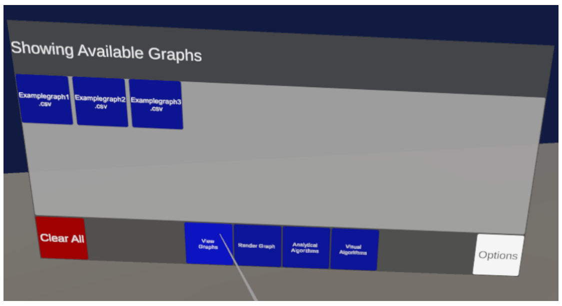


Using the trigger and grip buttons, the user can influence the distance between the interface and himself.



Using X or A, respectively, he can interact with the interactable elements inside of the UI and therefore trigger events.





4.2 Including Prefabs

4.2.1 Nodes

If some geometry is needed but not yet included (see figure 1), then navigate to *Assets/Resources/Nodeobjects* (see figure 2) and add a new geometry , for example a capsule (see figure 3). Now it should already be usable by the system (see figure 4). Hence, the system should also be able to render it (see figure 5).

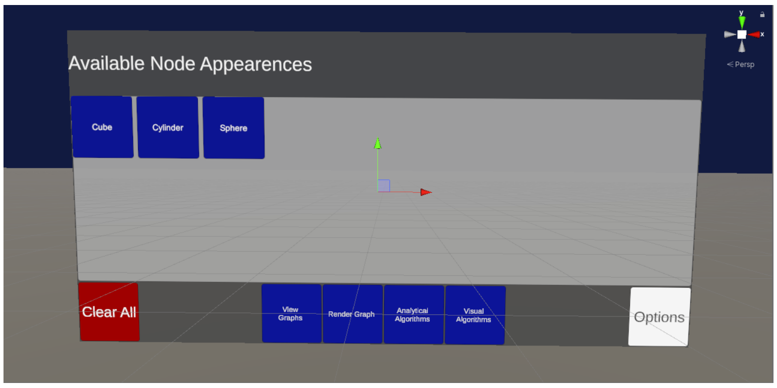


Figure 1: List of appearances of nodes without capsules.

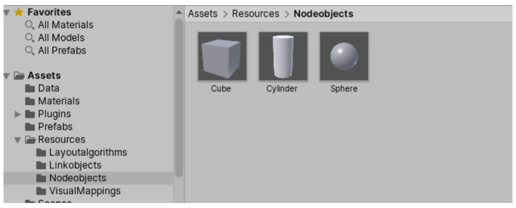


Figure 2: Folder, where prefabs are located.

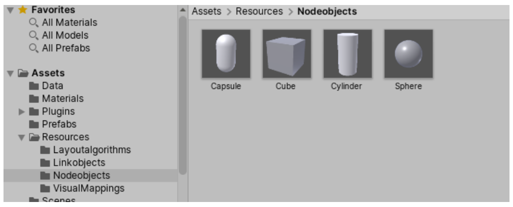


Figure 3: Adding the capsule geometry.

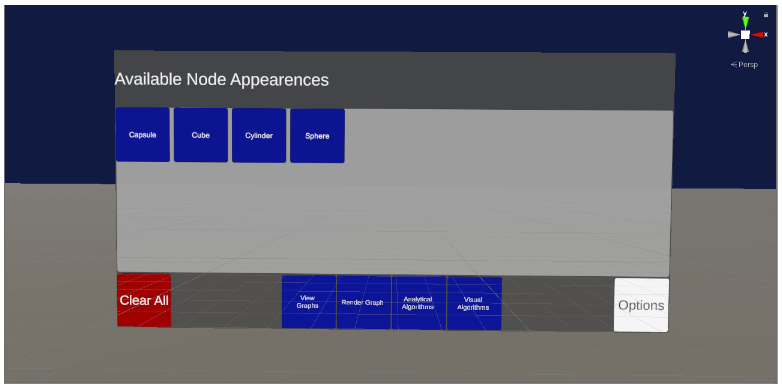


Figure 4: List of available node-appearances with capsules.

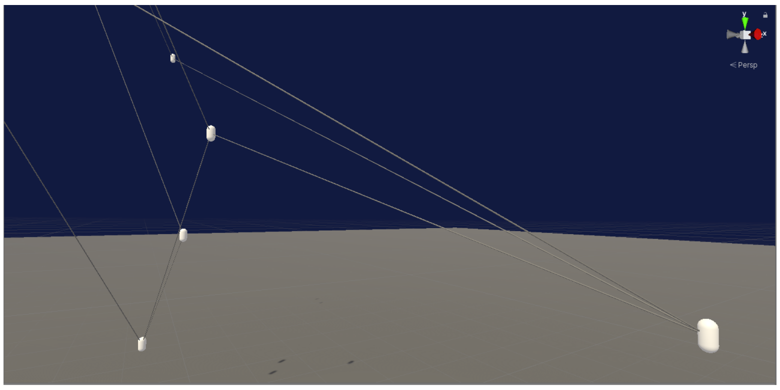


Figure 5: Rendered graph with newly added capsule prefab.

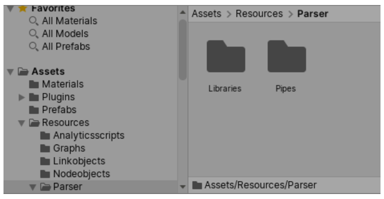
4.2.2 Links

Adding new representations for links is done analogously within the folder *Assets/Resources/Linkobjects*. Note, that the prototype is only optimized for big geometries right now. That means, that using a cube may result in only being half as long of an edge, for example.

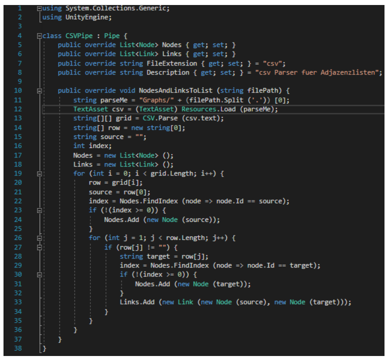
4.3 Including a new Script

4.3.1 Parser

To add a new parser, a contributor has to navigate to *Assets/Resources/Parser*.



The contributor adds the parser itself to the *Libraries* folder (this is only to maintain a better overview over which parsing libraries are implemented already). A new c# file has to be created in the *Assets/Resources/Parser/Pipes* folder.
The functions of this file work as a pipe, from the file to the framework. The new file has to implement the abstract class *Pipe*.



```
using System.Collections.Generic;
using UnityEngine;

class CSVPipe : Pipe {
    public override List<Node> Nodes { get; set; }
    public override List<Link> Links { get; set; }
    public override string FileExtension { get; set; } = "csv";
    public override string Description { get; set; } = "csv Parser fuer Adjazenzlisten";

    public override void NodesAndLinksToList (string filePath) {
        string parseMe = "Graphs/" + (filePath.Split ('.')) [0];
        TextAsset csv = (TextAsset) Resources.Load (parseMe);
        string[][] grid = CSV.Parse (csv.text);
        string[] row = new string[0];
        string source = "";
        int index;
        Nodes = new List<Node> ();
        Links = new List<Link> ();
        for (int i = 0; i < grid.Length; i++) {
            row = grid[i];
            source = row[0];
            index = Nodes.FindIndex (node => node.Id == source);
            if (!(index >= 0)) {
                Nodes.Add (new Node (source));
            }
            for (int j = 1; j < row.Length; j++) {
                if (row[j] != "") {
                    string target = row[j];
                    index = Nodes.FindIndex (node => node.Id == target);
                    if (!(index >= 0)) {
                        Nodes.Add (new Node (target));
                    }
                    Links.Add (new Link (new Node (source), new Node (target)));
                }
            }
        }
    }
}
```

This abstract class provides the contributor with some properties the contributor has to use. The lists of nodes and links have to be supplied with the parsed nodes as *Node* objects and links as *Link* objects. He has to define the file extension of the supported file type. Further, he has to define a description, which will be used by the framework, if more then one pipe is defined for the given file extension.
The function named *NodesAndLinksToList* has to trigger the assignment of nodes and links to the respective lists.

4.3.2 Visual Script

To add a new layout (see figure 6), first navigate to *Assets/Resources/Visualscripts* (see figure 7). Then, add a new c# script with the name of that functionality (that is the name it is displayed with, see figure 8). Now this new class must be using *globals* and be an abstract class of VisualScript (see figure 9). Then call up the hashtable from GlobalVariables you want to work with and execute the functionality. Note, that the contributor has to cast the rendered nodes and links to *Vertice* and *Edge* respectively. In the current state of the prototype, there is still a distinction of rendered objects and the ones inheriting information about the nodes and links internally. An update is currently handled by the prototype itself. The HasGuiButton boolean indicates, if the functionality shall be accessable in the UI. The execute function is the function that gets called upon button press. If the functionality has a UI button, the nodes and links will be set automatically by the prototype.
The algorithm should now appear on the panel for layout algorithms (see figure 10). Once clicked, it should execute the script that has been implemented (see figure 11).

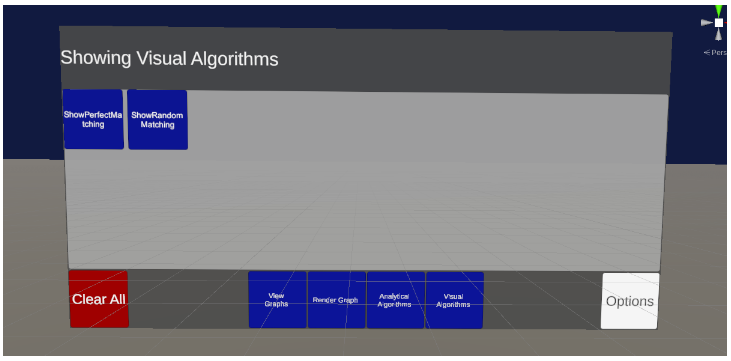


Figure 6: Layoutscript-panel, where the new algorithm can appear.

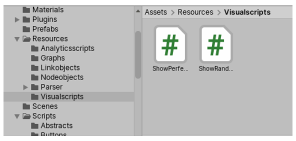


Figure 7: Folder, where visual algorithms get deployed.

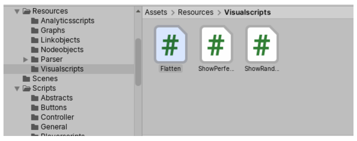
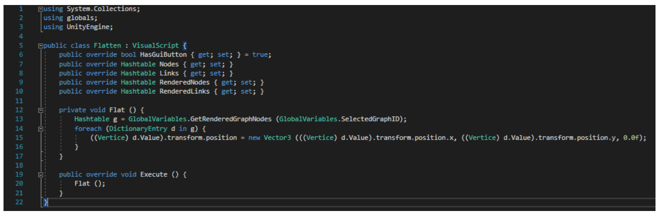


Figure 8: A new script has been added.



```
using System.Collections;
using globals;
using UnityEngine;

public class Flatten : VisualScript {
    public override bool HasGuiButton { get; set; } = true;
    public override Hashtable Nodes { get; set; }
    public override Hashtable Links { get; set; }
    public override Hashtable RenderedNodes { get; set; }
    public override Hashtable RenderedLinks { get; set; }

    private void Flat () {
        Hashtable g = GlobalVariables.GetRenderedGraphNodes (GlobalVariables.SelectedGraphID);
        foreach (DictionaryEntry d in g) {
            ((Vertice) d.Value).transform.position = new Vector3 (((Vertice) d.Value).transform.position.x, ((Vertice) d.Value).transform.position.y, 0.0f);
        }
    }

    public override void Execute () {
        Flat ();
    }
}
```

Figure 9: The necessary functions and variables have to be set.

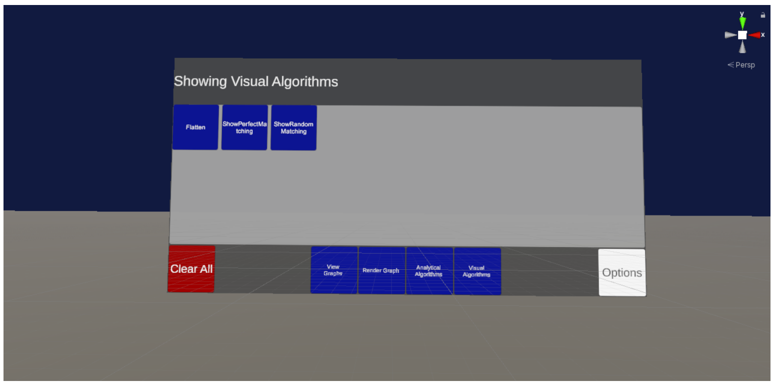


Figure 10: The new algorithm appearing in the respective panel.

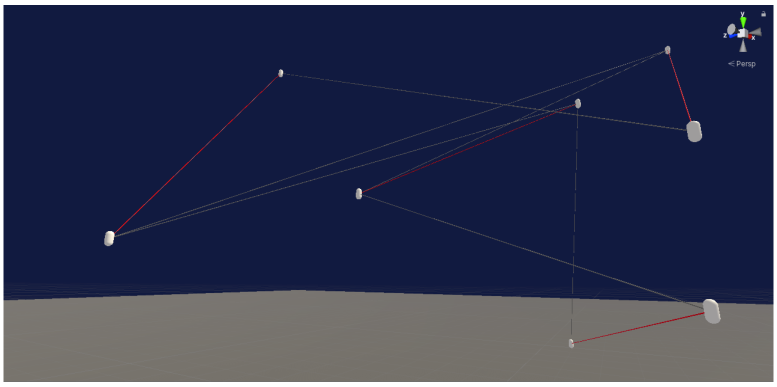


Figure 11: In this example, after interacting with the button, a perfect matching is highlighted with colors.

4.3.3 Analytical Script

The analytical scripts are analogous to the visual scripts. It additionally provides a hashtable, which holds string keys and values. These keys and values will be shown in the UI, if *HasResult-Button* is true. This functionality will be improved in the future.

5 Architecture

5.1 Component Structure

The component diagram in figure 12 depicts the structure of the prototype. Its main components are the data components, UI components, the framework logic, interfaces, and controller.

5.1.1 Data Components

The data components provide the framework logic and the UI components, with definitions of core objects, which are to be shown by the framework.

5.1.2 Framework Logic

The framework logic also transmits information to the UI components, since those ultimately trigger actions in the scene.
Inside the framework logic, the unified renderer component, and the validator component provide the global variables component with information about the actual data, additionally to data provided by the controller component. The global variables component contains information for all controller components.
The main objective of the *framework logic* is to handle information and present it in a valid format to the other elements of the prototype.

5.1.3 Interfaces

The interfaces determine the available algorithms controlled by the controller components. The pipe interface defines the skeleton of parsers and forwards the information of those parsers to the validator.
The *VisualScript* and the *AnalyticsScript* components provide the prototype with interfaces that serve as frames for algorithms, wanted in the prototype.

5.1.4 Controller

The controller components are handling the actual translation from interfaces into usable functionality in VR.

5.1.5 Diagram

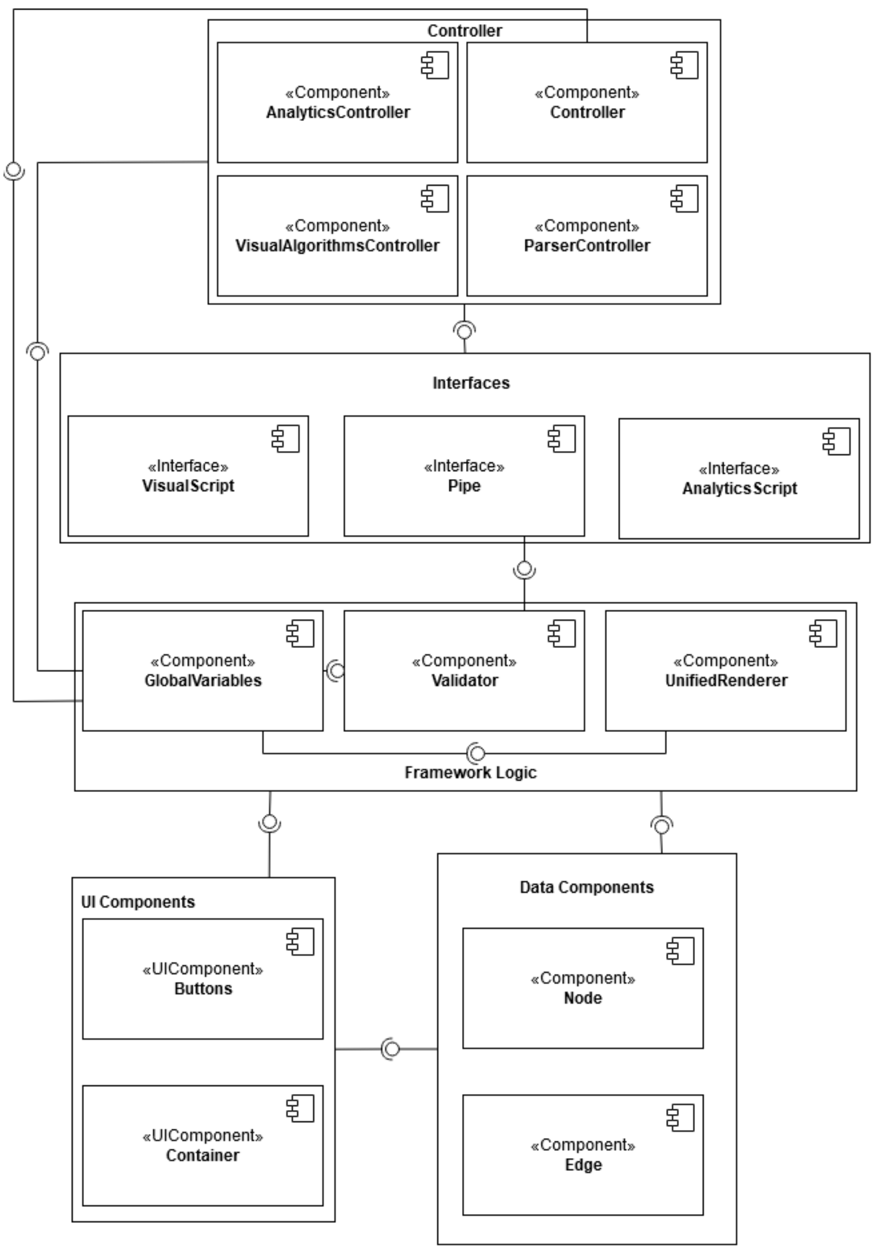


Figure 12: Component diagram of the prototype architecture

5.2 Class Structure

The class diagram in figure 13 depicts the relationships between the classes in the prototype. Its purpose is to give a rough overview of the implemented classes and their usage.
It consists of the sections interfaces, general, controller, player scripts, buttons, and renderable. The sections' names are similar to the names of the folders the classes are located.

5.2.1 Interfaces

VisualScript is an interface for classes, that perform algorithms and have the potential to interact with the scene.
AnalyticsScript is an interface for classes that perform algorithms and analyze graph data.
Both use the functionality of *GlobalVariables* to transmit graph information to it. *Pipe* is an interface for classes, which alter parser output to framework compatible data structures. It automatically transmits information to the validator class.

5.2.2 General

The validator class checks for missing crucial information. It takes the output from parsers and adds information to it that is necessary for the prototype to work. After validating the information, the validator forwards the new information to the *GlobalVariables* class.
The *GlobalVariables* provides information for different parts of the prototype and further inherits some functionality that gets used frequently. In the final framework, many of the features will be located in different, appropriate places.
The unified renderer takes the graph information from global variables and renders the scene. It also provides global variables with internal representations of the rendered objects for quick access.

5.2.3 Controller

The controller class provides global variables with lists of all available node and link styles.
The *AnalyticsController* uses the functionality of global variables to access the scripts that implement the *AnalyticsScript* abstract class. It then populates the prototypes UI with the actual functionality of found classes.
The *VisualAlgorithmsController* uses the functionality of global variables to access the scripts that implement the *VisualScript* abstract class. It then populates the prototypes UI with the actual functionality of found classes.
The *ParserController* uses the functionality of global variables to access the scripts that implement the *Pipe* abstract class. It then iterates over all files found in the resources folder for graphs and generates buttons for these graphs. If there is only one parser for the file extension that file has, then the parser controller generates a button and assigns that parser to it. If there is more than one parser available, then the parser controller generates a button, which then again opens another submenu. In this submenu, a button for each available parser appears with the parsers' description on it. The prototype parses the graph with the chosen parser.

5.2.4 Buttons

All buttons use information or functionality from the general classes.
ViewGraphs holds the function to show all available graphs.
ContainerViewPort is the class of the viewport in the UI that inherits most of the dynamic content of that UI.
NodeAppearance displays the different available node styles.

LinkAppearance displays the different available link styles.
RenderButton initiates the rendering of a chosen graph.
FWOptions shows and hides the actual options menu.
ToggleCustomEdge is a listener for the checkbox that determines, if a line renderer is used to depict edges.
ClearButton removes all nodes and links from the scene and the global variables.

5.2.5 Renderables

Vertices are the actual representation of nodes in the scene.
Nodes are the objects generated by the prototype as data representation. They will fall together with vertices in the future.
Edges are the actual representations of links in the scene.
Links are the objects generated by the prototype as data representation. They will fall together with Edges in the future.

5.2.6 Playerscripts

The pointer is the class generating the raycast in the scene.
VRInputModule handles the interaction of the raycast with the UI.
OpenUIScript handles the positioning of the UI in the scene.

5.2.7 Diagram

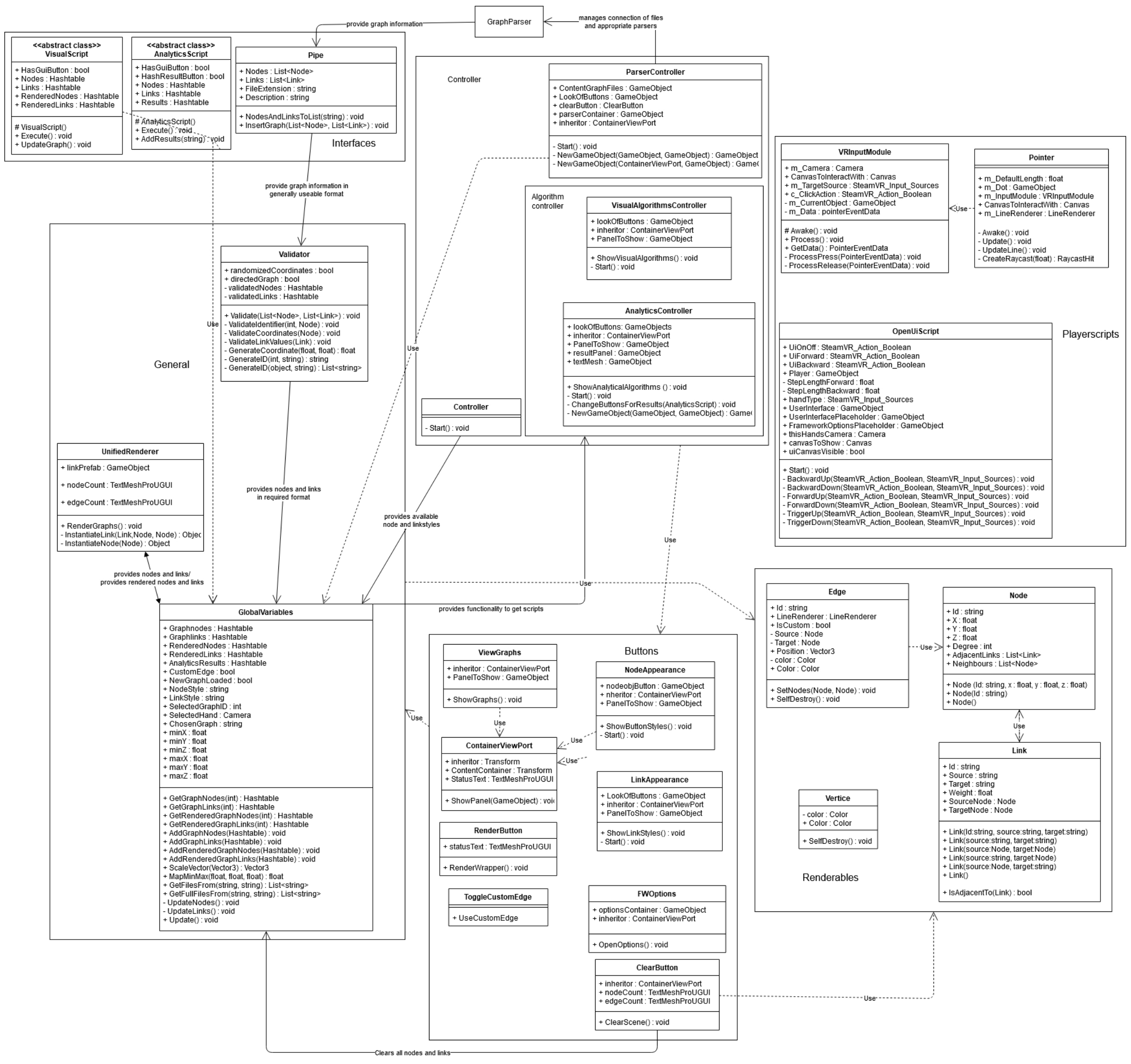

GraphParser
provide graph information
manages connection of files and appropriate parsers
<<abstract class>> VisualScript
<<abstract class>> AnalyticsScript
Pipe
Interfaces
Controller
ParserController
Algorithm controller
VisualAlgorithmsController
AnalyticsController
Controller
VRInputModule
Pointer
OpenUiScript
Playerscripts
provide graph information in generally useable format
Validator
General
UnifiedRenderer
provides nodes and links in required format
provides available node and linkstyles
provides nodes and links/ provides rendered nodes and links
GlobalVariables
provides functionality to get scripts
Use
Buttons
ViewGraphs
NodeAppearance
ContainerViewPort
LinkAppearance
RenderButton
ToggleCustomEdge
FWOptions
ClearButton
Edge
Node
Link
Vertice
Renderables
Clears all nodes and links

Figure 13: UML class diagram of overall prototype architecture WIP

References

[1] Google. Google earth vr, 2018

[2] S. Lee, J. Seo, G. J. Kim, and C. mo Park. Evaluation of pointing techniques for ray casting selection in virtual environments. In In Third International Conference on Virtual Reality and Its Application in Industry, pages 38–44, 2003.

[3] A. Drogemuller, A. Cunningham, J. Walsh, M. Cordeil, W. Ross and B. Thomas, "Evaluating Navigation Techniques for 3D Graph Visualizations in Virtual Reality," 2018 International Symposium on Big Data Visual and Immersive Analytics (BDVA), Konstanz, 2018, pp. 1-10.

[4] Marriott K. et al. (2018) Just 5 Questions: Toward a Design Framework for Immersive Analytics. In: Marriott K. et al. (eds) Immersive Analytics. Lecture Notes in Computer Science, vol 11190. Springer, Cham

[5] K. Marriott, F. Schreiber, T. Dwyer, K. Klein, N. H. Riche, T. Itoh, W. Stuerzlinger, B. H. Thomas, Immersive Analytics, Springer International Publishing, 2018.